TUGAS AKHIR - IF184802

STUDI KINERJA ALGORITMA BIDIRECTIONAL SEARCH PADA PENENTUAN RUTE PERGERAKAN PEMAIN DALAM MAXIMUM STEPS CONSTRAINT: STUDI KASUS PERMASALAHAN TIMUS ONLINE JUDGE 1589 - SOKOBAN

ADITYA PRATAMA
NRP 05111540000101

Dosen Pembimbing I
Rully Soelaiman, S.Kom., M.Kom.

Dosen Pembimbing II
M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil.

Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

TUGAS AKHIR – IF184802

STUDI KINERJA ALGORITMA BIDIRECTIONAL SEARCH PADA PENENTUAN RUTE PERGERAKAN PEMAIN DALAM MAXIMUM STEPS CONSTRAINT: STUDI KASUS PERMASALAHAN TIMUS ONLINE JUDGE 1589 - SOKOBAN

ADITYA PRATAMA
NRP 05111540000101

Dosen Pembimbing I
Rully Soelaiman, S.Kom., M.Kom.

Dosen Pembimbing II
M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil.

Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESIS – IF184802

PERFORMANCE STUDY OF BIDIRECTIONAL SEARCH ALGORITHM IN DETERMINATION OF PLAYER MOVEMENT ROUTES IN MAXIMUM STEPS CONSTRAINT: CASE STUDY ON TIMUS PROBLEM ONLINE JUDGE 1589 - SOKOBAN

ADITYA PRATAMA
NRP 05111540000101

First Advisor
Rully Soelaiman, S.Kom., M.Kom.

Second Advisor
M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil.

Department of Informatics Engineering
Faculty of Intelligent Electrical and Informatics Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2020

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

STUDI KINERJA ALGORITMA BIDIRECTIONAL SEARCH PADA PENENTUAN RUTE PERGERAKAN PEMAIN DALAM MAXIMUM STEPS CONSTRAINT: STUDI KASUS PERMASALAHAN TIMUS ONLINE JUDGE 1589 - SOKOBAN

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Algoritma dan Pemrograman
Program Studi S-1 Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Oleh:
ADITYA PRATAMA
NRP: 05111540000101

Disetujui oleh Pembimbing Tugas Akhir:

1. Rully Soelaiman, S.Kom., M.Kom. .....
   (NIP. 19700213 199402 1 001) (Pembimbing 1)

2. M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil. .....
   (NIP. 19740209 200212 1 001) (Pembimbing 2)

SURABAYA
Januari, 2020

(Halaman ini sengaja dikosongkan)

STUDI KINERJA ALGORITMA BIDIRECTIONAL SEARCH PADA PENENTUAN RUTE PERGERAKAN PEMAIN DALAM MAXIMUM STEPS CONSTRAINT: STUDI KASUS PERMASALAHAN TIMUS ONLINE JUDGE 1589 - SOKOBAN

Nama Mahasiswa	**: Aditya Pratama**
NRP	**: 05111540000101**
Jurusan	**: Informatika, Fakultas Teknologi Elektro dan Informatika Cerdas-ITS**
Dosen Pembimbing 1	**: Rully Soelaiman, S.Kom., M.Kom.**
Dosen Pembimbing 2	**: M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil.**

ABSTRAK

Dengan berkembangnya teknologi, penyelesaian suatu *game* juga dikembangkan ke arah yang lebih modern, utamanya untuk *game* dengan kompleksitas yang tinggi. Contoh dari jenis *game* ini adalah Sokoban. Tugas Akhir (TA) ini mengacu pada penyelesaian permasalahan pada *Timus Online Judge* dengan kode 1589[1] yang berjudul Sokoban, di mana diberikan sebuah *puzzle* yang merupakan map dua dimensi yang terdiri dari beberapa entitas yang direpresentasikan oleh beberapa karakter. Pendekatan penulis untuk menyelesaikan permasalahan tersebut adalah dengan menggunakan algoritma *bidirectional search* dengan algoritma A* sebagai *forward move* dan algoritma BFS (*Breadth-First Search*) sebagai *backward move*. Pendekatan *heuristic* untuk mendapatkan langkah optimal dalam penyelesaian *puzzle* pada sistem yang dibuat adalah *goal pull distance*. Analisis algoritma dan pendekatan *heuristic* lain yang mungkin bisa menyelesaikan permasalahan ini, juga dijelaskan lebih lanjut. Hasil TA ini belum mampu menyelesaikan secara penuh permasalahan di atas, namun sudah bisa menyelesaikan hingga *testcase* ke 55 dari 93 *testcase* yang tersedia, sehingga dari hasil ini bisa disimpulkan bahwa Tugas Akhir ini berhasil menyelesaikan sebagian besar *testcase*

yang diberikan oleh daring *Timus Online Judge*. Diharapkan dengan adanya TA ini dapat memberikan gambaran untuk pengembangan sistem berikutnya pada permasalahan yang sama atau yang serupa.

Kata kunci: *bidirectional search, graph, game theory, sokoban.*

PERFORMANCE STUDY OF BIDIRECTIONAL SEARCH ALGORITHM IN DETERMINATION OF PLAYER MOVEMENT ROUTES IN MAXIMUM STEPS CONSTRAINT: CASE STUDY ON TIMUS PROBLEM ONLINE JUDGE 1589 - SOKOBAN

Student's Name : Aditya Pratama
Student's ID : 05111540000101
Department : Informatics Engineering, Faculty of Intelligent Electrical and Informatics Technology -ITS
First Advisor : Rully Soelaiman, S.Kom., M.Kom.
Second Advisor : M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil.

ABSTRACT

*With the development of technology, the completion of a game is also developed in a more modern direction, especially for games with high complexity. An example of this type of game is Sokoban. This Final Project (TA) refers to solving a problem in the Timus Online Judge with code 1589[1] entitled Sokoban, where given a puzzle which is a two-dimensional map consisting of several entities represented by several characters. The author's approach to solving this problem is to use the bidirectional search algorithm with the A * algorithm as a forward move and the BFS (Breadth-First Search) algorithm as a backward move. The heuristic approach to get the optimal step in solving the puzzle on the system created is the goal pull distance. Analysis of algorithms and other heuristic approaches that might solve this problem are also further explained. The results of this TA have not been able to fully solve the above problems, but have been able to finish up to the 55th testcase out of 93 testcases, so from this result it can be concluded that this Final Project successfully completed the majority of the testcases given by the online Timus Online Judge. It is hoped that the existence of this TA can provide an overview for subsequent system development on the same or similar problems.*

Keywords: ***bidirectional search, graph, game theory, sokoban.***

KATA PENGANTAR

Puji syukur saya sampaikan kepada Tuhan yang Maha Esa karena berkat rahmat-Nya saya dapat melaksanakan Tugas Akhir yang berjudul:

"STUDI KINERJA ALGORITMA BIDIRECTIONAL SEARCH PADA PENENTUAN RUTE PERGERAKAN PEMAIN DALAM MAXIMUM STEPS CONSTRAINT: STUDI KASUS PERMASALAHAN TIMUS ONLINE JUDGE 1589 - SOKOBAN"

Terselesaikannya Tugas Akhir ini tidak terlepas dari bantuan dan dukungan banyak pihak, oleh karena itu melalui lembar ini penulis ingin mengucapkan terima kasih dan penghormatan kepada:

1. Allah SWT, karena dengan limpahan rahmat dan karunia-Nya penulis dapat menyelesaikan Tugas Akhir dan juga perkuliahan di Departemen Teknik Informatika ITS.
2. Kedua orangtua penulis, dan anggota keluarga lainnya yang telah memberikan dukungan doa, moral, dan material kepada penulis sehingga penulis dapat menyelesaikan Tugas Akhir ini.
3. Rully Soelaiman, S.Kom., M.Kom. dan M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil. selaku pembimbing I dan II yang telah membimbing dan memberikan motivasi, nasihat dan bimbingan dalam menyelesaikan Tugas Akhir ini.
4. Dr. Eng. Darlis Herumurti, S.Kom., M.Kom. selaku Ketua Departemen Teknik Informatika ITS dan seluruh dosen dan karyawan Departemen Teknik Informatika ITS yang telah memberikan ilmu dan pengalaman kepada penulis selama menjalani masa kuliah di Departemen Teknik Informatika ITS.

5. Nuzul Ristyantika, Frieda Uswatun Hasanah, Irsyad Rizaldi, dan Achmad Ibnu Malik Al Chasni, dari komunitas "Lab Mulyos" yang telah memberikan dukungan dan motivasi kepada penulis dalam kegiatan perkuliahan, khususnya selama 4 tahun pertama masa perkuliahan penulis.
6. Reinardus dan Bagus Dharma, yang telah menjadi teman penulis dalam mengerjakan Tugas Akhir pada semester ini.
7. Tim "ElDorado" dan "| || || |_" yang merupakan teman seperjuangan penulis dalam kontes pemrograman.
8. Teman-teman Hexavara, yang telah memberikan ilmu tambahan disela kesibukan penulis dalam mengerjakan Tugas Akhir ini.
9. Seluruh mahasiswa Informatika ITS angkatan 2015 yang telah menjadi teman penulis selama menjalani masa kuliah di Departemen Teknik Informatika ITS.
10. Serta semua pihak yang telah turut membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis menyadari bahwa laporan Tugas Akhir ini masih memiliki banyak kekurangan. Oleh karena itu dengan segala kerendahan hati penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depannya. Selain itu, penulis berharap laporan Tugas Akhir ini dapat berguna bagi pembaca secara umum.

Surabaya, Januari 2020

Aditya Pratama

DAFTAR ISI

LEMBAR PENGESAHAN ..... **vii**
ABSTRAK ..... **ix**
ABSTRACT ..... **xi**
KATA PENGANTAR ..... **xiii**
DAFTAR ISI ..... **xv**
DAFTAR TABEL ..... **xxiii**
DAFTAR KODE SUMBER ..... **xxv**
DAFTAR GAMBAR ..... **xxxiii**
BAB I PENDAHULUAN **1**
1.1 Latar Belakang 1
1.2 Rumusan Masalah 2
1.3 Batasan Permasalahan 2
1.4 Tujuan 3
1.5 Manfaat 3
1.6 Metodologi 3
1.6.1 Penyusunan Proposal Tugas Akhir 3
1.6.2 Studi Literatur 3
1.6.3 Implementasi Perangkat Lunak 4
1.6.4 Pengujian dan Evaluasi 4
1.6.5 Penyusunan Buku 4
1.7 Sistematika Penulisan Laporan 4
BAB II DASAR TEORI **7**
2.1 Deskripsi Permasalahan 7
2.1.1 Parameter Input 8
2.1.2 Batasan Permasalahan 9
2.1.3 Output Permasalahan 9
2.2 Sokoban 9
2.2.1 Entitas Sokoban 9
2.2.2 Aturan Permainan Sokoban 10
2.3 Terminologi 11
2.3.1 Entitas 11
2.3.2 Indeks 12

2.3.3 Representasi Bit 13
2.3.4 *Node* 14
2.4 Strategi Penyelesaian Masalah 15
2.4.1 *Bidirectional Search* 16
2.4.2 Optimasi *Node* 23
2.4.3 *Pruning* 26
2.4.4 *Macro Moves* 28
BAB III PERANCANGAN SISTEM 31
3.1 Deskripsi Umum Sistem 31
3.2 Penjelasan Sistem 31
3.3 Desain Struktur *Node* 31
3.3.1 Desain Fungsi Heuristic Value 32
3.3.2 Desain Atribut *Push* 37
3.3.3 Desain Atribut *State* 37
3.4 Desain Mark State 38
3.5 Desain *Path Graph* 39
3.5.1 Desain Fungsi Get Path Map Key 40
3.5.2 Desain Fungsi Get Path Map Value 40
3.5.3 Desain Fungsi Mark Path 41
3.5.4 Desain Fungsi Split Path Map Value 41
3.5.5 Desain Fungsi Get Full Path 42
3.6 Pemangkasan State (*Pruning*) 44
3.6.1 Desain Deteksi *Simple Deadlock* 44
3.6.2 Desain Deteksi *Freeze Deadlock* 47
3.7 Desain Fungsi Validasi 49
3.8 Desain Fungsi Global 51
3.9 Desain Solusi Utama 53
3.10 Desain Fungsi Run Input 54
3.10.1 Desain Fungsi Init Puzzle 54
3.11 Desain Variabel Global 55
3.11.1 Desain Batasan Global 56
3.11.2 Desain Arah Konstan Global 56
3.11.3 Desain Ukuran Global 57
3.11.4 Desain Arah Mata Angin Global 57
3.11.5 Desain Representasi Bit Global 58

3.12 Desain Fungsi Init Global Variabels 58
3.13 Desain Fungsi Reinit Walls 60
3.14 Desain Fungsi Generate Tunnels 61
3.14.1 Desain Fungsi Is Horizontal Tunnel 63
3.14.2 Desain Fungsi Is Vertical Tunnel 63
3.15 Desain Tabel Jarak 64
3.15.1 Desain Fungsi Init Table 64
3.15.2 Desain Fungsi Create Distance Table 66
3.16 Desain Fungsi Init Forward Global Variables 66
3.17 Desain Fungsi Init Backward Global Variables 68
3.18 Desain Fungsi Add Initial Node 70
3.18.1 Desain Fungsi Add Forward Initial Node 70
3.18.2 Desain Fungsi Add Backward Initial Node 71
3.19 Desain Fungsi Run Forward Move 73
3.19.1 Desain Fungsi Generate Forward Box Successors . 75
3.19.2 Desain Check Meet In The Middle untuk Forward Move 77
3.20 Desain Fungsi Run Backwad Move 80
3.20.1 Desain Fungsi Generate Backward Box Successors 81
3.20.2 Desain Fungsi Check Meet In The Middle untuk Backward Move 83
3.21 Desain Solusi Alternatif 86
3.21.1 Desain Bidirectional Search DFS dan BFS 86
3.21.2 Desain Bidirectional Search Greedy DFS dan BFS 86
BAB IV IMPLEMENTASI 87
4.1 Lingkungan Implementasi 87
4.2 Implementasi Program Utama 87
4.2.1 Penggunaan *Library* 87
4.2.2 Penggunaan Preprocessor Directives 88
4.2.3 Penggunaan Reserved Keyword 88
4.2.4 Implementasi Fungsi Main 89
4.3 Implementasi Fungsi Struktur *Node* 89
4.3.1 Implementasi Fungsi Heuristic Value 89
4.3.2 Implementasi Struct Node Compare 93

4.4 Implementasi Mark State.....93
4.5 Implementasi *Path Graph* 93
4.5.1 Implementasi Fungsi Get Path Map Key 94
4.5.2 Implementasi Fungsi Get Path Map Value 94
4.5.3 Implementasi Fungsi Mark Path 94
4.5.4 Implementasi Fungsi Split Path Map Value 95
4.5.5 Implementasi Fungsi Get Full Path.....96
4.6 Implementasi Deteksi Deadlock.....97
4.6.1 Implementasi Fungsi Is Simple Deadlock.....98
4.6.2 Implementasi Fungsi Is Freeze Deadlock 99
4.7 Implementasi Fungsi Validasi 100
4.7.1 Implementasi Fungsi Is Box 100
4.7.2 Implementasi Fungsi Is Deadlock.....101
4.7.3 Implementasi Fungsi Is Goal 101
4.7.4 Implementasi Fungsi Is Goals.....101
4.7.5 Implementasi Fungsi Is Game Over.....102
4.7.6 Implementasi Fungsi Is Game Over Without Move 102
4.7.7 Implementasi Fungsi Is Lock.....103
4.7.8 Implementasi Fungsi Is Obstacle 103
4.7.9 Implementasi Fungsi Is On Same Column 103
4.7.10 Implementasi Fungsi Is On Same Row.....104
4.7.11 Implementasi Fungsi Is State Seen 104
4.7.12 Implementasi Fungsi Is Tunnel.....104
4.7.13 Implementasi Fungsi Is Valid Index 105
4.7.14 Implementasi Fungsi Is Valid Index Move.....105
4.7.15 Implementasi Fungsi Is Wall 106
4.8 Implementasi Fungsi Global.....106
4.8.1 Implementasi Fungsi Clear Template 106
4.8.2 Implementasi Fungsi Memset Template 106
4.8.3 Implementasi Fungsi Assign To Vector.....107
4.8.4 Implementasi Fungsi Get Index Moved.....107
4.8.5 Implementasi Fungsi Get New Node 108
4.8.6 Implementasi Fungsi Get Shortest Path 108
4.8.7 Implementasi Fungsi Get Boxes Moved.....110

4.8.8 Implementasi Fungsi Assign Winds Direction 111
4.8.9 Implementasi Fungsi Convert To Vector 111
4.8.10 Implementasi Fungsi Generate Successors 112
4.9 Implementasi Fungsi Run Input 113
4.10 Implementasi Fungsi Reinit Walls 113
4.11 Implementasi Fungsi Generate Tunnels 114
4.11.1 Implementasi Fungsi Is Horizontal Tunnel 115
4.11.2 Implementasi Fungsi Is Vertical Tunnel 115
4.12 Implementasi Tabel Jarak 116
4.12.1 Implementasi Fungsi Init Table 116
4.12.2 Implementasi Fungsi Create Distance Table 118
4.13 Implementasi Fungsi Solve 118
4.13.1 Implementasi Fungsi Init Global Variables 119
4.13.2 Implementasi Fungsi Add Initial Node 122
4.13.3 Implementasi Fungsi Run Forward Move 125
4.13.4 Implementasi Fungsi Run Backward Move 130
4.14 Implementasi Solusi Alternatif 135
4.14.1 Implementasi Bidirectional Search DFS dan BFS 135
4.14.2 Implementasi Bidirectional Search Greedy DFS dan BFS 136
BAB V UJI COBA DAN EVALUASI 139
5.1 Lingkungan Uji Coba 139
5.2 Uji Coba Kebenaran 139
5.2.1 Uji Coba Kebenaran Lokal 139
5.2.2 Uji Coba Kebenaran Situs Penilaian Daring Timus Online Judge 141
5.2.3 Uji Coba Kinerja Lokal 143
5.2.4 Data Uji Coba Kinerja Lokal 143
5.2.5 Skenario Uji Coba Kinerja Lokal 143
5.2.6 Evaluasi Uji Coba Kinerja Lokal 143
5.2.7 Uji Coba Kinerja Situs Penilaian Daring Timus Online Judge 154
5.3 Analisis dan Kesimpulan Umum 160
BAB VI KESIMPULAN DAN SARAN 163
6.1 Kesimpulan 163

6.2 Saran.....164
DAFTAR PUSTAKA 165
LAMPIRAN A: Hasil Uji Coba pada Situs *Timus Online Judge* Sebanyak 10 Kali 167
Tabel A1 Hasil Uji Coba Program Utama (*Bidirectional Search* A* dan BFS).....167
Tabel A2 Hasil Uji Coba Program Tambahan (*Bidirectional Search* DFS dan BFS) 168
Tabel A3 Hasil Uji Coba Program Tambahan (*Bidirectional Search Greedy* DFS dan BFS).....169
Tabel A4 Hasil Uji Coba Program Tambahan (*Manhattan Heuristic*).....170
Tabel A5 Hasil Uji Coba Program Tambahan (*Euclidean Heuristic*).....171
LAMPIRAN B: Data Uji.....173
Tabel B1 Data Uji Coba Koleksi *Puzzle Microban* (1).....173
Tabel B2 Data Uji Coba Koleksi *Puzzle Microban* (2).....174
Tabel B3 Data Uji Coba Koleksi *Puzzle Microban* (3).....175
Tabel B4 Data Uji Coba Koleksi *Puzzle Microban* (4).....176
Tabel B5 Data Uji Coba Koleksi *Puzzle Microban* (5).....177
Tabel B6 Data Uji Coba Koleksi *Puzzle Microban* (6).....178
Tabel B7 Data Uji Coba Koleksi *Puzzle Microban* (7).....179
Tabel B8 Data Uji Coba Koleksi *Puzzle Microban* (8).....180
Tabel B9 Data Uji Coba Koleksi *Puzzle Microban* (9).....181
Tabel B10 Data Uji Coba Koleksi *Puzzle Microban* (10) 182
Tabel B11 Data Uji Coba Koleksi *Puzzle Microcosmos* (1) 182
Tabel B12 Data Uji Coba Koleksi *Puzzle Microcosmos* (2) 183
Tabel B13 Data Uji Coba Koleksi *Puzzle Microcosmos* (3) 184
Tabel B14 Data Uji Coba Koleksi *Puzzle Minicosmos* (1) 184

Tabel B15 Data Uji Coba Koleksi *Puzzle Minicosmos* (2) 185
Tabel B16 Data Uji Coba Koleksi *Puzzle Minicosmos* (3) 186
Tabel B17 Data Uji Coba Koleksi *Puzzle Minicosmos* (4) 187
Tabel B18 Data Uji Coba Koleksi *Puzzle Nabokosmos* (1) 188
Tabel B19 Data Uji Coba Koleksi *Puzzle Nabokosmos* (2) 189
Tabel B20 Data Uji Coba Koleksi *Puzzle Sasquatch5* 189
Tabel B21 Data Uji Coba Koleksi *Puzzle Sokogen* (1) ...190
Tabel B22 Data Uji Coba Koleksi *Puzzle Sokogen* (2) ...191
Tabel B23 Data Uji Coba Koleksi *Puzzle Sokogen* (3) ...192
Tabel B24 Data Uji Coba Koleksi *Puzzle Sokogen* (4) ...193
Tabel B25 Data Uji Coba Koleksi *Puzzle Sokogen* (5) ...194
Tabel B26 Data Uji Coba Koleksi *Puzzle Sokogen* (6) ...195
Tabel B27 Data Uji Coba Koleksi *Puzzle Sokogen* (7) ...196
Tabel B28 Data Uji Coba Koleksi *Puzzle Sokogen* (8) ...197
Tabel B29 Data Uji Coba Koleksi *Puzzle Yoshio* (1).....197
Tabel B30 Data Uji Coba Koleksi *Puzzle Yoshio* (2).....198
Tabel B31 Data Uji Coba Koleksi *Puzzle Yoshio* (3).....199
Tabel B32 Data Uji Coba Koleksi *Puzzle Yoshio* (4).....200
Tabel B33 Data Uji Coba Koleksi *Puzzle Yoshio* (5).....201
Tabel B34 Data Uji Coba Koleksi *Puzzle Yoshio* (6).....202
Tabel B35 Data Uji Coba Koleksi *Puzzle Yoshio* (7).....203
Tabel B36 Data Uji Coba Koleksi *Puzzle Yoshio* (8).....204
Tabel B37 Data Uji Coba Koleksi *Puzzle Yoshio* (9).....205
Tabel B38 Data Uji Coba Koleksi *Puzzle Yoshio* (10)....206
Tabel B39 Data Uji Coba Koleksi *Puzzle Yoshio* (11)....207
Tabel B40 Data Uji Coba Koleksi *Puzzle Yoshio* (12)....208
Tabel B41 Data Uji Coba Koleksi *Puzzle Yoshio* (13)....209
Tabel B42 Data Uji Coba Koleksi *Puzzle Yoshio* (14)....210
Tabel B43 Data Uji Coba Koleksi *Puzzle Hard Case* (1)211
Tabel B44 Data Uji Coba Koleksi *Puzzle Hard Case* (2)212

Tabel B45 Data Uji Coba Koleksi *Puzzle Hard Case* (3)213
BIODATA PENULIS 215

DAFTAR TABEL

Tabel 2.1 Terminologi Entitas Sokoban 12
Tabel 2.2 Perhitungan Representasi Bit 14
Tabel 2.3 Atribut *Node* 15
Tabel 3.1a Tabel Fungsi Validasi 50
Tabel 3.1b Tabel Fungsi Validasi 51
Tabel 3.2 Tabel Fungsi Global 52
Tabel 3.3 Tabel Variabel Global Berjenis Batasan Global 56
Tabel 3.4 Tabel Variabel Global Berjenis Arah Konstan Global 57
Tabel 3.5 Tabel Variabel Global Berjenis Ukuran Global . 57
Tabel 3.6 Tabel Variabel Global Berjenis Representasi Bit Global 58
Tabel 3.7 Tabel Variabel pada *Forward Move* 67
Tabel 3.8 Tabel Variabel pada *Backward Move* 69
Tabel 5.1 Tabel Data Uji Coba Kebenaran Lokal dengan Data Sampel 140
Tabel 5.2 Hasil Uji Coba Kinerja Lokal untuk Tiap Variasi Algoritma *Bidirectional Search* 144
Tabel 5.3 Perbandingan Jumlah *State* Tiap Variasi Algoritma *Bidirectional Search* 145
Tabel 5.4 Hasil Uji Coba Tiap Variasi Algoritma *Bidirectional Search Dataset Hard Case* 146
Tabel 5.5 Perbandingan Jumlah *State* Tiap Variasi Algoritma *Bidirectional Search Dataset Hard Case* 147
Tabel 5.6 Konfigurasi *Push* Tiap Algoritma *Dataset Hard Case* 147
Tabel 5.7 Hasil Uji Coba Kinerja Lokal untuk Tiap Variasi Algoritma *Bidirectional Search* Menggunakan Konfigurasi *Push* 148
Tabel 5.8 Perbandingan Jumlah *State* Tiap Variasi Algoritma *Bidirectional Search* Menggunakan Konfigurasi *Push* 149

Tabel 5.9 Hasil Uji Coba Kinerja Lokal untuk Tiap *Heuristic* 150
Tabel 5.10 Grafik Perbandingan Jumlah *State* Tiap Variasi *Heuristic* 151
Tabel 5.11 Hasil Uji Coba Tiap Variasi Heurtistik *Dataset Hard Case* 152
Tabel 5.12 Grafik Perbandingan Jumlah *State* Tiap Variasi *Heuristic Dataset Hard Case* 152
Tabel 5.13 Konfigurasi *Push* Tiap Heuristic *Dataset Hard Case* 153
Tabel 5.14 Hasil Uji Coba Kinerja Lokal untuk Tiap Variasi *Heuristic* Menggunakan Konfigurasi *Push* 153
Tabel 5.15 Grafik Perbandingan Jumlah *State* Tiap Variasi *Heuristic* Menggunakan Konfigurasi *Push* 154
Tabel 5.16 Tabel Data Waktu Hasil Uji Coba Kinerja Variasi Algoritma *Bidirectional Search* Situs Penilaian Daring Timus Online Judge 155
Tabel 5.17 Tabel Data Memori Hasil Uji Coba Kinerja Variasi Algoritma *Bidirectional Search* Situs Penilaian Daring Timus Online Judge 156
Tabel 5.18 Tabel Data Waktu Hasil Uji Coba Kinerja Variasi *Heuristic* Situs Penilaian Daring Timus Online Judge 158
Tabel 5.19 Tabel Data Memori Hasil Uji Coba Kinerja Variasi *Heuristic* Situs Penilaian Daring Timus Online Judge 159

DAFTAR KODE SUMBER

Kode Sumber 3.1 Struktur *Node Graph* Pencarian 32
Kode Sumber 3.2 *Pseudocode* Fungsi Heuristic Value 33
Kode Sumber 3.3a *Pseudocode* Fungsi Greedy Bipartite Heuristic 34
Kode Sumber 3.3b *Pseudocode* Fungsi Greedy Bipartite Heuristic 35
Kode Sumber 3.4 Deklarasi Tipe Data BGEDGE 35
Kode Sumber 3.5 *Pseudocode* Fungsi Search Box Goals 36
Kode Sumber 3.6 *Pseudocode* Fungsi Box Goal Edge Compare 37
Kode Sumber 3.7 Variabel-Variabel yang Menyimpan *State* 39
Kode Sumber 3.8 *Pseudocode* Fungsi Mark State 39
Kode Sumber 3.9 Variabel-Variabel *Path Graph* 40
Kode Sumber 3.10 *Pseudocode* Fungsi Get Path Map Key 40
Kode Sumber 3.11 *Pseudocode* Fungsi Get Path Map Value 40
Kode Sumber 3.12 *Pseudocode* Fungsi Mark Path 41
Kode Sumber 3.13 *Pseudocode* Fungsi Split Path Map Value 42
Kode Sumber 3.14 *Pseudocode* Fungsi Get Full Path 43
Kode Sumber 3.15 *Pseudocode* Fungsi Is Simple Deadlock 45
Kode Sumber 3.16 *Pseudocode* Fungsi Generate Deadlock 45
Kode Sumber 3.17 *Pseudocode* Fungsi Expand Safe Indices 47
Kode Sumber 3.18 *Pseudocode* Fungsi Is Freeze Deadlock 49
Kode Sumber 3.19 *Pseudocode* Fungsi Run Input 54
Kode Sumber 3.20 *Pseudocode* Fungsi Init Puzzle 55

Kode Sumber 3.21 *Pseudocode* Fungsi Get Row Normalized 55
Kode Sumber 3.22a *Pseudocode* Fungsi Init Global Variables 59
Kode Sumber 3.22b *Pseudocode* Fungsi Init Global Variables 60
Kode Sumber 3.23 *Pseudocode* Fungsi Reinit Walls 61
Kode Sumber 3.24 *Pseudocode* Fungsi Generate Tunnels .62
Kode Sumber 3.25 *Pseudocode* Fungsi Is Horizontal Tunnel 63
Kode Sumber 3.26 *Pseudocode* Fungsi Is Vertical Tunnel 63
Kode Sumber 3.27 *Pseudocode* Fungsi Init Table 65
Kode Sumber 3.28 *Pseudocode* Fungsi Create Distance Table 66
Kode Sumber 3.29 *Pseudocode* Fungsi Init Forward Global Variables 68
Kode Sumber 3.30 *Pseudocode* Fungsi Init Backward Global Variables 70
Kode Sumber 3.31 *Pseudocode* Fungsi Add Initial Node ..70
Kode Sumber 3.32 *Pseudocode* Fungsi Add Forward Initial Node 71
Kode Sumber 3.33 *Pseudocode* Fungsi Add Backward Initial Node 72
Kode Sumber 3.34 *Pseudocode* Fungsi Run Forward Move 74
Kode Sumber 3.35a *Pseudocode* Fungsi Generate Forward Box Successors 76
Kode Sumber 3.35b *Pseudocode* Fungsi Generate Forward Box Successors 77
Kode Sumber 3.36 *Pseudocode* Check Meet In The Middle untuk Forward Move 79
Kode Sumber 3.37a *Pseudocode* Fungsi Run Backward Move 80
Kode Sumber 3.37b *Pseudocode* Fungsi Run Backward Move 81

Kode Sumber 3.38a *Pseudocode* Fungsi Generate Backward Box Successors 82
Kode Sumber 3.38b *Pseudocode* Fungsi Generate Backward Box Successors 83
Kode Sumber 3.39 *Pseudocode* Check Meet In The Middle untuk Backward Move 85
Kode Sumber 4.1 *Header* Program Utama 88
Kode Sumber 4.2 *Preprocessor Directives* Program Utama 88
Kode Sumber 4.3 *Reserved Keyword* Program Utama 89
Kode Sumber 4.4 Implementasi Fungsi Main 89
Kode Sumber 4.5 Implementasi Fungsi Heuristic Value .90
Kode Sumber 4.6a Implementasi Fungsi Greedy Bipartite Heuristic 90
Kode Sumber 4.6b Implementasi Fungsi Greedy Bipartite Heuristic 91
Kode Sumber 4.7 Implementasi Fungsi Search Box Goals 92
Kode Sumber 4.8 Implementasi Fungsi Bgedge Comp 92
Kode Sumber 4.9 Implementasi Fungsi Operator Node Compare 93
Kode Sumber 4.10 Implementasi Fungsi Mark State 93
Kode Sumber 4.11 Implementasi Fungsi Get Path Map Key 94
Kode Sumber 4.12 Implementasi Fungsi Get Path Map Value 94
Kode Sumber 4.13 Implementasi Fungsi Mark Path 95
Kode Sumber 4.14 Implementasi Fungsi Split Path Map Value 96
Kode Sumber 4.15 Implementasi Fungsi Get Full Path 97
Kode Sumber 4.16 Implementasi Fungsi Is Simple Deadlock 98
Kode Sumber 4.17 Implementasi Fungsi Generate Deadlock 98

Kode Sumber 4.18 Implementasi Fungsi Expand Safe Indices 99
Kode Sumber 4.19 Implementasi Fungsi Is Freeze Deadlock 100
Kode Sumber 4.20 Implementasi Fungsi Is Box 101
Kode Sumber 4.21 Implementasi Fungsi Is Deadlock 101
Kode Sumber 4.22 Implementasi Fungsi Is Goal 101
Kode Sumber 4.23 Implementasi Fungsi Is Goals 102
Kode Sumber 4.24 Implementasi Fungsi Is Game Over ..102
Kode Sumber 4.25 Implementasi Fungsi Is Game Over Without Move 102
Kode Sumber 4.26 Implementasi Fungsi Is Lock 103
Kode Sumber 4.27 Implementasi Fungsi Is Obstacle 103
Kode Sumber 4.28 Implementasi Fungsi Is On Same Column 103
Kode Sumber 4.29 Implementasi Fungsi Is On Same Row 104
Kode Sumber 4.30 Implementasi Fungsi Is State Seen 104
Kode Sumber 4.31 Implementasi Fungsi Is Tunnel 104
Kode Sumber 4.32 Implementasi Fungsi Is Valid Index ..105
Kode Sumber 4.33 Implementasi Fungsi Is Valid Index Move 105
Kode Sumber 4.34 Implementasi Fungsi Is Wall 106
Kode Sumber 4.35 Implementasi Fungsi Clear Template 106
Kode Sumber 4.36 Implementasi Fungsi Memset Template 107
Kode Sumber 4.37 Implementasi Fungsi Assign To Vector 107
Kode Sumber 4.38 Implementasi Fungsi Get Index Moved 108
Kode Sumber 4.39 Implementasi Fungsi Get New Node .108
Kode Sumber 4.40a Implementasi Fungsi Get Shortest Path 109
Kode Sumber 4.40b Implementasi Fungsi Get Shortest Path 110

Kode Sumber 4.41 Implementasi Fungsi Get Boxes Moved 111
Kode Sumber 4.42 Implementasi Fungsi Assign Winds Direction 111
Kode Sumber 4.43 Implementasi Fungsi Convert To Vector 112
Kode Sumber 4.44 Implementasi Fungsi Generate Successors 112
Kode Sumber 4.45 Implementasi Fungsi Run Input 113
Kode Sumber 4.46 Implementasi Fungsi Reinit Walls 114
Kode Sumber 4.47 Implementasi Fungsi Generate Tunnels 115
Kode Sumber 4.48 Implementasi Fungsi Is Horizontal Tunnel 115
Kode Sumber 4.49 Implementasi Fungsi Is Vertical Tunnel 116
Kode Sumber 4.50 Implementasi Fungsi Init Table 117
Kode Sumber 4.51 Implementasi Fungsi Create Distance Table 118
Kode Sumber 4.52 Implementasi Fungsi Solve 119
Kode Sumber 4.53a Implementasi Fungsi Init Global Variables 120
Kode Sumber 4.53b Implementasi Fungsi Init Global Variables 121
Kode Sumber 4.54 Implementasi Fungsi Init Forward Global Variables 121
Kode Sumber 4.55 Implementasi Fungsi Init Backwars Global Variables 122
Kode Sumber 4.56 Implementasi Fungsi Add Initial Node 122
Kode Sumber 4.57 Implementasi Fungsi Add Forward Initial Node 123
Kode Sumber 4.58a Implementasi Fungsi Add Backward Initial Node 123

Kode Sumber 4.58b Implementasi Fungsi Add Backward Initial Node 124
Kode Sumber 4.59a Implementasi Fungsi Run Forward Move 125
Kode Sumber 4.59b Implementasi Fungsi Run Forward Move 126
Kode Sumber 4.60a Implementasi Fungsi Generate Forward Box Successors 126
Kode Sumber 4.60b Implementasi Fungsi Generate Forward Box Successors 127
Kode Sumber 4.60c Implementasi Fungsi Generate Forward Box Successors 128
Kode Sumber 4.61a Implementasi Fungsi Check Meet In The Middle untuk *Forward Move* 129
Kode Sumber 4.61b Implementasi Fungsi Check Meet In The Middle untuk *Forward Move* 130
Kode Sumber 4.62a Implementasi Fungsi Run Backward Move 130
Kode Sumber 4.62b Implementasi Fungsi Run Backward Move 131
Kode Sumber 4.63a Implementasi Fungsi Generate Backward Move Box Successors 132
Kode Sumber 4.63b Implementasi Fungsi Generate Backward Move Box Successors 133
Kode Sumber 4.63c Implementasi Fungsi Generate Backward Move Box Successors 134
Kode Sumber 4.64a Implementasi Fungsi Check Meet In The Middle untuk *Backward Move* 134
Kode Sumber 4.64b Implementasi Fungsi Check Meet In The Middle untuk *Backward Move* 135
Kode Sumber 4.65 Variabel *Graph* Pencarian *Forward Move* pada Solusi *Bidirectional Search* DFS dan BFS 136

Kode Sumber 4.66 Implementasi Modifikasi Fungsi Heuristic Value pada Solusi *Bidirectional Search* DFS dan BFS 136
Kode Sumber 4.67 Implementasi Modifikasi Fungsi Run Forward Move pada Solusi *Bidirectional Search* Greedy DFS dan BFS 137
Kode Sumber 4.68 Implementasi Fungsi Successor Comp pada Solusi *Bidirectional Search* Greedy DFS dan BFS 137

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

Gambar 2.1 Contoh Standar Input Program 7
Gambar 2.2 Contoh Standar Output Program 8
Gambar 2.3 Contoh *Puzzle* Sokoban dan Entitas-Entitasnya 9
Gambar 2.4 Aturan *Action Push* pada Sokoban 11
Gambar 2.5 Visualisasi *State* 12
Gambar 2.6 *Index* pada Sebuah *State* 13
Gambar 2.7 *Bidirectional Search* menggunakan A* dan BFS 17
Gambar 2.8 Ilustrasi A* 18
Gambar 2.9 Ilustrasi DFS 18
Gambar 2.10 Ilustrasi *Greedy* DFS 19
Gambar 2.11 Penjelasan Kondisi *Meet in The Middle* 20
Gambar 2.12 Visualisasi Pemeriksaan *Meet in The Middle* . 21
Gambar 2.13 Visualisasi *Goal Pull Distance* 23
Gambar 2.14 Pembuatan *Node* dengan Cara Naif 24
Gambar 2.15 Pembuatan *Node* dari *Action Push* 24
Gambar 2.16 Ilustrasi *Path Graph* 25
Gambar 2.17 Contoh *Simple Deadlock* 26
Gambar 2.18 Contoh *Freeze Deadlock* 27
Gambar 2.19 Contoh *Bipartite Deadlock* 27
Gambar 2.20 Contoh *Corral Deadlock* 28
Gambar 2.21 Ilustrasi *Macro Tunnel Moves* 29
Gambar 3.1 Ilustrasi *Bipartite Matching* 32
Gambar 3.2 Visualisasi Indeks pada Sebuah *State* yang Merujuk Posisi *Box* 38
Gambar 3.3 *Simple Deadlock* 45
Gambar 3.4 Pencarian *Safe Indices* 46
Gambar 3.5 *Freeze Deadlock* 47
Gambar 3.6 *Freeze Deadlock* yang Dideteksi Oleh Sistem 48
Gambar 3.7 *Flowchart* Sistem 53
Gambar 3.8 Variabel Global Berjenis Arah Mata Angin Global 58
Gambar 3.9 Ilustrasi Proses Reinisialisasi *Walls* 60

Gambar 3.10 Ilustrasi Proses Penentuan *State* Baru dengan Bantuan *Tunnel* 62
Gambar 3.11 Jarak Antara Dua Indeks pada Sebuah *State* ... 64
Gambar 5.1 Hasil Uji Coba Kebenaran Situs Penilaian Daring Timus Online Judge 142
Gambar 5.2 Jumlah *Testcase* oleh *Problem Setter* 142
Gambar 5.3 Kinerja Hasil Uji Coba Kebenaran Situs Penilaian Daring Timus Online Judge 142
Gambar 5.4 Grafik Waktu Hasil Pengumpulan Kode Berkas Variasi Algoritma Bidirectional Search Sebanyak 10 Kali 157
Gambar 5.5 Grafik Memori Hasil Pengumpulan Kode Berkas Variasi Algoritma Bidirectional Search Sebanyak 10 Kali 157
Gambar 5.6 Grafik Waktu Hasil Pengumpulan Kode Berkas Variasi Heuristic Sebanyak 10 Kali 159
Gambar 5.7 Grafik Memori Hasil Pengumpulan Kode Berkas Variasi Heuristic Sebanyak 10 Kali 160

BAB I
PENDAHULUAN

1.1 Latar Belakang

Game merupakan jenis hiburan yang disukai oleh semua orang dari usia anak-anak, dewasa, maupun lansia. Selain digunakan untuk menghilangkan kepenatan setelah beraktivitas, sebuah *game* juga dapat berfungsi untuk melatih pola pikir seseorang untuk mencari solusi dan memecahkan suatu permasalahan yang diberikan melalui *game* tersebut. Dahulu, *game* dimainkan secara tradisional seperti permainan kartu, catur, ular tangga, petak umpet, dan lainnya. Seiring dengan berkembangnya teknologi, permainan mulai dikembangkan ke arah yang lebih modern dengan teknologi yang ada. Lebih jauh lagi, teknologi sekarang juga bisa digunakan untuk membuat solusi dari sebuah *game* yang memiliki solusi yang sangat kompleks. Salah satu jenis dari *game* tersebut adalah Sokoban.

Topik Tugas Akhir ini mengacu pada penyelesaian problem pada Timus Online Judge dengan kode 1589 yang berjudul "Sokoban" [1]. Deskripsi dari permasalahan ini adalah diberikan sebuah *puzzle* yang merupakan map dua dimensi yang terdiri dari beberapa entitas. Entitas-entitas tersebut direpresentasikan oleh beberapa karakter. Pendekatan penulis untuk menyelesaikan permasalahan tersebut adalah dengan menggunakan algoritma *bidirectional search* dengan algoritma A* sebagai *forward move* dan algoritma *Breadth-First Seach (BFS)* sebagai *backward move* serta *goal pull distance* sebagai pendekatan *heuristic* untuk mendapatkan langkah optimal dalam penyelesaian *puzzle*. Hasil dari Tugas Akhir ini diharapkan dapat memberikan pemahaman yang lebih terhadap penggunaan graf ber-*heuristic* dalam menyelesaikan permasalahan di atas dan diharapkan dapat memberikan kontribusi pada pengembangan ilmu pengetahuan dengan model permasalahan yang serupa.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam Tugas Akhir ini adalah sebagai berikut.

1. Bagaimana cara menentukan algoritma yang tepat untuk menyelesaikan permasalahan?
2. Bagaimana cara menentukan pendekatan *heuristic* agar didapatkan hasil yang optimal?
3. Bagaimana cara melakukan *pruning* untuk memangkas jumlah *node* yang dihasilkan?

1.3 Batasan Permasalahan

Permasalahan yang dibahas pada Tugas Akhir ini memiliki beberapa batasan, yaitu sebagai berikut.

1. Implementasi algoritma menggunakan bahasa pemrograman C++.
2. Batas maksimal ukuran map dalam $n \times m$ adalah 8×8.
3. *Dataset* yang digunakan adalah *dataset* yang dikumpulkan dari sebuah *website* [2]. Pengumpulan dilakukan secara otomatis oleh sistem, sehingga *dataset* yang diambil hanya yang berada pada batasan permasalahan.
4. Batas maksimal waktu perangkat lunak berjalan adalah 600 detik (atau 10 menit) untuk *dataset Hard Case*.

Adapun batasan yang diberikan oleh *Timus Online Judge* adalah sebagai berikut.

1. Implementasi algoritma menggunakan bahasa pemrograman C++.
2. Batas maksimal ukuran map dalam $n \times m$ adalah 8×8.
3. Batas maksimal waktu perangkat lunak berjalan adalah 5 detik.
4. Batas memori perangkat lunak adalah 64 MB.
5. Ukuran *source code* maksimal yang dikirim adalah 64KB.

6. *Dataset* yang digunakan adalah *dataset* pada problem *Timus Online Judge*, Sokoban (Problem set nomor 1589).

1.4 Tujuan

Melakukan analisis dan desain algoritma untuk menyelesaikan permasalahan *Timus Online Judge* dengan kode 1589.

1.5 Manfaat

Membantu penggunaan algoritma yang tepat dan optimal dalam penyelesaian *puzzle* pada *game* Sokoban dari *Timus Online Judge* – 1589.

1.6 Metodologi

Pembuatan Tugas Akhir ini dilakukan dengan menggunakan metodologi sebagai berikut.

1.6.1 Penyusunan Proposal Tugas Akhir

Tahapan awal dari Tugas Akhir ini adalah penyusunan Proposal Tugas Akhir yang berisi pendahuluan, deskripsi dan gagasan metode-metode yang dibuat dalam Tugas Akhir ini. Pendahuluan ini terdiri dari latar belakang diajukannya Tugas Akhir, rumusan masalah dan batasan masalah yang ditetapkan, serta manfaat dari hasil pembuatan Tugas Akhir ini. Selain itu, dijabarkan pula tinjauan pustaka yang digunakan sebagai referensi pendukung pembuatan Tugas Akhir. Terdapat pula subbab jadwal kegiatan yang menjelaskan jadwal pengerjaan Tugas Akhir.

1.6.2 Studi Literatur

Pada tahap ini dilakukan pencarian literatur berupa jurnal yang digunakan sebagai referensi untuk pengerjaan Tugas Akhir ini. Literatur yang dipelajari pada pengerjaan Tugas Akhir ini berasal dari jurnal ilmiah yang diambil dari berbagai sumber di internet, beserta berbagai literatur online tambahan terkait *Game Sokoban, Searching Algorithm, Memory Optimization* dan *Online Judge*.

1.6.3 Implementasi Perangkat Lunak

Pada tahap ini akan dilaksanakan implementasi metode dan algoritma yang telah direncanakan. Implementasi sistem menggunakan *C++* sebagai bahasa pemrograman.

1.6.4 Pengujian dan Evaluasi

Tahap pengujian dan evaluasi dilakukan menggunakan *dataset problem set* nomor 1589, Sokoban, pada sistem penilaian milik *Timus Online Judge* untuk mengetahui hasil dan kinerja algoritma yang telah dibangun. Evaluasi didapatkan dari hasil *judge* yang telah dilakukan oleh *platform*.

1.6.5 Penyusunan Buku

Pada tahap ini dilakukan penyusunan buku yang menjelaskan seluruh konsep, teori dasar dari metode yang digunakan, implementasi, serta hasil yang telah dikerjakan sebagai dokumentasi dari pelaksanaan Tugas Akhir.

1.7 Sistematika Penulisan Laporan

Sistematika penulisan laporan Tugas Akhir adalah sebagai berikut.

Bab I Pendahuluan
Bab ini berisikan penjelasan mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi, dan sistematika penulisan dari pembuatan Tugas Akhir.

Bab II Dasar Teori
Bab ini berisi kajian teori dari metode dan algoritma yang digunakan dalam penyusunan Tugas Akhir ini.

Bab III Desain

Bab ini menjelaskan desain algoritma yang akan dibangun berdasarkan dasar teori yang dijelaskan pada bab 2.

Bab IV Implementasi

Bab ini membahas implementasi dari perancangan yang telah dibuat pada bab sebelumnya. Terdapat juga penjelasan berupa kode yang digunakan untuk proses implementasi.

Bab V Uji Coba dan Evaluasi

Bab ini membahas tahapan uji coba, kemudian hasil uji coba dievaluasi terhadap kinerja dari sistem yang dibangun.

Bab VI Kesimpulan dan Saran

Bab ini merupakan bab yang menjelaskan kesimpulan dari hasil uji coba yang dilakukan, masalah-masalah yang dialami pada proses dan tertulis saat pengerjaan Tugas Akhir, dan saran untuk pengembangan solusi ke depannya.

(Halaman ini sengaja dikosongkan)

BAB II
DASAR TEORI

Bab ini membahas mengenai teori-teori dasar yang digunakan dalam Tugas Akhir. Pada bagian awal, bab ini menjelaskan deskripsi permasalahan pada Timus Online Judge 1589 – Sokoban, dilanjutkan dengan menjelaskan beberapa terori yang akan digunakan pada Tugas Akhir ini.

2.1 Deskripsi Permasalahan

Permasalahan yang diberikan berupa sebuah *puzzle* Sokoban yang direpresentasikan oleh karakter-karakter 2D. Setiap karakter yang berbeda mewakili sebuah entitas tertentu. Gambar 2.1 adalah contoh standar input yang diberikan [1].

input

```
########
#@  $ .#
########
```

```
 ######
##   .#
#@  ###
#   * #
#   $ #
#     #
#######
```

Gambar 2.1 Contoh Standar Input Program

Spasi yang terdapat pada input menandakan sebuah indeks kosong, dan untuk karakter lainnya dijelaskan sebagai berikut.

- **#** adalah representasi entitas merupakan potongan dinding.
- **.** adalah sel kosong yang harus diisikan dengan sebuah kotak. Disebut juga dengan sel tujuan.
- **@** adalah representasi entitas posisi awal pemain, di mana pemain tidak berada pada sel tujuan.

- **+** adalah representasi entitas sel yang berisikan posisi awal pemain, di mana pemain berada pada sel tujuan.
- **$** adalah representasi entitas kotak, di mana kotak tidak berada pada sel tujuan.
- ***** adalah representasi entitas kotak, di mana kotak berada pada sel tujuan.

Gambar 2.2 Contoh Standar Output Program

Output dari program adalah sebuah *string* yang berisikan huruf **r**, **u**, **l**, dan **d**, yang mana menandakan empat kemungkinan arah gerak dari karakter. Ketika karakter bergerak dalam kondisi melakukan *action push*, huruf yang terdapat pada *string* menjadi huruf kapital (**R**, **U**, **L**, dan **D** sesuai urutan sebelumnya). Untuk setiap *testcase* pada *dataset*, dapat dipastikan selalu ada solusinya.

2.1.1 Parameter Input

Parameter input pada permasalahan Timus Online Judge 1589 – Sokoban adalah seperti di bawah ini.

1. N dan M, yang merupakan ukuran baris dan kolom pada *string* 2D, memiliki ukuran maksimal 8×8.
2. Input karakter berupa semua karakter yang telah dideskripsikan sebelumnya.

2.1.2 Batasan Permasalahan

Batasan pada permasalahan Timus Online Judge 1589 – Sokoban adalah:

1. Batas maksimal waktu perangkat lunak berjalan adalah 5 detik.
2. Batas memori perangkat lunak adalah 64 MB.
3. Ukuran *source code* maksimal yang dikirim adalah 64 KB.

2.1.3 Output Permasalahan

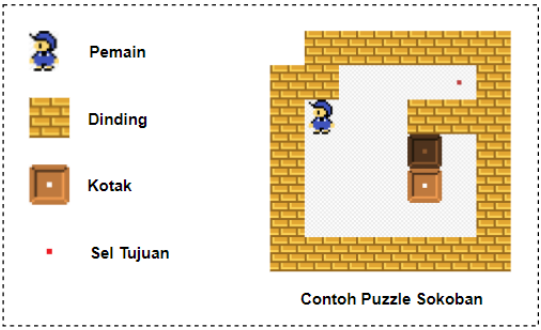
Output program adalah sebuah *string* yang merupakan hasil pencarian solusi dari standar input yang diberikan [1]. Panjang output tidak lebih dari 10000 karakter.

2.2 Sokoban

Sokoban merupakan sebuah *puzzle game* karya Hiroyuki Imabayashi yang diciptakan pada tahun 80an. Ide dari *game* ini adalah pemain diminta untuk menggeser sejumlah kotak ke tempat yang telah ditentukan (sel tujuan) untuk menyelesaikan permainan [3].

2.2.1 Entitas Sokoban

Dalam sokoban, terdapat beberapa entitas yang memiliki perlakuan berbeda-beda dalam permainan. Contoh *puzzle* sokoban dan entitas-entitasnya dapat dilihat pada Gambar 2.3.



Gambar 2.3 Contoh *Puzzle* Sokoban dan Entitas-Entitasnya

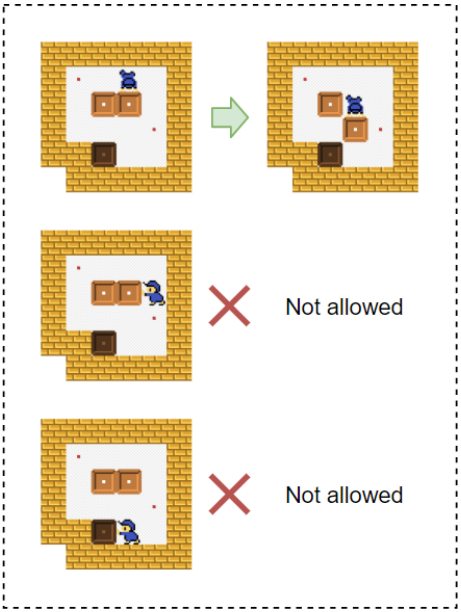
Berikut adalah penjelasan dari setiap entitas yang terdapat pada Gambar 2.3:

- Pemain: Entitas bergerak dalam permainan. Pergerakan dari pemain dapat mengubah posisi dari kotak dan menentukan arah penyelesaian permainan.
- Dinding: Entitas pembatas (*border*) dari setiap *puzzle*.
- Kotak: Entitas yang dapat dipindahkan oleh pemain.
- Sel Tujuan: Entitas yang juga merupakan sebuah sel kosong (sel yang dapat ditempati oleh pemain ataupun kotak). Permainan dinyatakan selesai apabila semua sel tujuan telah ditempati kotak.

2.2.2 Aturan Permainan Sokoban

Terdapat beberapa aturan dalam permainan sokoban, baik dalam cara penyelesaian maupun susunan entitas pada *puzzle* agar valid dan mungkin untuk diselesaikan. Aturan-aturan tersebut dituliskan sebagai berikut:

1. Setiap *puzzle* yang valid selalu dibatasi oleh dinding.
2. Jumlah dari kotak dan jumlah dari sel tujuan haruslah sama agar *puzzle* dapat diselesaikan.
3. Pemain hanya dapat digerakkan menuju empat titik lainnya pada bidang dua dimensi terhitung dari titiknya saat ini. Misalkan posisi pemain saat ini berada pada titik (i, j). Pemain hanya dapat digerakkan ke atas menuju titik $(i-1, j)$, ke kanan menuju titik $(i, j+1)$, ke bawah menuju titik $(i+1, j)$, dan ke kiri menuju titik $(i, j-1)$.
4. Pemain dapat melakukan *action push*, yaitu bergerak sambil memindahkan kotak yang berada pada titik tujuan pemain. *Action push* hanya dapat dilakukan apabila titik yang ditempati oleh kotak setelah dipindahkan adalah sebuah titik yang merupakan sel kosong (yang dapat ditempati). Visualisasi aturan *action push* dapat dilihat pada Gambar 2.4.



Gambar 2.4 Aturan *Action Push* pada Sokoban

2.3 Terminologi

Pada subbab ini, akan dibahas kumpulan istilah dan penjelasan (atau terminologi) yang akan sering muncul pada pembahasan-pembahasan berikutnya.

2.3.1 Entitas

Entitas-entitas pada sokoban pada subbab selanjutnya akan disebutkan menggunakan kata-kata yang tertera pada Tabel 2.1.

Tabel 2.1 Terminologi Entitas Sokoban

Nama Entitas	Bentuk Tunggal	Bentu Jamak
Pemain	*player*	-
Dinding	*wall*	*walls*
Kotak	*box*	*boxes*
Sel Tujuan	*goal*	*goals*

2.3.2 Indeks

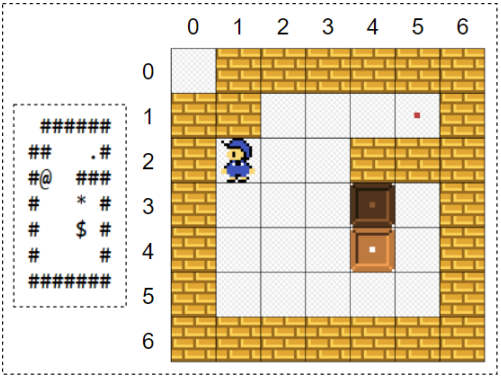
Indeks (yang selanjutnya disebut *index* atau *indices* dalam bentuk jamak) adalah sebuah integer yang mewakili titik sebuah entitas pada *puzzle* dua dimensi milik sokoban. Apabila jumlah baris dan jumlah kolom dari sebuah *puzzle* direpresentasikan oleh n_row dan n_col, rentang nilai *index* dari *puzzle* sokoban tersebut didefinisikan oleh Persamaan 2.1.

$$0 \leq index < (n\_row \times n\_col) \tag{2.1}$$

Sedangkan nilai *index* dari sebuah entitas e pada titik (i, j) didefinisikan oleh Persamaan 2.2.

$$index(e) = (n\_col \times i) + j \tag{2.2}$$

Sebagai contoh, silakan perhatikan visualisasi sebuah *state* yang berukuran 7×7 pada Gambar 2.5.



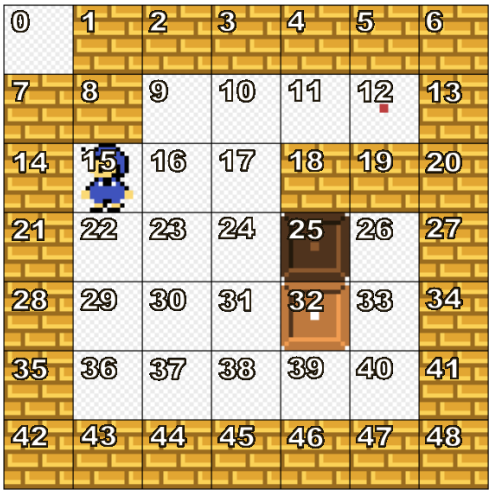
Gambar 2.5 Visualisasi *State*

Nilai *index* dari *player* yang terdapat pada *state* dalam Gambar 2.5 ditunjukkan oleh Persamaan 2.3.

$$index(player) = (7 \times 2) + 1 = 15 \qquad (2.3)$$

2.3.3 Representasi Bit

Representasi bit adalah posisi dari banyak *index* entitas sejenis dalam bentuk nilai integer 64bit. Dikatakan representasi bit karena semua *index* tersebut dilambangkan sebagai bit hidup (yang bernilai 1) dari bentuk *binary* sebuah integer 64bit. Sebagai contoh, silakan perhatikan *puzzle* beserta nilai-nilai *index*-nya yang terdapat pada Gambar 2.6.



Gambar 2.6 *Index* pada Sebuah *State*

Nilai representasi bit dari *boxes* pada *puzzle* tersebut adalah total hasil OR (∨) semua *shift left* sebanyak *index* dari masing-masing *box*. Detail perghitungan dapat dapat dilihat pada Tabel 2.2.

Tabel 2.2 Perhitungan Representasi Bit

Index / Shift Left	Binary	Nilai Integer
25	00000000000000000000 00000000000010000000 00000000000000000	2^{25} $= 33554432$
32	00000000000000000000 00001000000000000000 00000000000000000	2^{32} $= 4294967296$
Representasi Bit	00000000000000000000 00001000000010000000 00000000000000000	**4328521728**

2.3.4 *Node*

Node adalah *vertex* dari graf pencarian solusi yang terdapat pada sistem. *Node* sendiri terdiri dari beberapa informasi atau atribut yang mendukung sistem dalam pencarian solusi. Atribut-atribut yang terdapat pada *node* akan dijelaskan pada Tabel 2.3.

Tabel 2.3 Atribut *Node*

Nama Atribut	**Tipe Data**	**Keterangan**
heuristic	`unsigned long long`	Nilai yang menentukan urutan pengambilan *node* dalam pencarian solusi
push	`int`	Nilai yang menyimpan banyaknya jumlah *action push* yang telah dilakukan hingga mencapai *state* dari *node* saat ini
state	`pair<unsigned long long, int>`	Nilai *unique* pada sebuah *node*. *State* adalah pasangan dari representasi bit *boxes* dan *player index* pada *node* saat ini.

2.4 Strategi Penyelesaian Masalah

Ukuran dari *puzzle* sokoban yang diberikan maksimalnya adalah 8×8, dengan jaminan setiap *puzzle* pasti dikelilingi oleh *walls*. Maka dari itu, ukuran permainan yang dapat membentuk sebuah *state* adalah 6×6, atau sama dengan 36 indeks. Misalkan P adalah banyak indeks yang bukan merupakan sebuah *wall* yang mana merupakan area permainan, dan B adalah banyaknya *box* pada sebuah *puzzle*, maka banyaknya jumlah *state* yang mungkin (dilambangkan dengan Ω) didefinisikan oleh Persamaan 2.4.

$$|\Omega| = C_P^B * (P - B) \tag{2.4}$$

Dimana:

- C_P^B adalah banyaknya kombinasi posisi semua *box*
- $(P - B)$ adalah banyaknya posisi *player* yang mungkin untuk setiap kombinasi *boxes*.

Mudah untuk diketahui bahwa fungsi $|\Omega|$ memiliki nilai maksimum ketika P = 36 dan B = 18. Nilai maksimal dari $|\Omega|$ ditunjukkan oleh Persamaan 2.5.

$$|\Omega_{max}| = C_{36}^{18} * (36 - 18) \quad (2.5)$$

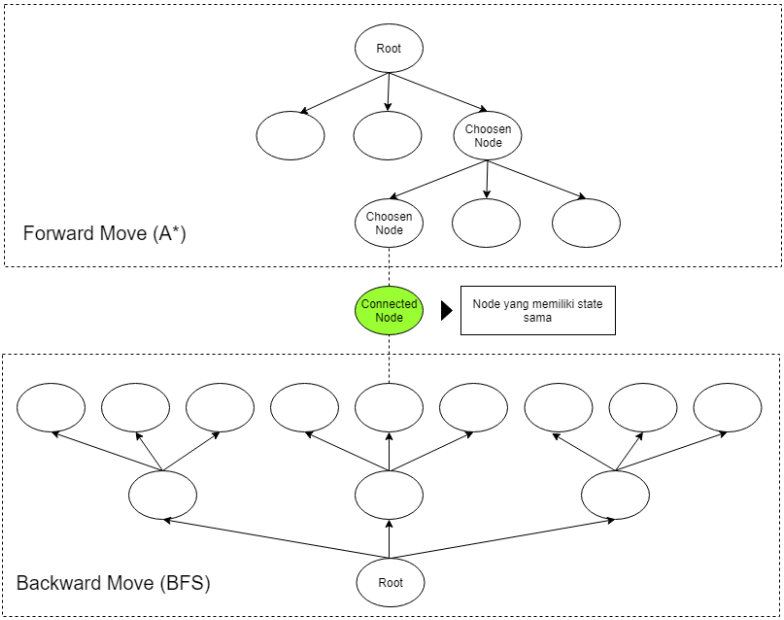
$$|\Omega_{max}| = 163{,}352{,}435{,}400$$

Worst case tersebut hanya akan didapat apabila pencarian solusi dilakukan dengan cara *naive*, seperti melakukan pencarian dengan algoritma *BFS* satu arah. Faktanya semakin dalam pencarian solusi pada sebuah *graph*, maka *node* yang dihasilkan akan jauh semakin banyak, dikarenakan jumlah *node* yang dihasilkan pada tiap tingkatannya bersifat eksponensial. Solusi yang diberikan oleh sistem adalah pencarian menggunakan *bidirectional search*, yang memiliki kedalaman pencarian lebih kurang setengah dari pencarian solusi satu arah. Pada subbab ini akan dijelaskan mengenai strategi penyelesaian masalah klasik 1589 – Sokoban pada daring *Timus Online Judge*. Algoritma-algoritma beserta optimasi-optimasi yang digunakan oleh sistem akan dijelaskan lebih lanjut.

2.4.1 *Bidirectional Search*

Bidirectional Search adalah salah satu algoritma yang digunakan dalam pencarian rute pada struktur data graf. Pencarian rute yang dilakukan oleh algoritma ini menggunakan dua buah graf. Satu graf bergerak dengan pencarian maju (*forward move*) dari *state* awal (*initial state*) menuju *state* akhir (*goal state*), dan satu graf lainnya melakukan pencarian mundur (*backward move*) dari *goal state* menuju *initial state* [4]. Ilustrasi dari salah satu kombinasi *bidirectional search*, algoritma A* sebagai *forward*

move dan algoritma *Breadth-First Search* (BFS) sebagai *backward move*, ditunjukkan oleh Gambar 2.7.



Gambar 2.7 *Bidirectional Search* menggunakan *A** dan *BFS*

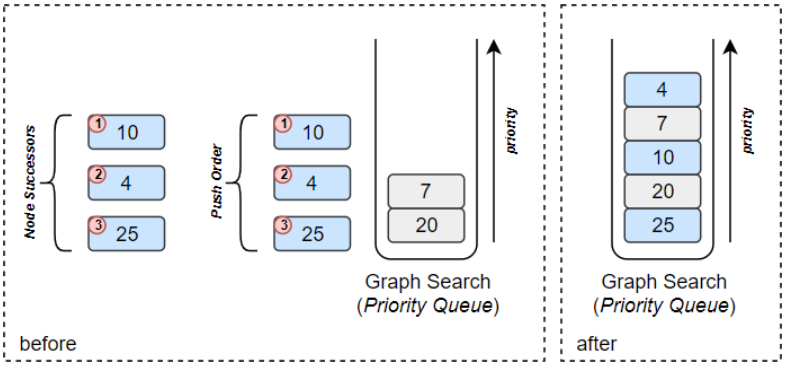
2.4.1.1 *Forward Move*

Forward move adalah pencarian rute dari *node* awal menuju *node* yang dicari. Pada sokoban, *forward move* dimulai dari *node* yang memuat *initial state* hingga *node* yang memuat *goal state* [4]. *Forward move* yang digunakan sistem untuk menyelesaikan permasalahan berdasarkan pada beberapa algoritma, yaitu:

a. A*

A* adalah salah satu algoritma untuk pencarian jalur (atau *path*) yang mana termasuk ke dalam jenis *informed graph* [5]. Algoritma A* menggunakan pendekatan heuristik yang menentukan priotitas pengambilan *node* dalam mencari rute untuk

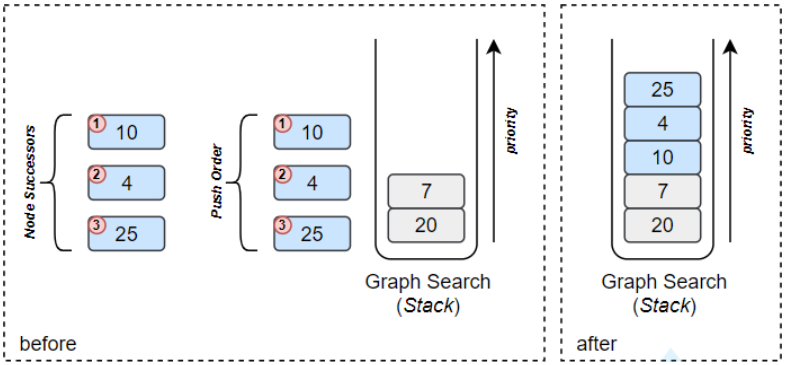
mencapai *goal state*. Ilustrasi dari algoritma A* dapat dilihat pada Gambar 2.8.



Gambar 2.8 Ilustrasi *A**

b. *Depth-First Search* (DFS)

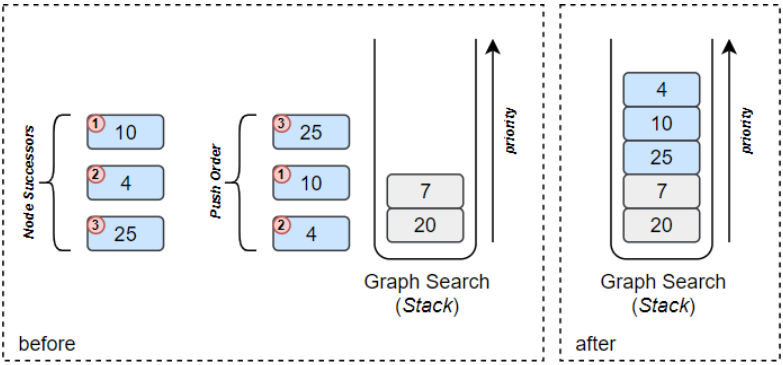
Depth-First Search merupakan graf yang termasuk dalam jenis *uninformed graph* [5]. Pencarian rute yang dilakukan oleh algoritma ini tidak tergantung pada nilai heuristik, sehingga untuk setiap *node* yang dihasilkan, akan langsung dimasukkan ke dalam *stack* pencarian solusi. Ilustrasi DFS dapat dilihat pada Gambar 2.9.



Gambar 2.9 Ilustrasi DFS

c. *Greedy* DFS

Greedy DFS menggunakan struktur data yang sama dengan DFS biasa dalam mencari solusi. Hanya saja, *greedy* DFS mengatur urutan *node* yang dimasukkan ke dalam *stack* pencarian tergantung dari nilai *heuristic*-nya. *Node* yang memiliki nilai *heuristic* terkecil akan dimasukkan di paling akhir, dengan tujuan agar dapat langsung digunakan untuk mencari solusi di level pencarian berikutnya. Ilustrasi *greedy* DFS dapat dilihat pada Gambar 2.10.



Gambar 2.10 Ilustrasi *Greedy* DFS

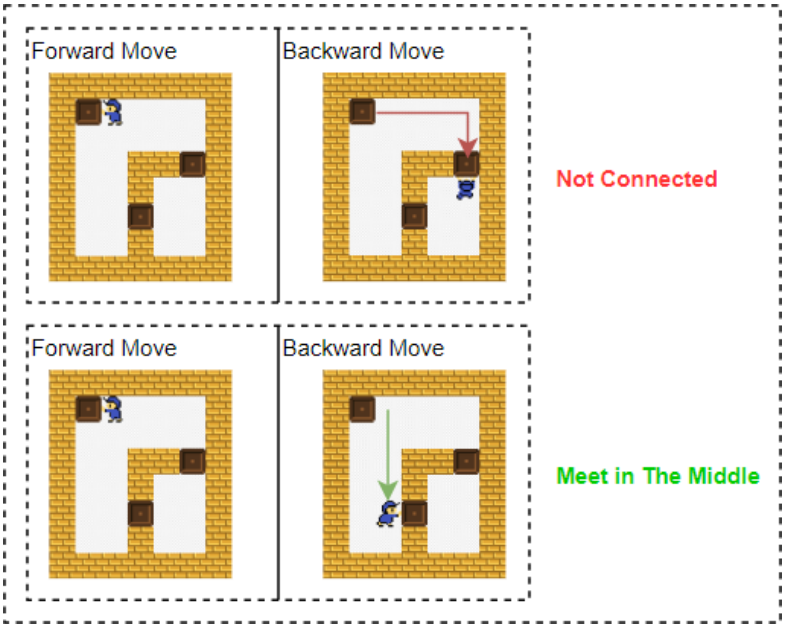
2.4.1.2 *Backward Move*

Backward move adalah pencarian rute dari *node* akhir yang dicari menuju *node* awal. Dalam sistem, *backward move* dimulai dari *node* yang memuat *goal state* hingga *node* yang memuat *initial state* [4]. *Backward move* yang digunakan untuk menyelesaikan permasalahan hanya menggunakan algoritma *Breadth-First Search* (BFS). Hal ini dimaksudkan agar peluang untuk terjadinya kondisi *meet in the middle* lebih besar.

2.4.1.3 *Meet in The Middle*

Meet in the middle adalah sebuah kondisi di mana terdapat pasangan *node* pada *forward move* dan *backward move* yang memiliki *state* yang bisa saling terhubung [4]. *Boxes* pada kedua

state akan dipastikan terlebih dahulu, apakah memiliki nilai yang sama atau tidak. Jika iya, maka posisi dari *player* kedua *state* akan diperiksa, bisa dihubungkan atau tidak. Penjelasan tentang kondisi yang memenuhi *meet in the middle* pada dua buah *state* yang berada pada *graph* pencarian berbeda dapat dilihat pada Gambar 2.11.



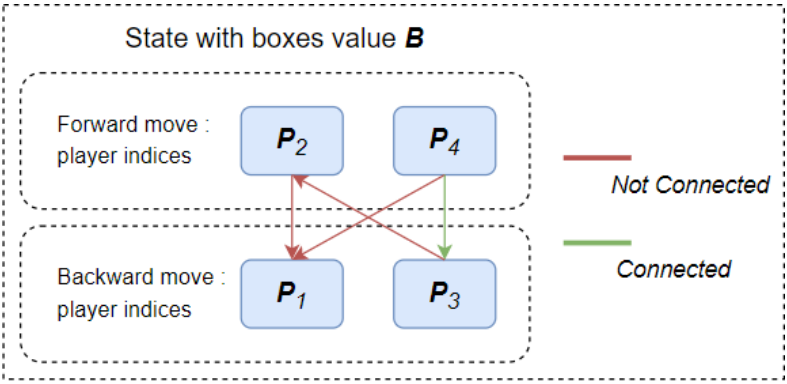
Gambar 2.11 Penjelasan Kondisi *Meet in The Middle*

2.4.1.4 Memorisasi *State*

Setiap *state* yang telah dikunjungi akan disimpan dalam sebuah map dua dimensi yang disebut dengan *map state*. Dimana *boxes* dan *player* dari sebuah *node* akan menjadi *key* dari *map state* tersebut. Penggunaan *map state* juga dimaksudkan untuk memudahkan pemeriksaan kondisi *meet in the middle*.

Bisa atau tidaknya dua buah *node* terhubung, akan dilihat dari *map state* untuk arah pencarian lainnya. Sebagai contoh, apabila *node* yang diperiksa saat ini berada pada *forward move*,

maka indeks *player* milik *node* akan dicoba untuk dihubungkan dengan semua indeks *player* pada *backward move* yang memiliki nilai *boxes* sama. Visualisasi dari pemeriksaan *meet in the middle* dapat dilihat pada Gambar 2.12.



Gambar 2.12 Visualisasi Pemeriksaan *Meet in The Middle*

Nilai *B* dan *P* pada Gambar 2.12 merepresentasikan nilai *boxes* dan *player* pada sebuah *node*. Untuk semua *node*, memiliki nilai *boxes* yang sama yaitu *B*. Sedangkan nilai *P* diikuti dengan angka menandakan urutan kemunculan *node* dalam mencapai nilai *boxes B*. Node 2 yang berasal dari *forward move*, diperiksa keterhubungannya dengan *node* 1. *Node* 3 yang berasal dari *backward move*, diperiksa keterhubungannya dengan *node* 2. Dan seterusnya hingga pada *node* 4, setelah dilakukan dua kali pemeriksaan (dikarenakan pada *backward move* sudah ada dua *node*), posisi *player* dari kedua *node* dapat terhubung dan kondisi *meet in the middle* telah terpenuhi.

2.4.1.5 Pendekatan Heuristik

Heuristic adalah seni dan ilmu pengetahuan yang berhubungan dengan suatu penemuan [6]. Dalam *Informed Graph*, *heuristic* berperan untuk menentukan bobot dari suatu *node*. Ada bermacam-macam *heuristic* yang bisa diaplikasikan dalam penyelesaian *game* sokoban. Berikut adalah macam-macam

pendekatan heuristik yang digunakan oleh sistem untuk menentukan nilai *heuristic* sebuah *node*:

a. *Manhattan Distance*

Bobot dihitung dengan nilai dari penjumlahan selisih absolut dari dua titik pada tiap bidangnya. Didefenisikan dengan Persamaan 2.6 sebagai berikut ini.

$$d\big((Ax, Ay), (Bx, By)\big) = |Ax - Bx| + |Ay - By| \quad (2.6)$$

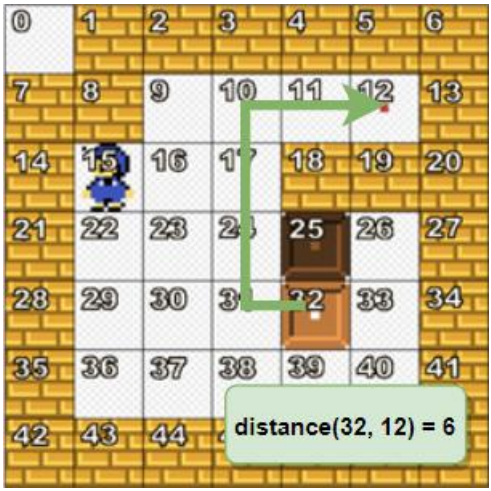
b. *Euclidean Distance*

Bobot dihitung dengan nilai dari akar penjumlahan selisih kuadrat dari dua titik pada tiap bidangnya. Didefenisikan dengan Persamaan 2.7 sebagai berikut ini.

$$d\big((Ax, Ay), (Bx, By)\big) = \sqrt{(Ax - Bx)^2 + (Ay - By)^2} \quad (2.7)$$

c. *Goal Pull Distance*

Bobot dihitung dari simulasi pergerakan kotak dari sel tujuan ke posisi kotak sekarang, seakan-akan kita melakukan *action pull*. Adapun cara mencari jaraknya bisa menggunakan algoritma *Breadth-First Search* (BFS) ataupun menggunakan tabel jarak (*distance table*) yang telah di-*precompute* menggunakan algoritma *Floyd*. Dikarenakan algoritma BFS membutuhkan banyak komputasi untuk setiap *node*-nya, akan jauh lebih efektif jika menggunakan algoritma *Floyd*. Visualisasi goal pull distance ditunjukkan oleh Gambar 2.13.



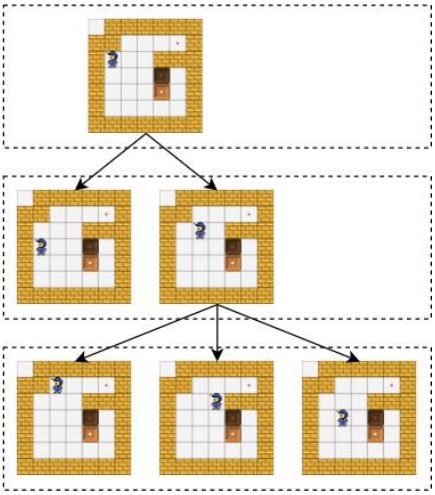
Gambar 2.13 Visualisasi *Goal Pull Distance*

2.4.2 Optimasi *Node*

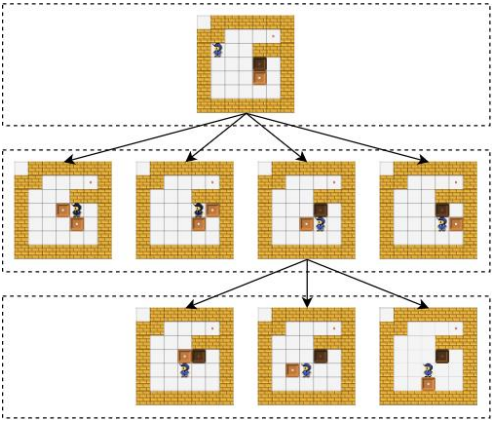
Node yang dihasilkan untuk setiap level pencarian, haruslah diminimalisir untuk mengurangi penggunaan memori pada sistem. Pada subbab ini, akan dibahas cara mengoptimasi *node*. Baik dalam pencarian, maupun optimasi atributnya.

2.4.2.1 Pembuatan *Node* dari *Action Push*

Untuk menyelesaikan *puzzle*, pencarian solusi yang dilakukan oleh sistem bukanlah dengan cara menyimulasikan pergerakan *player*. Melainkan dengan melakukan pencarian rute dari posisi *player* saat ini menuju sebuah indeks di samping *box*, lalu melakukan *action push* terhadap *box* tersebut. Dengan kata lain, sebuah *state* baru terbentuk dari sebuah *action push* terhadap satu *box* dari *state* saat ini. Sebagai perbandingan, contoh pembuatan *node* baru dengan cara naif dan pembuatan *node* dari *action push* tiap *box* pada sebuah *state* ditunjukkan oleh Gambar 2.14 dan Gambar 2.15.



Gambar 2.14 Pembuatan *Node* dengan Cara Naif

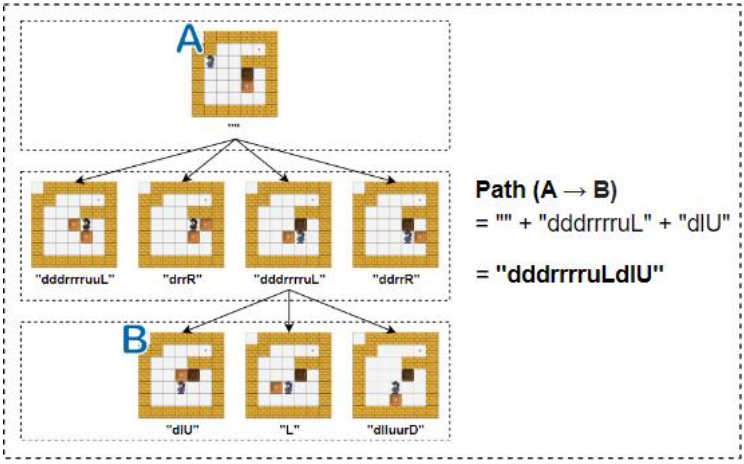


Gambar 2.15 Pembuatan *Node* dari *Action Push*

2.4.2.2 *Path Graph*

Path (atau rute) adalah sebuah *string* yang berisikan urutan pergerakan *player* dari *state* sebelumnya hingga *state* saat ini. *Path* juga dapat dikatakan sebagai sebuah jawaban dari *puzzle* pada sistem apabila *path* terbentuk merupakan hasil pengerakan *player* dari *initial state* hingga mencapai *goal state*. *Path* hanya terdiri dari karakter **l**, **r**, **u**, dan **d** yang mewakili arah kiri (*left*), kanan (*right*), atas (*up*), dan bawah (*down*), serta karakter dengan huruf yang sama namun berupa bentuk kapitalnya (yaitu **L**, **R**, **U**, dan **D**) yang menandakan gerakan yang dengan arah sama, hanya dengan sambil melakukan *action push*.

Pada sistem, *path* ini tersimpan dalam sebuah graf. Hal ini dilakukan agar memori yang digunakan lebih efisien dibandingkan sistem yang menyimpan *path*-nya sebagai atribut dari sebuah *node*. *State* yang akan disimpan diubah menjadi sebuah *unique key*, lalu *parent state* dari *state* saat ini (yang juga merupakan sebuah *key*) dan *path* yang terbentuk dari langkah-langkah dari *state* sebelumnya hingga *state* sekarang akan digabungkan menjadi sebuah *string* sebelum menjadi *value* dari *vertex* grafnya. Ilustrasi dari *path graph* ditunjukkan oleh Gambar 2.16.



Gambar 2.16 Ilustrasi *Path Graph*

2.4.3 *Pruning*

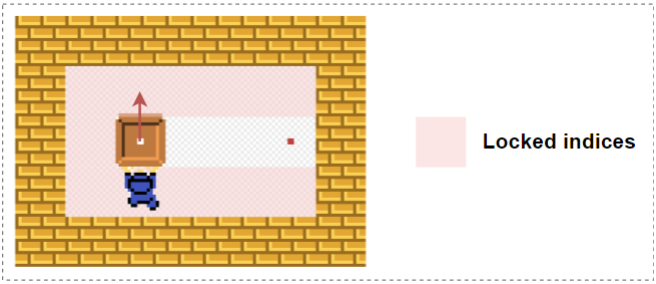
Pruning adalah proses untuk memangkas *state* yang dihasilkan oleh sistem [7]. Hal ini perlu dilakukan untuk mengurangi *space* dalam pencarian solusi. Salah satu graf pencarian yang digunakan, A*, membutuhkan pengurutan (*sorting*) dalam implementasinya. Pengurangan jumlah *state* selain mengurangi jumlah pemakaian memori, juga berarti meningkatkan efektifitas dalam pencarian solusi.

2.4.3.1 Deteksi *Deadlock*

Deadlock adalah suatu kondisi dimana *puzzle* tidak akan bisa diselesaikan lagi setelah mencapai *state* tertentu [8]. *State* yang berada pada kondisi ini tidak akan dimasukkan ke dalam graf pencarian. Untuk itu, perlu dilakukan *pruning* berupa deteksi *deadlock*. Berikut adalah macam-macam *deadlock* pada sokoban:

a. Simple Deadlock

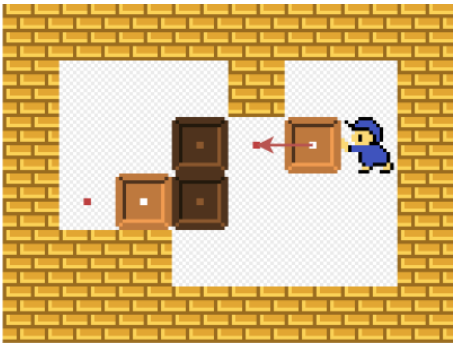
Simple deadlock adalah kondisi *deadlock* dimana satu atau lebih *box* menempati daerah "terlarang" (*locked indices*), yaitu daerah yang membuat *box* yang menempatinya tidak dapat mencapai *goal* manapun. Contoh bentuk dari *simple deadlock* ditunjukkan oleh Gambar 2.17.



Gambar 2.17 Contoh *Simple Deadlock*

b. Freeze Deadlock

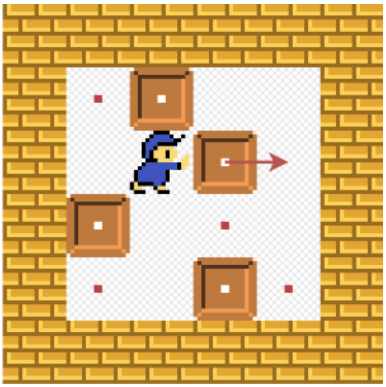
Freeze Deadlock adalah kondisi dimana ada dua atau lebih *box* yang tidak bisa digerakkan sama sekali, dan tidak semua *box* tersebut berada pada *goal*. Contoh bentuk dari *freeze deadlock* ditunjukkan oleh Gambar 2.18.



Gambar 2.18 Contoh *Freeze Deadlock*

c. Bipartite Deadlock

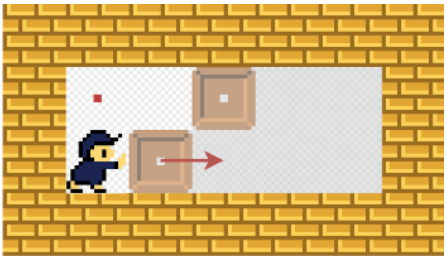
Ada kalanya suatu *goal* hanya bisa ditempati oleh satu *box* tertentu. Apabila *box* yang dimaksud sudah tidak bisa mencapai goal yang dimaksud, saat itulah *bipartite deadlock* terjadi. Contoh bentuk dari *bipartite deadlock* ditunjukkan oleh Gambar 2.19.



Gambar 2.19 Contoh *Bipartite Deadlock*

d. Corral Deadlock

Corral Deadlock adalah kondisi *deadlock* dimana ada area yang tidak dapat dijangkau oleh *player*. Contoh bentuk dari *corral deadlock* ditunjukkan oleh Gambar 2.20.



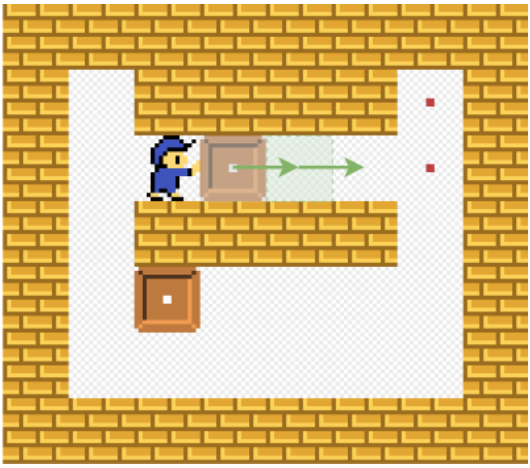
Gambar 2.20 Contoh *Corral Deadlock*

2.4.4 *Macro Moves*

Macro moves adalah sebuah langkah pasti yang dicapai hanya melalui satu arah perpindahan *box* [7]. Hanya dengan melakukan sebuah *action push* ataupun *pull*, kita dapat mencapai sebuah *state* yang seharusnya baru dicapai dua, tiga, atau bahkan lebih setelahnya.

2.4.4.1 *Macro Tunnel Moves*

Salah satu bentuk dari *macro moves* adalah *macro tunnel moves*, yaitu *macro moves* yang dilakukan karena adanya terowongan (*tunnel*) pada *puzzle*. *Index* yang dikategorikan sebagai *tunnel*, tidak akan ditempati oleh *box* apabila *action push* masih mungkin untuk dilakukan. Ilustrasi dari macro tunnel move ditunjukkan oleh Gambar 2.21.



Gambar 2.21 Ilustrasi *Macro Tunnel Moves*

(Halaman ini sengaja dikosongkan)

BAB III
PERANCANGAN SISTEM

Bab ini menjelaskan mengenai perancangan sistem untuk menyelesaikan permasalahan 1589 – Sokoban pada *Timus Online Judge* dengan menggunakan algoritma *bidirectional search.*

3.1 Deskripsi Umum Sistem

Sistem yang dirancang berupa sebuah aplikasi berbasis konsol, menggunakan input standar sebagai input sistem. Input berupa beberapa baris *string* yang berisikan karakter-karakter yang telah didefinisikan sebelumnya pada subbab 2.1. Input ini nantinya akan membentuk sebuah *puzzle* dua dimensi yang menjadi *state* awal dari *problem* yang akan diselesaikan. Solusi didapatkan dengan melakukan pencarian *state* akhir dengan menggunakan algoritma *bidirectional search.*

3.2 Penjelasan Sistem

Sistem yang dirancang terdiri dari beberapa metode, baik dari variasi algoritma pencarian solusi, maupun variasi *heuristic* dalam algoritma yang bersifat *informed graph*. Pada bab ini, penjelasan akan perancangan sistem lebih difokuskan pada solusi terbaik yang ditemukan, yaitu *bidirectional search* dengan algoritma *A** sebagai *forward move*, dan algoritma *Breadth-First Search* (*BFS*) sebagai *backward move*. Serta *heuristic* yang digunakan pada *forward move* adalah *goal pull distance*.

Batasan penggunaan memori yang sangat terbatas, disertai dengan *constraint* waktu yang sangat singkat membuat pemilihan bahasa pemrograman untuk pembuatan sistem ini jatuh pada bahasa pemrograman C++.

3.3 Desain Struktur *Node*

Node adalah kumpulan informasi yang terkandung dalam sebuah ruang permainan. Berisikan *state*, nilai *heuristic* dari *state*, dan banyaknya jumlah *action push* yang telah dilakukan untuk

mencapai *state* saat ini. Struktur *node* dari *graph* pencarian solusi ditunjukkan pada Kode Sumber 3.1.

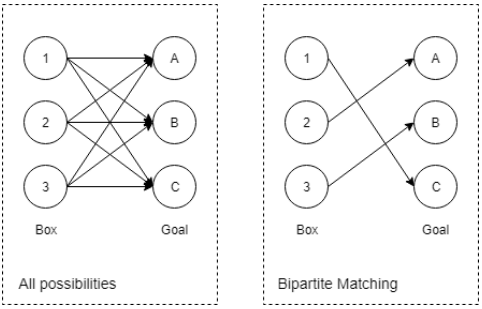
```
1.  // Node with nested pair (NODE) \
2.     (({heuristic}, {push}), ({boxes}, {player}))
3.  typedef pair<pair<int,int>, pair<ULL,int> > NODE;
4.
5.  #define _n_heuristic(x) (x).first.first
6.  #define _n_push(x) (x).first.second
7.  #define _n_state(x) (x).second
8.  #define _n_boxes(x) (x).second.first
9.  #define _n_player(x) (x).second.second
```

Kode Sumber 3.1 Struktur *Node Graph* Pencarian

Pada kode sumber di atas, juga terdapat *preprocessor function* yang memiliki *prefix* **_n** yang digunakan untuk mengakses atribut dari sebuah *node*.

3.3.1 Desain Fungsi Heuristic Value

Heuristic merupakan salah atribut yang menentukan urutan pengambilan *node* dalam pencarian *goal state*. *Heuristic* yang digunakan adalah *greedy bipartite macthing*. *Bipartite matching* sendiri adalah pemilihan *edge* pada dua buah set atribut (dalam hal ini pasangan *box* dan *goal*) dimana setiap *edge* tidak ada yang memiliki *endpoint* yang sama. Ilustrasi dari *bipartite matching* ditunjukkan oleh Gambar 3.1.



Gambar 3.1 Ilustrasi *Bipartite Matching*

Pemilihan pasangan ini dilakukan dengan pendekatan *greedy*. Dari semua pasangan *box* dan *goal* yang sudah di-*generate*, akan diambil dari *edge* yang memiliki bobot paling kecil, pencarian akan terus dilakukan hingga semua *box* dan *goal* memiliki pasangannya sendiri. Bobot dari sebuah *edge* adalah jarak dari indeks *box* dan indeks *goal* yang sudah dikalkulasi sebelumnya. *Pseudocode* fungsi yang menentukan nilai *heuristic* dari sebuah *node* dapat dilihat pada Kode Sumber 3.2.

```
1.  FUNCTION heurisicValue(&boxes, &player, forward_check):

2.      IF isDeadlock(boxes):
3.          RETURN INT_MAX
4.      ENDIF
5.
6.      box_list <- []
7.      convertToVector(boxes, box_list)
8.
9.      RETURN greedyBipartiteHeuristic(box_list, forward_c
    heck)
10. ENDFUNCTION
```

Kode Sumber 3.2 *Pseudocode* Fungsi Heuristic Value

Sebuah *node* yang memiliki kondisi *deadlock*, akan diberikan nilai -INF (dibaca *minus infinity*), yang menandakan bahwa *node* tersebut tidak valid. Sebuah *node* yang valid akan memiliki nilai *heuristic* yang didapat dari fungsi Greedy Bipartite Heuristic. *Pseudocode* dari fungsi ini ditunjukkan pada Kode Sumber 3.3.

```
1.  FUNCTION greedyBipartiteHeuristic(&box_list, forward_ch
    eck):
2.      bgedges <- []
3.
4.      FOR i = 0 to len(box_list):
5.          searchBoxGoals(box_list[i], i, bgedges, forward
    _check)
6.      ENDFOR
7.      sort(bgedges.begin(), bgedges.end(), bgedgeComp)
8.
9.      box_marked <- [SET DEFAULT false WITH LENGTH n_attr
    ]
10.     goal_marked <- [SET DEFAULT false WITH LENGTH n_att
    r]
11.
12.     counter <- 0
13.     score <- 0
14.     overlap_count <- 0
15.
16.     FOREACH bgedge IN bgedges:
17.         dist <- bgedge.weight
18.         box_idx <- bgedge.box_idx
19.         goal_idx <- bgedge.goal_idx
20.
21.         IF NOT box_marked[box_idx] AND NOT goal_marked[
    goal_idx]:
22.             box_marked[box_idx] <- true
23.             goal_marked[goal_idx] <- true
24.             score <- score + dist
25.             counter <- counter + 1
26.
27.             IF dist EQUAL TO 0:
28.                 overlap_count <- overlap_count + 1
29.             ENDIF
30.
31.             IF counter EQUAL TO n_attr:
32.                 BREAK
33.             ENDIF
```

Kode Sumber 3.3a *Pseudocode* Fungsi Greedy Bipartite Heuristic

```
34.         ENDIF
35.     ENDFOREACH
36.
37.     RETURN score - (OVERLAP_VALUE * overlap_count)
38. ENDFUNCTION
```

Kode Sumber 3.3b *Pseudocode* Fungsi Greedy Bipartite Heuristic

Skor/nilai dari *heuristic* akan dikurang dengan hasil dari `OVERLAP_VALUE * overlap_count`. Hal ini digunakan untuk memprioritaskan *state* yang memiliki jumlah *box* yang berada pada indeks *goal* lebih banyak untuk mencari solusi.

3.3.1.1 Desain Box Goal Edge

BGEDGE merupakan sebuah tipe data *custom* untuk sebuah *edge*. Berisikan bobot *edge*, indeks dari *box*, dan indeks dari *goal*. Semua atribut dari BGEDGE dapat diakses dari fungsi yang sudah didefinisikan. Berikut deklarasi BGEDGE pada Kode Sumber 3.4.

```
1.  // Box Goal Edge (BGEGDE) \
2.     ({weight}, ({box_idx}, {goal_idx}))
3.  typedef pair<int, pair<int,int> > BGEGDE;
4.
5.  #define _bge_weight(x) (x).first
6.  #define _bge_box_idx(x) (x).second.first
7.  #define _bge_goal_idx(x) (x).second.second
```

Kode Sumber 3.4 Deklarasi Tipe Data BGEDGE

3.3.1.2 Desain Fungsi Search Box Goals

Fungsi ini berguna untuk men-*generate* pasangan *box* dan *goal*. Berikut penjelasan untuk setiap parameternya:

- **box_index (INT)**: nomor indeks *box*.
- **box_idx (INT)**: nomor *id box* (posisi dalam array **box_list**).
- **box_goal_pair (VECTOR<BGEDGE>)**: variabel untuk menyimpan semua pasangan *box* dan *goal*.

- **forward_check (BOOL)**: sebuah value yang menentukan apakah *goals* yang digunakan milik *forward move* atau *backward move*.

Pseudocode untuk fungsi ini ditunjukkan oleh Kode Sumber 3.5.

```
1.  FUNCTION searchBoxGoals(&box_index, &box_idx, &bgedges,
     forward_check):
2.      goal_list <- []
3.
4.      IF forward_check:
5.          goal_list <- forward_goal_list
6.      ELSE:
7.          goal_list <- backward_goal_list
8.      ENDIF
9.
10.     FOR i = 0 to len(goal_list):
11.         goal_index <- goal_list[i]
12.         bgedges.push_back(make_pair(table[box_index][go
    al_index], make_pair(box_idx, i)))
13.     ENDFOR
14. ENDFUNCTION
```

Kode Sumber 3.5 *Pseudocode* Fungsi Search Box Goals

3.3.1.3 Desain Fungsi Box Goal Edge Compare

Fungsi ini digunakan sebagai fungsi pembanding/*compare function* untuk mengurutkan *edges* (pasangan-pasangan *box* dan *goal*) yang telah di-*generate*. Semua *egde* tersebut diurutkan berdasarkan bobotnya, yang bernilai terkecil akan diletakkan pada posisi paling awal. *Pseudocode* untuk fungsi ini ditunjukkan oleh Kode Sumber 3.6.

```
1.  FUNCTION bgedgeComp (&bgea, &bgeb):
2.      IF bgea.weight EQUAL TO bgeb.weight:
3.          IF bgea.box_idx EQUAL TO bgeb.box_idx:
4.              RETURN bgea.goal_idx LESS THAN bgeb.goal_id
    x
5.          ENDIF
6.
7.          RETURN bgea.box_idx LESS THAN bgeb.box_idx
8.      ENDIF
9.
10.     RETURN bgea.weight LESS THAN bgeb.weight
11. ENDFUNCTION
```

Kode Sumber 3.6 *Pseudocode* Fungsi Box Goal Edge Compare

3.3.2 Desain Atribut *Push*

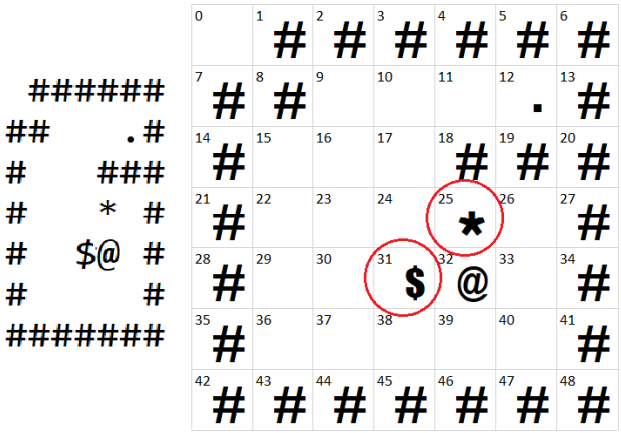
Atribut *push* berguna untuk menyimpan banyaknya jumlah *action push* yang telah dilakukan hingga mencapai *state* dari *node* sekarang. Sistem yang dibuat kali ini membutuhkan banyaknya *action push* sebagai parameter pembanding dalam menentukan *node* yang akan diperiksa selanjutnya dalam *graph* pencarian.

3.3.3 Desain Atribut *State*

State dalam sebuah *node* didefinisikan sebagai *boxes* (yang merupakan posisi semua *box*) dan *player* (yang merupakan posisi *player*) pada sebuah *state*.

3.3.3.1 Desain Atribut *Boxes*

Boxes adalah sebuah atribut yang berisikan posisi-posisi dari *box* yang direpresentasikan oleh sebuah nilai integer 64bit. Setiap bit yang hidup (atau bit yang bernilai 1) mewakili posisi dari sebuah *box*. Visualisasi indeks pada sebuah *state* yang merujuk posisi *box* dapat dilihat pada Gambar 3.2.



Gambar 3.2 Visualisasi Indeks pada Sebuah *State* yang Merujuk Posisi *Box*

Berikut adalah beberapa poin penting terkait ilustrasi di atas:

- Puzzle berukuran 7*7, sehingga penomoran indeks dimulai dari 0 hingga 48 (inklusif).
- Karakter '*' dan '$' yang merupakan karakter *box* berada pada indeks ke 25 dan 31.

Berdasarkan detail di atas, nilai *boxes* dalam representasi integer 64bit nya adalah:

$(0000000000000000000000000000000100000100000000000000000000000000)_2 = (2181038080)_{10}$

3.3.3.2 Desain Atribut *Player*

Player adalah sebuah atribut yang berisikan indeks dari *player* pada sebuah *state*. Atribut ini memiliki tipe data integer.

3.4 Desain Mark State

State adalah atribut yang menjadi nilai *unique* dari sebuah *node*, sehingga *state* juga perlu dimemorisasi agar *graph* pencarian untuk penyelesaian *puzzle* tidak *cyclic*/berulang dan dapat diselesaikan. Pada sistem ini, *state* disimpan dalam bentuk struktur

data *map* dua dimensi, dimana atribut *boxes* dan *player* pada sebuah *state* menjadi *key*-nya. *State* untuk *forward move* dan *backward move* juga harus dipisah, dikarenakan berada dalam *graph* pencarian yang berbeda. Variabel-variabel yang menyimpan *state* dapat dilihat pada Kode Sumber 3.7.

```
1.  map<ULL, map<int,bool> > forward_state_map;
2.  map<ULL, map<int,bool> > backward_state_map;
```

Kode Sumber 3.7 Variabel-Variabel yang Menyimpan *State*

Diberikan dua buah *key* untuk menandai *state*, dengan tujuan untuk memudahkan proses pemeriksaan *meet in the middle*. *Pseudocode* dari fungsi dapat dilihat pada Kode Sumber 3.8.

```
1.  FUNCTION markState(&node, forward_check):
2.      IF forward_check:
3.          forward_state_map[_n_boxes(node)][_n_player(nod
    e)] <- true
4.      ELSE:
5.          backward_state_map[_n_boxes(node)][_n_player(no
    de)] <- true
6.      ENDIF
7.  ENDFUNCTION
```

Kode Sumber 3.8 *Pseudocode* Fungsi Mark State

3.5 Desain *Path Graph*

Path adalah sebuah *string* yang berisikan urutan pergerakan *player* dari *initial state* hingga *state* saat ini. *Path* juga dapat dikatakan sebagai sebuah jawaban dari *puzzle* pada sistem setelah *path* pada *forward graph search* dan *backward graph search* telah digabungkan. *Path* hanya terdiri dari karakter **l**, **r**, **u**, dan **d** yang mewakili arah kiri (*left*), kanan (*right*), atas (*up*), dan bawah (*down*), serta karakter dengan huruf yang sama namun berupa bentuk kapitalnya (yaitu **L**, **R**, **U**, dan **D**) yang menandakan gerakan yang dengan arah sama, hanya dengan sambil melakukan

action push. Variabel-variabel yang merupakan *path graph* dapat dilihat pada Kode Sumber 3.9.

```
1.  map<ULL,string> forward_path_map;
2.  map<ULL,string> backward_path_map;
```

Kode Sumber 3.9 Variabel-Variabel *Path Graph*

Pada sistem, *path* ini tersimpan dalam sebuah *graph*. *State* yang akan disimpan akan diubah menjadi sebuah *unique key*, lalu dari *parent state* dari *state* saat ini (yang juga merupakan sebuah *key*) dan *path* yang terbentuk dari langkah-langkah dari *state* sebelumnya hingga *state* sekarang akan digabungkan menjadi sebuah *string* sebelum menjadi *value* dari *map*-nya.

3.5.1 Desain Fungsi Get Path Map Key

Fungsi ini mengembalikan sebuah *value* yang nantinya akan menjadi *key* pada *path graph*. *Pseudocode* dari fungsi ini dapat dilihat pada Kode Sumber 3.10.

```
1.  FUNCTION getPathMapKey(&boxes, &player):
2.      RETURN to_string(boxes) + "-" + to_string(player)
3.  ENDFUNCTION
```

Kode Sumber 3.10 *Pseudocode* Fungsi Get Path Map Key

3.5.2 Desain Fungsi Get Path Map Value

Fungsi ini mengembalikan sebuah *value* yang akan menjadi *value* pada *path graph*. Terdiri dari *parent key* dan *path* dari *parent state* ke *state* sekarang yang dipisahkan oleh karakter ‘;’. *Pseudocode* dari fungsi ini dapat dilihat pada Kode Sumber 3.11.

```
1.  FUNCTION getPathMapValue(&parent_key, &path):
2.      RETURN parent_key + ";" + path;
3.  ENDFUNCTION
```

Kode Sumber 3.11 *Pseudocode* Fungsi Get Path Map Value

3.5.3 Desain Fungsi Mark Path

Fungsi ini berguna untuk memorisasi *path* dari setiap *state*. *Pseudocode* dari fungsi ini dapat dilihat pada Kode Sumber 3.12.

```
1.  FUNCTION markPath(&key, &parent_key, &path, forward_che
    ck <- true):
2.      IF forward_check:
3.          forward_path_map[key] <- getPathMapValue(parent
    _key, path)
4.      ELSE:
5.          backward_path_map[key] <- getPathMapValue(paren
    t_key, path)
6.      ENDIF
7.  ENDFUNCTION
```

Kode Sumber 3.12 *Pseudocode* Fungsi Mark Path

3.5.4 Desain Fungsi Split Path Map Value

Fungsi ini berguna untuk memisahkan nilai dari *graph* milik *path* dan menaruhnya pada variabel-variabel baru yang menjadi parameter. *Pseudocode* dari fungsi ini dapat dilihat pada Kode Sumber 3.13.

```
1.  FUNCTION splitPathMapValue(&path_map_value, &parent, &p
    ath):
2.      parent <- ""
3.      path <- ""
4.      parent_build <- true
5.
6.      FOR i = 0 to len(path_map_value):
7.          c <- path_map_value[i]
8.
9.          IF c EQUAL TO ';':
10.             parent_build <- false
11.         ELSE:
12.             IF parent_build:
13.                 parent <- parent + c
14.             ELSE:
15.                 path <- path + c
16.             ENDIF
17.         ENDIF
18.     ENDFOR
19. ENDFUNCTION
```

Kode Sumber 3.13 *Pseudocode* Fungsi Split Path Map Value

3.5.5 Desain Fungsi Get Full Path

Fungsi ini berguna untuk mendapatkan *path* yang utuh/sebenarnya dari sebuah *state*. *Pseudocode* dari fungsi ini dapat dilihat pada Kode Sumber 3.14.

```
1.  FUNCTION getFullPath(&start_key, forward_check <- true)
    :
2.      root_key <- ""
3.
4.      IF forward_check:
5.          root_key <- forward_path_map_root
6.      ELSE:
7.          root_key <- backward_path_map_root
8.      ENDIF
9.
10.     full_path <- ""
11.     key <- start_key;
12.
13.     WHILE key NOT EQUAL TO root_key:
14.         path_map_value <- ""
15.
16.         IF forward_check:
17.             path_map_value <- forward_path_map[key]
18.         ELSE:
19.             path_map_value <- backward_path_map[key]
20.         ENDIF
21.
22.         splitPathMapValue(path_map_value, parent, path)

23.
24.         IF forward_check:
25.             full_path <- path + full_path
26.         ELSE:
27.             full_path <- full_path + path
28.         ENDIF
29.
30.         key <- parent
31.     ENDWHILE
32.
33.     RETURN full_path
34. ENDFUNCTION
```

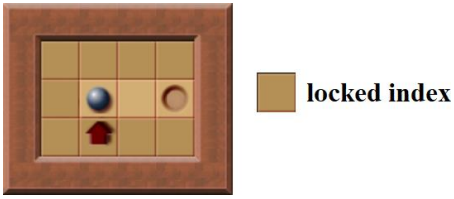
Kode Sumber 3.14 *Pseudocode* Fungsi Get Full Path

3.6 Pemangkasan State (*Pruning*)

Pruning adalah proses untuk memangkas *state* yang di-*generate* oleh sistem. Hal ini perlu dilakukan untuk mengurangi *space* dalam pencarian solusi. Salah satu *graph* pencarian yang digunakan, A*, membutuhkan *sorting* dalam implementasinya. Pengurangan jumlah *state* selain mengurangi jumlah pemakaian *memory*, juga berarti meningkatkan efektifitas dalam pencarian solusi. Bentuk *pruning state* yang dilakukan oleh sistem berupa deteksi *deadlock*. *Deadlock* adalah suatu kondisi dimana *puzzle* tidak akan bisa diselesaikan lagi setelah mencapai *state* tertentu. *State* yang berada pada kondisi ini tidak akan dimasukkan ke dalam *graph* pencarian.

3.6.1 Desain Deteksi *Simple Deadlock*

Simple deadlock adalah kondisi *deadlock* dimana satu atau lebih *box* menempati daerah terkunci/*locked index* yaitu daerah yang membuat *box* yang menempatinya tidak dapat mencapai *goal* manapun. Gambaran dari *simple deadlock* ditunjukkan oleh Gambar 3.3.



Gambar 3.3 *Simple Deadlock*

Dari gambar di atas, *simple deadlock* akan terjadi apabila *box* didorong selain ke arah kanan. Semua *locked index* dapat ditentukan dengan cara melakukan *action pull* dari semua posisi *goal*. Dengan kata lain, apabila *box* bisa ditarik menuju ke suatu indeks, maka dari indeks tersebut *box* juga dapat didorong ke indeks saat ini. Pencarian *locked index* ini dilakukan menggunakan metode *flood fill*. *Pseudocode* untuk fungsi yang mendeteksi *simple deadlock* ditunjukkan oleh Kode Sumber 3.15.

```
1.  FUNCTION isSimpleDeadlock(&boxes):
2.      RETURN (locks ∧ boxes) NOT EQUAL TO 0
3.  ENDFUNCTION
```

Kode Sumber 3.15 *Pseudocode* Fungsi Is Simple Deadlock

3.6.1.1 Desain Fungsi Generate Deadlock

Fungsi ini digunakan untuk menginisialisasi nilai dari variable *locks* yang merepresentasikan semua posisi dari *locked index*. Nilai dari *locks* adalah *inverse bits* dari *safe indices* yang merepresentasikan semua indeks dari *safe index*. *Pseudocode* fungsi ini ditunjukkan oleh Kode Sumber 3.16.

```
1.  FUNCTION generateDeadlock():
2.      forward_goal_list <- []
3.      convertToVector(forward_goals, forward_goal_list)
4.
5.      visited <- [SET DEFAULT false WITH LENGTH ULL_BIT_S
    IZE]
6.      safe_indices <- 0
7.
8.      FOREACH goal_index IN forward_goal_list:
9.          expandSafeIndices(safe_indices, goal_index, vis
    ited)
10.     ENDFOREACH
11.
12.     locks <- ¬safe_indices
13. ENDFUNCTION
```

Kode Sumber 3.16 *Pseudocode* Fungsi Generate Deadlock

3.6.1.2 Desain Fungsi Expand Safe Indices

Fungsi ini digunakan untuk mengindentifikasi semua *safe index*, yaitu indeks yang dapat ditempati oleh *box* tanpa mengakibatkan terjadinya *simple deadlock*. Identifikasi dilakukan dengan cara melakukan simulasi *action pull* terhadap semua *box* yang dimisalkan awalnya berada pada *goal index*. Semua indeks yang dapat ditempati oleh player selama melakukan simulasi

tersebutlah yang dikatakan sebagai *safe indices*. Ilustrasi pencarian *safe indices* ditunjukkan pada Gambar 3.4.

Gambar 3.4 Pencarian *Safe Indices*

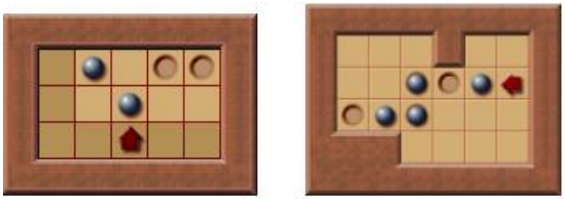
Ilustrasi yang ditunjukkan oleh Gambar 3.4 adalah contoh *action pull* ke arah kiri dari sebuah *state*, apabila arah yang ingin dituju oleh *player* sudah merupakan sebuah *wall*, maka *action pull* ke arah tersebut sudah tidak dapat dilakukan. Akibatnya, indeks yang ditempati oleh *player* saat ini sudah tidak bisa dijadikan *safe index*. Namun, tidak menutup kemungkinan indeks tersebut bisa menjadi *safe index* apabila untuk *state* yang tercipta dari *action pull box* yang lainnya bisa mencapainya. Simulasi *action pull* ini dilakukan pada fungsi Expand Safe Indices. *Pseudocode* dari fungsi ini dapat dilihat pada Kode Sumber 3.17.

```
1.  FUNCTION expandSafeIndices(&safe_indices, index, visite
    d):
2.      IF NOT isValidIndex(index) OR visited[index]:
3.          RETURN
4.      ENDIF
5.
6.      visited[index] <- true
7.      safe_indices <- safe_indices ∨ mask(index)
8.      directions = [U_DIR, R_DIR, D_DIR, L_DIR]
9.
10.     FOREACH DIR IN directions:
11.         f1_side <- getIndexMoved(index, DIR)
12.         f2_side <- getIndexMoved(f1_side, DIR)
13.
14.         IF isValidIndex(f1_side) AND NOT isWall(f1_side
    ):
15.             IF isValidIndex(f2_side) AND NOT isWall(f2_
    side):
16.                 expandSafeIndices(safe_indices, f1_side
    , visited)
17.             ENDIF
18.         ENDIF
19.     ENDFOREACH
20. ENDFUNCTION
```

Kode Sumber 3.17 *Pseudocode* Fungsi Expand Safe Indices

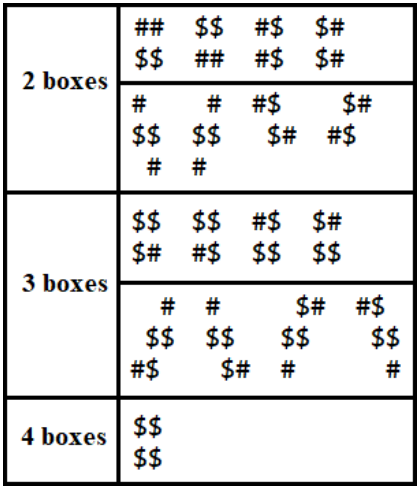
3.6.2 Desain Deteksi *Freeze Deadlock*

Freeze deadlock adalah suatu kondisi dimana sebuah *state* memiliki dua atau lebih *box* yang tidak dapat dipindahkan lagi dan tidak semua *box* tersebut sudah berada pada *goal index*. Gambaran dari *freeze deadlock* ditunjukkan oleh Gambar 3.5.



Gambar 3.5 *Freeze Deadlock*

Pada sistem ini, pendeteksi untuk *freeze deadlock* terbatas hanya hingga empat *box*. Bentuk *freeze deadlock* yang dapat dideteksi oleh sistem ditunjukkan oleh Gambar 3.6.



Gambar 3.6 *Freeze Deadlock* yang Dideteksi Oleh Sistem

Pseudocode dari fungsi yang mendeteksi *freeze deadlock* dapat dilihat pada Kode Sumber 3.18.

```
1.  FUNCTION isFreezeDeadlock(&boxes):
2.      box_list <- []
3.      convertToVector(boxes, box_list)
4.
5.      deadlock <- false
6.
7.      FOREACH box IN box_list:
8.          IF deadlock:
9.              BREAK
10.         ENDIF
11.
12.         assignWindsDirection(box)
13.
14.         deadlock <- false
15.                     OR isFDTwoBoxes1(boxes, box)
16.                     OR isFDTwoBoxes2(boxes, box)
17.                     OR isFDThreeBoxes1(boxes, box)
18.                     OR isFDThreeBoxes2(boxes, box)
19.                     OR isFDFourBoxes(boxes, box)
20.     ENDFOREACH
21.
22.     RETURN deadlock
23. ENDFUNCTION
```

Kode Sumber 3.18 *Pseudocode* Fungsi Is Freeze Deadlock

3.7 Desain Fungsi Validasi

Fungsi pengecek adalah fungsi yang digunakan untuk memvalidasi sebuah kondisi. Umumnya hanya digunakan dalam *statement if-else* pada program ini, agar logika yang diberikan lebih *readable*. Detail dari tiap fungsi ini akan dijelaskan pada Tabel 3.1.

Tabel 3.1a Tabel Fungsi Validasi

No	Nama Fungsi	Kegunaan
1	`isBox`	Memeriksa apakah suatu indeks merupakan sebuah *box* pada *boxes*
2	`isDeadlock`	Memeriksa apakah suatu *state* berada dalam kondisi *deadlock*
3	`isGoal`	Memeriksa apakah suatu indeks merupakan sebuah *goal* pada *goals*
4	`isGoals`	Memeriksa suatu representasi bit apakah setiap bit yang hidupnya merupakan sebuah *goal*
5	`isGame Over`	Memeriksa apakah suatu *node* telah mencapai kondisi akhir dari *game*
6	`isGameOverWithoutMove`	Memeriksa apakah *state* paling awal sudah merupakan akhir *game*
7	`isLock`	Memeriksa apakah suatu indeks merupakan sebuah *locked indices*
8	`isObstacle`	Memeriksa apakah suatu indeks merupakan sebuah *obstacle*. Hambatan/*obstacle* adalah sebutan untuk sebuah indeks yang tidak mungkin lagi ditempati oleh sebuah *box*. Hal ini dikarenakan indeks tersebut merupakan sebuah *wall* ataupun sebuah indeks yang telah ditempati *box* lainnya.

Tabel 3.1b Tabel Fungsi Validasi

No	Nama Fungsi	Kegunaan
9	`isOnSameColumn`	Memeriksa apakah dua buah indeks berada pada kolom yang sama
10	`isOnSameRow`	Memeriksa apakah dua buah indeks berada pada baris yang sama
11	`isStateSeen`	Memeriksa apakah suatu *state* sudah pernah dikunjungi.
12	`isTunnel`	Memeriksa apakah sebuah indeks merupakan bagian dari representasi bit *tunnels*
13	`isValidIndex`	Memeriksa apakah sebuah indeks masih berada dalam batas valid
14	`isValidIndexMove`	Memeriksa apakah sebuah indeks yang merupakan hasil dari perpindahan dari indeks sebelumnya merupakan hasil yang valid
15	`isWall`	Memeriksa apakah sebuah indeks merupakan bagian dari representasi bit *walls*

3.8 Desain Fungsi Global

Fungsi ini dapat digunakan secara global dan mendukung penyelesaian *puzzle* nantinya. Yang dimaksud dengan global adalah, fungsi ini digunakan baik oleh *graph* pencarian milik

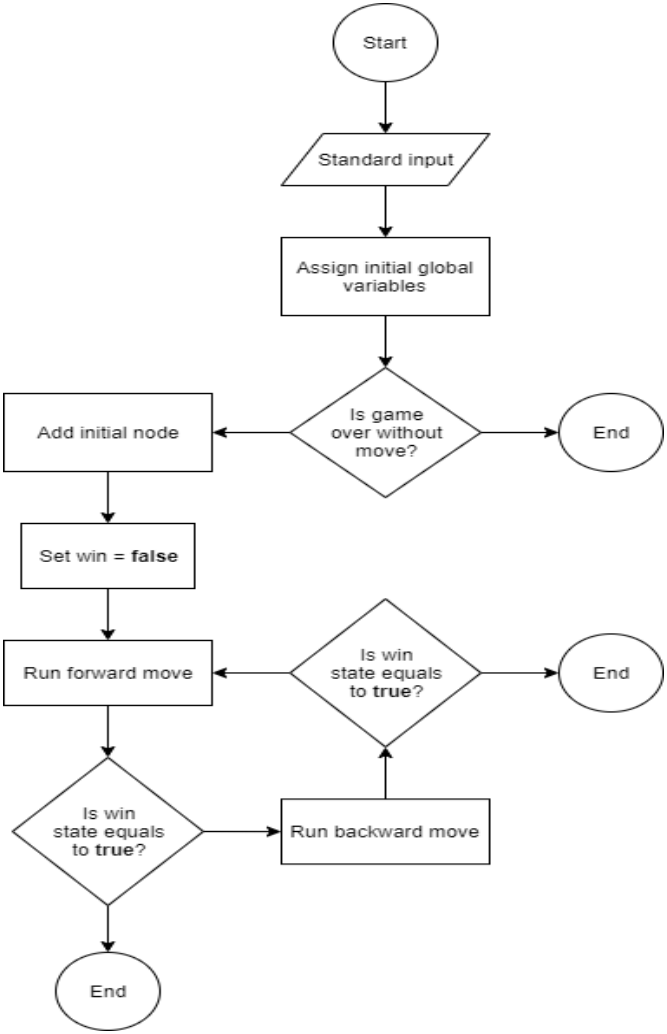
forward move, maupun *backward move*. Detail dari tiap fungsi ini akan dijelaskan pada Tabel 3.2.

Tabel 3.2 Tabel Fungsi Global

No	Nama Fungsi	Kegunaan
1	`clearTemplate`	Menginisialisasi semua *class* dengan nilai *default*-nya
2	`memsetTemplate`	Menginisialisasi nilai suatu *array* dengan sebuah *default value*
3	`assignToVector`	Mengubah nilai *array* pada *range* tertentu menjadi sebuah *vector*
4	`getIndexMoved`	Mendapatkan nilai indeks yang sudah dipindahkan sesuai dengan arah yang dituju
5	`getNewNode`	Mengembalikan sebuah *node* utuh
6	`getShortestPath`	Mencari *shortest path* dari satu indeks ke indeks lain tanpa menggeser *box* manapun
7	`getBoxesMoved`	Mengembalikan nilai represenatsi bit baru setelah satu *box*-nya dipindahkan
8	`assignWindsDirection`	Mengubah nilai dari variabel global yang menunjukkan arah mata angin dari indeks pusatnya
9	`convertToVector`	Mengubah semua bit yang hidup pada sebuah representasi bit menjadi *vector* indeks
10	`generateSuccessors`	Men-*generate node successors*, baik untuk *forward move* maupun *backward move*

3.9 Desain Solusi Utama

Dalam subbab ini, akan diberikan sebuah *flowchart* yang menggambarkan jalannya sistem secara umum pada Gambar 3.7.



Gambar 3.7 *Flowchart* Sistem

3.10 Desain Fungsi Run Input

Input dari sistem adalah sebuah standar input sebanyak *n* baris, dimana nilai dari *n* tidak melebihi 8 (yang merupakan batas maksimal jumlah baris *puzzle* sokoban) dan diakhiri dengan *end of file*. Input perbaris yang merupakan sebuah *string* akan disimpan terlebih dahulu ke dalam sebuah *vector* dan nantinya akan dinormalisasi lagi. *Pseudocode* dari fungsi yang mengambil standar input sistem ditunjukkan oleh Kode Sumber 3.19.

```
1.  FUNCTION runInput():
2.      inp <- []
3.
4.      WHILE s <- input():
5.          n_row <- n_row + 1
6.          n_col <- max(n_col, s.length)
7.
8.          inp.push_back(s)
9.      ENDWHILE
10.
11.     initPuzzle(inp)
12. ENDFUNCTION
```

Kode Sumber 3.19 *Pseudocode* Fungsi Run Input

3.10.1 Desain Fungsi Init Puzzle

Variabel *puzzle* yang merupakan variabel global awalnya hanya berisikan sebuah *string* kosong yang nilainya akan terus digabungkan dengan nilai input yang telah dinormalisasi. *Pseudocode* fungsi yang melakukan penggabungan ini ditunjukkan oleh Kode Sumber 3.20.

```
1.  FUNCTION initPuzzle(&inp):
2.      FOREACH s IN inp:
3.          row <- getRowNormalized(s)
4.          puzzle <- puzzle + row
5.      ENDFOREACH
6.  ENDFUNCTION
```

Kode Sumber 3.20 *Pseudocode* Fungsi Init Puzzle

3.10.1.1 Desain Fungsi Get Row Normalized

Normalisasi dari nilai input pada subbab 3.12.1 hanya berupa penambahan panjang dari input dengan beberapa karakter ‘ ’ (spasi) hingga panjangnya sama dengan n_col. *Pseudocode* fungsi yang melakukan penggabungan ini ditunjukkan oleh Kode Sumber 3.21.

```
1.  FUNCTION getRowNormalized(&row):
2.      row_normalized <- row
3.
4.      FOR i = row.length() to n_col:
5.          row_normalized <- row_normalized + " "
6.      ENDFOR
7.
8.      RETURN row_normalized
9.  ENDFUNCTION
```

Kode Sumber 3.21 *Pseudocode* Fungsi Get Row Normalized

3.11 Desain Variabel Global

Pada sistem ini, yang dimaksud variabel global adalah sebuah variabel yang digunakan oleh kedua *graph*. Nilai dari variabel-variabel global ini ada yang bernilai konstan dan ada pula yang nilainya relatif tergantung dari *puzzle* yang diberikan. Jenis-jenis variabel global berdasarkan penggunaannya akan dijelaskan lebih lanjut pada subbab dibawah.

3.11.1 Desain Batasan Global

Variabel global yang digunakan sebagai batasan tertentu, baik sebagai batas dalam pencarian maupun batas ukuran dari sebuah *array*. Nama variabel beserta nilai dan keterangan akan dijelaskan pada Tabel 3.3.

Tabel 3.3 Tabel Variabel Global Berjenis Batasan Global

Nama Variabel	Nilai	Keterangan
`_N_DIR`	4	Banyaknya arah perpindahan *box*
`MAX_LEN`	8	Nilai batasan untuk jumlah maksimal baris dan kolom *puzzle*
`MAX_PUSH_FORWARD`	*INF*	Banyaknya *push* yang diperbolehkan dalam setiap *graph*. Untuk membatasi dalamnya pencarian.
`MAX_PUSH_BACKWARD`	*INF*	Banyaknya *push* yang diperbolehkan dalam setiap *graph*. Untuk membatasi dalamnya pencarian.
`ULL_BIT_SIZE`	64	Ukuran bit dari tipe data **`unsigned long long`**

3.11.2 Desain Arah Konstan Global

Arah pergerakan *player* maupun *box* pada sistem didefinisikan dengan sebuah nilai integer konstan. Penjelasan tentang variabel global jenis ini diberikan pada Tabel 3.4.

Tabel 3.4 Tabel Variabel Global Berjenis Arah Konstan Global

Nama Variabel	Nilai	Keterangan
`U_DIR`	0	Arah atas
`R_DIR`	1	Arah kanan
`D_DIR`	2	Arah bawah
`L_DIR`	3	Arah kiri
`NONE`	4	Tidak berarah

3.11.3 Desain Ukuran Global

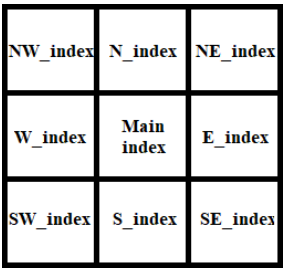
Jenis variabel global ini berguna untuk menyimpan informasi terkait ukuran *puzzle*. Penjelasan tentang variabel global jenis ini diberikan pada Tabel 3.5.

Tabel 3.5 Tabel Variabel Global Berjenis Ukuran Global

Nama Variabel	Nilai	Keterangan
`n_row`	$relatif$	Ukuran baris *puzzle*
`n_col`	$relatif$	Ukuran kolom *puzzle*
`n_attr`	$relatif$	Banyaknya pasangan *box* dan *goal*
`max_index`	$n\_row \times n\_col$	Nilai batas nomor indeks valid

3.11.4 Desain Arah Mata Angin Global

Variabel global ini berguna sebagai penampung nilai dari indeks yang berada disekitar indeks saat ini. Nama variabel beserta posisinya terhadap indeks utama diilustrasikan oleh Gambar 3.8.



Gambar 3.8 Variabel Global Berjenis Arah Mata Angin Global

3.11.5 Desain Representasi Bit Global

Semua variabel global jenis ini memiliki tipe data *unsigned long long*. Sama seperti posisi semua *box* pada sebuah *state*, ada beberapa atribut pada *puzzle* yang posisinya harus diperhatikan, contohnya dinding/*wall*. Posisi dari *wall* pada sebuah *puzzle* tidak akan pernah berubah dan sering digunakan, maka dari itu *wall* dan beberapa atribut yang sifatnya sejenis dijadikan sebagai variabel global. Semua nama variabel jenis ini beserta penjelasannya diberikan pada Tabel 3.6.

Tabel 3.6 Tabel Variabel Global Berjenis Representasi Bit Global

Nama Variabel	Nilai	Keterangan
`walls`	*relatif*	Posisi semua *wall* pada *puzzle*
`tunnels`	*relatif*	Posisi semua *tunnel* pada *puzzle*
`locks`	*relatif*	Posisi semua *locked index* pada *puzzle*

3.12 Desain Fungsi Init Global Variabels

Setelah nilai dari *puzzle* diinisialisasi, semua karakternya akan diiterasi dari posisi paling awal hingga akhir. Posisi dari tiap karakter akan menjadi nilai sebuah indeks, dan akan digunakan untuk menginisialisasi nilai dari semua variabel global, baik itu milik *forward move* ataupun milik *backward move*. *Pseudocode* fungsi ini ditunjukkan oleh Kode Sumber 3.22.

```
1.  FUNCTION initGlobalVariables():
2.      FOR i = 0 to len(puzzle):
3.          c <- puzzle[i]
4.
5.          SWITCH(c):
6.              CASE '#':
7.                  walls <- walls ∨ mask(i)
8.                  BREAK
9.              CASE '.':
10.                 forward_goals <- forward_goals ∨ mask(i
    )
11.                 forward_goal_list.push_back(i)
12.                 BREAK
13.             CASE '@':
14.                 initial_forward_player <- i
15.                 BREAK
16.             CASE '+':
17.                 initial_forward_player <- i
18.                 forward_goals <- forward_goals ∨ mask(i
    )
19.                 forward_goal_list.push_back(i)
20.                 BREAK
21.             CASE '$':
22.                 initial_forward_boxes <- initial_forwar
    d_boxes ∨ mask(i)
23.                 BREAK
24.             CASE '*':
25.                 initial_forward_boxes <- initial_forwar
    d_boxes ∨ mask(i)
26.                 forward_goals <- forward_goals ∨ mask(i
    )
27.                 forward_goal_list.push_back(i)
28.                 BREAK
29.             default:
30.                 BREAK
31.         ENDSWICTH
32.     ENDFOR
33.
```

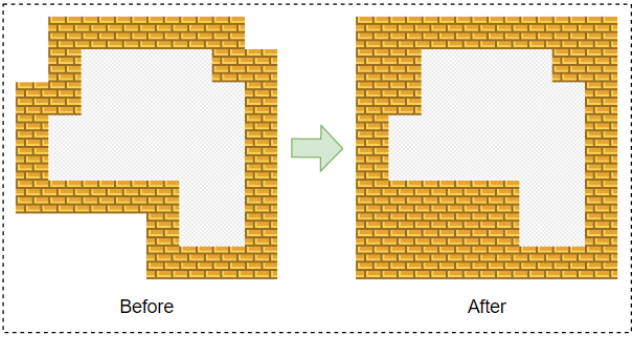
Kode Sumber 3.22a *Pseudocode* Fungsi Init Global Variables

```
34.     n_attr <- len(forward_goal_list)
35.     max_index <- n_col * n_row
36.     reinitWalls()
37.
38.     generateDeadlock()
39.     generateTunnels()
40.     createDistanceTable()
41.
42.     initForwardGlobalVariables()
43.     initBackwardGlobalVariables()
44. ENDFUNCTION
```

Kode Sumber 3.22b *Pseudocode* Fungsi Init Global Variables

3.13 Desain Fungsi Reinit Walls

Indeks yang merupakan sebuah *wall* merupakan sebuah indeks yang sama sekali tidak bisa dicapai oleh *player*. Begitu pula dengan semua indeks yang berada di luar area permainan pada *puzzle*. Untuk itu, area tersebut juga bisa dikatakan sebagai *wall*. Gambaran lebih jelas ditunjukkan oleh Gambar 3.9.



Gambar 3.9 Ilustrasi Proses Reinisialisasi *Walls*

Proses ini dilakukan dengan cara menyimulasikan pergerakan *player*. Dimulai dari posisi awal *player* pada *puzzle*, *player* akan terus digerakkan ke segala arah yang dapat dicapai dengan menggunakan algoritma BFS. *Pseudocode* fungsi Reinit Walls ditunjukkan oleh Kode Sumber 3.23.

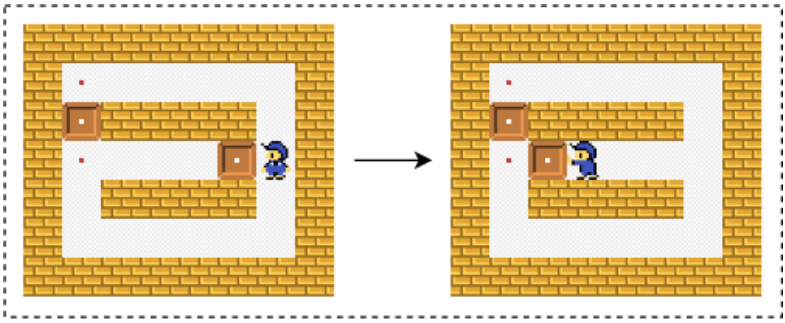
```
1.  FUNCTION reinitWalls():
2.      blocked_area <- ULLONG_MAX
3.      visited <- [SET DEFAULT false WITH LENGTH ULL_BIT_S
    IZE]
4.      tracks <- [initial_forward_player]
5.
6.      WHILE NOT tracks.empty():
7.          index <- tracks.front()
8.
9.          IF NOT visited[index] AND NOT isWall(index):
10.             blocked_area <- blocked_area ⊕ mask(index)

11.
12.             tracks.push_back(getIndexMoved(index, U_DIR
    ))
13.             tracks.push_back(getIndexMoved(index, R_DIR
    ))
14.             tracks.push_back(getIndexMoved(index, D_DIR
    ))
15.             tracks.push_back(getIndexMoved(index, L_DIR
    ))
16.
17.             visited[index] <- true
18.         ENDIF
19.
20.         tracks.pop_front()
21.     ENDWHILE
22.
23.     walls <- walls ∨ blocked_area
24. ENDFUNCTION
```

Kode Sumber 3.23 *Pseudocode* Fungsi Reinit Walls

3.14 Desain Fungsi Generate Tunnels

Terowongan/*tunnel* adalah indeks pada *puzzle* yang membatasi pergerakan player hanya sebatas bergerak secara horizontal atau vertikal. *Tunnel* juga menjadi sesuatu yang perlu diperhatikan, dikarenakan pergerakan player yang terbatas pada saat berada dalam indeks yang diindikasi sebagai *tunnel* bisa mengurangi jumlah *state* yang di-*generate*. Ilustrasi dari

pergerakan yang diakibatkan oleh *tunnel* dapat dilihat pada Gambar 3.10.



Gambar 3.10 Ilustrasi Proses Penentuan *State* Baru dengan Bantuan *Tunnel*

Box di-*push* ke arah kiri, namun tidak hanya sekali melainkan tiga kali. Hal ini dikarenakan *box* berada pada indeks yang merupakan sebuah *tunnel*. Sebuah indeks diindikasi sebagai sebuah *tunnel* jika memenuhi kondisi di mana semua indeks disekitarnya merupakan *wall*, kecuali dua buah indeks yang bisa membuat *player* yang berada pada indeks tersebut bisa bergerak secara vertikal atau horizontal. *Pseudocode* untuk fungsi yang membuat *tunnel* dapat dilihat pada Kode Sumber 3.24.

```
1.  FUNCTION generateTunnels():
2.      FOR index = 0 to max_index:
3.          IF NOT isWall(index):
4.              assignWindsDirection(index)
5.
6.              IF isHorizontalTunnel() OR isVerticalTunnel
    ():
7.                  tunnels <- tunnels ∨ mask(index)
8.              ENDIF
9.          ENDIF
10.     ENDFOR
11. ENDFUNCTION
```

Kode Sumber 3.24 *Pseudocode* Fungsi Generate Tunnels

3.14.1 Desain Fungsi Is Horizontal Tunnel

Fungsi ini berguna untuk mendeteksi *tunnel* pada posisi horizontal terhadap sebuah indeks. *Pseudocode* fungsi ini dapat dilihat pada Kode Sumber 3.25.

```
1.  FUNCTION isHorizontalTunnel():
2.      '''''
3.      ###
4.       $
5.      ###
6.      '''
7.      RETURN true
8.          AND isWall(NW_index) AND isWall(N_index)
9.          AND isWall(NE_index) AND isWall(SW_index)
10.         AND isWall(S_index) AND isWall(SE_index)
11. ENDFUNCTION
```

Kode Sumber 3.25 *Pseudocode* Fungsi Is Horizontal Tunnel

3.14.2 Desain Fungsi Is Vertical Tunnel

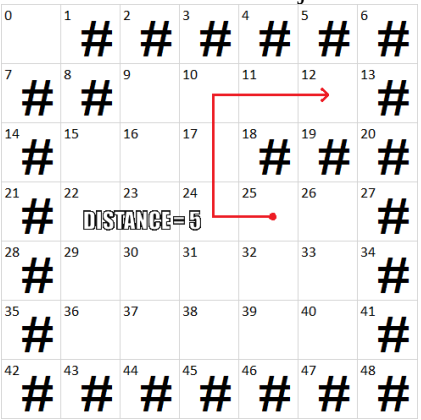
Fungsi ini berguna untuk mendeteksi tunnel pada posisi vertikal terhadap sebuah indeks. *Pseudocode* fungsi ini dapat dilihat pada Kode Sumber 3.26.

```
1.  FUNCTION isVerticalTunnel():
2.      '''''
3.      # #
4.      #$#
5.      # #
6.      '''
7.      RETURN true
8.          AND isWall(NW_index) AND isWall(W_index)
9.          AND isWall(SW_index) AND isWall(NE_index)
10.         AND isWall(E_index) AND isWall(SE_index)
11. ENDFUNCTION
```

Kode Sumber 3.26 *Pseudocode* Fungsi Is Vertical Tunnel

3.15 Desain Tabel Jarak

Tabel jarak/*distance table* digunakan sebagai *look up* jarak antara satu indeks dengan indeks lainnya. Tabel ini berukuran (**n_row** * **n_col**)^2. Jarak yang tersimpan pada tabel ini bukanlah jarak *manhattan* maupun *euclidean*, melainkan jarak tempuh sebenarnya dari *player* pada sebuah *puzzle* dengan asumsi tidak adanya halangan dalam mencapai jarak terpendeknya. Gambaran jarak dari satu indeks ke indeks lain ditunjukkan oleh Gambar 3.11.



Gambar 3.11 Jarak Antara Dua Indeks pada Sebuah *State*

3.15.1 Desain Fungsi Init Table

Nilai awal semua elemen dari tabel jarak adalah *infinity* (atau INF yang juga dilambangkan dengan INT_MAX), kecuali untuk jarak suatu indeks ke dirinya sendiri bernilai 0 dan jarak untuk menuju tetangganya (sebelah atas, bawah, kiri, dan kanan) yang bukan merupakan *wall* bernilai 1. *Pseudocode* dari fungsi Init Table dapat dilihat pada Kode Sumber 3.27.

```
1.  FUNCTION initTable() {
2.      table <- [SET DEFAULT [SET DEFAULT oo WITH LENGTH m
    ax_index] WITH LENGTH max_index]
3.
4.      FOR i = 0 to max_index:
5.          table[i][i] = 0
6.      ENDFOR
7.
8.      FOR i = 0 to max_index:
9.          IF NOT isWall(i):
10.             U_index <- getIndexMoved(i, U_DIR)
11.
12.             IF isValidIndexMove(i, U_index, U_DIR) AND
    NOT isWall(U_index):
13.                 table[i][U_index] <- 1
14.             ENDIF
15.
16.             R_index <- getIndexMoved(i, R_DIR)
17.
18.             IF isValidIndexMove(i, R_index, R_DIR) AND
    NOT isWall(R_index):
19.                 table[i][R_index] <- 1
20.             ENDIF
21.
22.             D_index <- getIndexMoved(i, D_DIR)
23.
24.             IF isValidIndexMove(i, D_index, D_DIR) AND
    NOT isWall(D_index):
25.                 table[i][D_index] <- 1
26.             ENDIF
27.
28.             L_index <- getIndexMoved(i, L_DIR)
29.
30.             IF isValidIndexMove(i, L_index, L_DIR) AND
    NOT isWall(L_index):
31.                 table[i][L_index] <- 1
32.             ENDIF
33.         ENDIF
34.     ENDFOR
35. ENDFUNCTION
```

Kode Sumber 3.27 *Pseudocode* Fungsi Init Table

3.15.2 Desain Fungsi Create Distance Table

Tabel jarak dibentuk menggunakan algoritma Floyd. *Pseudocode* dari fungsi ini dapat dilihat pada Kode Sumber 3.28.

```
1.  FUNCTION createDistanceTable():
2.      initTable()
3.
4.      FOR i = 0 to max_index:
5.          updateShortestPath(i)
6.      ENDFOR
7.  ENDFUNCTION
```

Kode Sumber 3.28 *Pseudocode* Fungsi Create Distance Table

3.16 Desain Fungsi Init Forward Global Variables

Sebelum memulai pencarian solusi dengan langkah maju/*forward move*, *state* awal beserta semua atribut pendukungnya harus diinisialisasi terlebih dahulu. Penjelasan dari semua variabel yang digunakan pada *forward move* dapat dilihat pada Tabel 3.7.

Tabel 3.7 Tabel Variabel pada *Forward Move*

Nama Variabel	Tipe Data	Keterangan
`Initial_ forward_ player`	`int`	Indeks *player* pada state pertama *forward move*
`forward_path_ map_root`	`string`	*Root key* pada *graph* untuk *path forward move*
`forward_ goals`	`ULL`	Representasi bit semua posisi *goal* pada *forward move*
`initial_ forward_ boxes`	`ULL`	Representasi bit semua posisi *box* pada *state* pertama *forward move*
`forward_ state_map`	`map<ULL, map<int,bool>>`	Menyimpan semua *state* yang telah dikunjungi pada *forward move*
`forward_ path_map`	`map<string, string>`	*Graph* untuk *path* pada *forward move*
`forward_ tracks`	`priority_queue <NODE, vector<NODE>, NodeCompare>`	*Graph* pencarian solusi dengan *forward move*
`forward_goal_ list`	`vector<int>`	Semua indeks *goal* pada *forward move*

Beberapa atribut untuk *forward move* telah diinisialisasi pada fungsi Init Global Variables. Untuk variabel lain, nilainya akan diinisialisasi pada fungsi Init Forward Global Variables. *Pseudocode* fungsi ini ditunjukkan oleh Kode Sumber 3.29.

```
1.  FUNCTION initForwardGlobalVariables():
2.      forward_path_map_root <- getPathMapKey(initial_forw
    ard_boxes, initial_forward_player)
3.
4.      clearTemplate(forward_path_map)
5.      clearTemplate(forward_tracks)
6.  ENDFUNCTION
```

Kode Sumber 3.29 *Pseudocode* Fungsi Init Forward Global Variables

3.17 Desain Fungsi Init Backward Global Variables

Sama halnya dengan *forward move*, semua atribut pada langkah mundur/*backward move* juga harus diinisialisasi. Penjelasan dari semua variabel yang digunakan pada *backward move* dapat dilihat pada Tabel 3.8.

Tabel 3.8 Tabel Variabel pada *Backward Move*

Nama Variabel	Tipe Data	Keterangan
backward_path_map_root	string	*Root key* pada *graph* untuk *path backward move*
backward_goals	ULL	Representasi bit semua posisi *goal* pada *backward move*
initial_backward_boxes	ULL	Representasi bit semua posisi *box* pada *state* pertama *backward move*
backward_state_map	map<ULL, map<int,bool> >	Memorisasi semua *state* yang telah dikunjungi pada *backward move*
backward_path_map	map<string, string>	*Graph* untuk *path* pada *backward move*
backward_tracks	list<NODE>	*Graph* pencarian solusi dengan *backward move*
backward_goal_list	vector<int>	Semua indeks *goal* pada *backward move*

Inisialiasi nilai dari semua variabel ini terdapat pada fungsi Init Backward Global Variables. *Pseudocode* dari fungsi ini ditampilkan pada Kode Sumber 3.30.

```
1.  FUNCTION initBackwardGlobalVariables():
2.      initial_backward_player <- 0
3.
4.      backward_goals <- initial_forward_boxes
5.      initial_backward_boxes <- forward_goals
6.      backward_path_map_root <- getPathMapKey(initial_bac
    kward_boxes, initial_backward_player)
7.
8.      clearTemplate(backward_path_map)
9.      clearTemplate(backward_tracks)
10.
11.     convertToVector(backward_goals, backward_goal_list)

12. ENDFUNCTION
```

Kode Sumber 3.30 *Pseudocode* Fungsi Init Backward Global Variables

3.18 Desain Fungsi Add Initial Node

Sebelum memulai pencarian solusi, setiap *graph* pencarian harus diinisialisasi dengan *node* awal. Fungsi Add Initial Node ini hanya membungkus fungsi lain di dalamnya, sebelum setiap *graph* pencarian benar-benar akan dimasukkan nilai awalnya. *Pseudocode* dari fungsi ini ditunjukkan oleh Kode Sumber 3.31.

```
1.  FUNCTION addInitialNode():
2.      addForwardInitialNode()
3.      addBackwardInitialNode()
4.  ENDFUNCTION
```

Kode Sumber 3.31 *Pseudocode* Fungsi Add Initial Node

3.18.1 Desain Fungsi Add Forward Initial Node

Fungsi ini berguna untuk memasukan nilai awal dari *graph* pencarian *forward move*. *Node* awal dari *forward move* hanya ada satu, yaitu sebuah *node* yang memuat *state* berupa *puzzle* dari sebenarnya. *Pseudocode* dari fungsi Add Forward Initial Node ditunjukkan oleh Kode Sumber 3.32.

```
1.  FUNCTION addForwardInitialNode():
2.      heuristic <- 0
3.      push <- 0
4.      boxes <- initial_forward_boxes
5.      player <- initial_forward_player
6.
7.      node <- getNewNode(heuristic, push, boxes, player)

8.
9.      markState(node, true)
10.     forward_tracks.push(node)
11. ENDFUNCTION
```

Kode Sumber 3.32 *Pseudocode* Fungsi Add Forward Initial Node

3.18.2 Desain Fungsi Add Backward Initial Node

Fungsi ini berguna untuk memasukan nilai awal dari *graph* pencarian *backward move*. Pada *backward move*, *graph* pencarian tidak hanya memiliki satu *node* awal, dikarenakan *backward move* tidak memiliki nilai awal dari *player* seperti *forward move*. Fungsi ini melakukan hal yang sama dengan men-*generate* sebuah *state* baru pada *backward move* (dengan cara melakukan *action pull*). Bedanya, *state* yang dibentuk tidak hanya dari sebuah *box*, melainkan dari semua *box*. *Pseudocode* dari fungsi Add Backward Initial Node ditunjukkan oleh Kode Sumber 3.33.

```
1.  FUNCTION addBackwardInitialNode():
2.      box_list <- []
3.      convertToVector(initial_backward_boxes, box_list)
4.      heuristic <- 0
5.      push <- 1
6.
7.      FOREACH box IN box_list:
8.          directions <- [U_DIR, R_DIR, D_DIR, L_DIR]
9.          push_paths <- ['D', 'L', 'U', 'R']
10.
11.         FOR i = 0 to len(directions):
12.             dir <- directions[i]
13.             f1_side <- getIndexMoved(box, dir)
14.             f2_side <- getIndexMoved(f1_side, dir)
15.
16.             IF NOT isObstacle(initial_backward_boxes, f
    1_side)
17.                 AND NOT isObstacle(initial_backward_box
    es, f2_side):
18.                 path <- push_paths[i]
19.                 boxes <- getBoxesMoved(initial_backward
    _boxes, box, dir)
20.                 player <- f2_side
21.
22.                 node <- getNewNode(heuristic, push, box
    es, player)
23.                 backward_tracks.push_back(node)
24.
25.                 key <- getPathMapKey(boxes, player)
26.                 markPath(key, backward_path_map_root, p
    ath, false)
27.                 markState(node, false)
28.             ENDIF
29.         ENDFOR
30.     ENDFOREACH
31. ENDFUNCTION
```

Kode Sumber 3.33 *Pseudocode* Fungsi Add Backward Initial Node

3.19 Desain Fungsi Run Forward Move

Fungsi inilah yang menjalankan pencarian *graph* pada *forward move*. *Pseudocode* dari fungsi ini ditunjukkan oleh Kode Sumber 3.34.

```
1.  FUNCTION runForwardMove(&win):
2.      IF NOT forward_tracks.empty():
3.          node <- forward_tracks.top()
4.
5.          forward_tracks.pop();
6.
7.          successors <- []
8.          generateSuccessors(node, successors, true)
9.
10.         FOREACH successor IN successors:
11.             markState(successor, true)
12.
13.             IF isGameOver(successor , true):
14.                 boxes <- _n_boxes(successor)
15.                 player <- _n_player(successor)
16.
17.                 key <- getPathMapKey(boxes, player)
18.                 ans <- getFullPath(key, true)
19.
20.                 PRINT ans
21.
22.                 win <- true
23.                 BREAK
24.             ENDIF
25.
26.             checkMeetInTheMiddle(successor, win, true)
27.
28.             IF NOT win:
29.                 forward_tracks.push(successor)
30.             ELSE:
31.                 BREAK
32.             ENDIF
33.         ENDFOREACH
34.     ENDIF
35. ENDFUNCTION
```

Kode Sumber 3.34 *Pseudocode* Fungsi Run Forward Move

3.19.1 Desain Fungsi Generate Forward Box Successors

Node yang merupakan *successor* berasal dari *action push* dari setiap *box* yang ada. Ada beberapa hal yang harus diperhatikan sebelum suatu *node* dapat di-*generate* sebagai sebuah *successor*, yaitu:

1. Nilai atribut *push* pada *node successor* adalah atribut *push* saat ini ditambah dengan satu. Apabila nilai *push node successor* telah melebihi batas yang diberikan, maka *node* tidak akan di-*generate*.
2. Kedua sisi yang segaris dengan arah perpindahan *box* haruslah bukan berupa hambatan/*obstacle*. Untuk memastikan bahwa indeks tempat *box* setelah di-*push* indeks *player* sebelum melakukan *action push* dapat ditempati. Sebagai contoh, apabila suatu *box* akan di-*push* ke bawah, maka indeks di atas dan di bawah *box* haruslah bukan berupa *obstacle*.
3. Posisi *player* sebelum melakukan *action push* bisa dicapai oleh posisi *player* pada *node* sebelumnya. Hal ini dapat dideteksi dengan mencari *path* dari kedua indeks tersebut. Apabila *path* tidak ditemukan, maka *node successor* tidak bisa didapat dengan cara melakukan *action push* pada arah yang diinginkan.
4. *State* akan di-*generate* belum pernah dikunjungi sama sekali.
5. Nilai dari *heuristic* lebih dari atau sama dengan nol. *Heuristic* yang valid adalah *heuristic* yang tidak bernilai tak terhingga/*infinity*.
6. Memeriksa posisi *box* apakah berada di dalam *tunnel* atau tidak. Apabila kondisi terpenuhi, maka *node* saat ini akan diperbarui dengan menjalankan *macro tunnel move*. Apabila *state* pada *node* yang telah diperbarui masih merupakan sebuah *node* yang valid, maka *node* tersebut resmi menjadi sebuah *successor node*.

Pseudocode dari fungsi Generate Forward Box Successors ditunjukkan oleh Kode Sumber 3.35.

```
1.  FUNCTION generateForwardBoxSuccessors(&box, &node, &suc
    cessors):
2.      push <- _n_push(node) + 1
3.
4.      IF push GREATER THAN MAX_PUSH_FORWARD)
5.          RETURN
6.      ENDIF
7.
8.      parent_key <- getPathMapKey(_n_boxes(node), _n_play
    er(node))
9.      directions <- [D_DIR, L_DIR, U_DIR, R_DIR]
10.     push_paths <- ['D', 'L', 'U', 'R']
11.
12.     FOR i = 0 to len(directions):
13.         f_dir <- directions[i]
14.         b_dir <- directions[(i + 2) % _N_DIR]
15.
16.         f_side <- getIndexMoved(box, f_dir)
17.         b_side <- getIndexMoved(box, b_dir)
18.
19.         IF NOT isObstacle(_n_boxes(node), f_side)
20.             AND NOT isObstacle(_n_boxes(node), b_side):

21.             p_to_bs <- getShortestPath(_n_boxes(node),
    _n_player(node), b_side)
22.
23.             IF p_to_bs NOT EQUAL TO "-":
24.                 boxes <- getBoxesMoved(_n_boxes(node),
    box, f_dir)
25.                 player <- box
26.
27.                 IF NOT isStateSeen(boxes, player, true)
    :
28.                     heuristic = heuristicValue(boxes, p
    layer, true)
```

Kode Sumber 3.35a *Pseudocode* Fungsi Generate Forward Box Successors

```
29.                     path = p_to_bs + push_paths[i]
30.
31.                     IF heuristic NOT EQUAL TO oo
32.                         node_new <- getNewNode(heuristi
    c, push, boxes, player)
33.                         box_pos_new <- getIndexMoved(bo
    x, f_dir)
34.                         valid <- doMacroTunnel(node_new
    , box_pos_new, f_dir, path, true)
35.
36.                         IF valid AND NOT isStateSeen(_n
    _boxes(node_new), _n_player(node_new), true)):
37.                             key <- getPathMapKey(_n_box
    es(node_new), _n_player(node_new))
38.                             markPath(key, parent_key, p
    ath, true)
39.
40.                             successors.push_back(node_n
    ew)
41.                         ENDIF
42.                     ENDIF
43.                 ENDIF
44.             ENDIF
45.         ENDIF
46.     ENDFOR
47. ENDFUNCTION
```

Kode Sumber 3.35b *Pseudocode* Fungsi Generate Forward Box Successors

3.19.2 Desain Check Meet In The Middle untuk Forward Move

Pada *forward move*, pemeriksaan kondisi *meet in the middle* hanya berupa keterhubungan antara *state* saat ini dengan *state* pada *backward move*. Untuk setiap *node* pada *backward move* yang memiliki nilai *boxes* yang sama dengan *node* saat ini, dilakukan pencarian *path* dari posisi *player forward move* pada tiap player pada *node-node backward move*. Apabila *path* ditemukan, maka kedua *node* tersebut dapat terhubung dan solusi ditemukan.

Pseudocode dari fungsi Check Meet In The Middle untuk *forward move* ditunjukkan oleh Kode Sumber 3.36.

```
1.  FUNCTION checkMeetInTheMiddle(&node, &win, forward_chec
    k <- true):
2.      IF forward_check:
3.          IF backward_state_map.find(_n_boxes(node)) NOT
    EQUAL TO backward_state_map.end():
4.              temp <- &backward_state_map[_n_boxes(node)]

5.
6.              FOR it = temp->begin() to temp->end():
7.                  backward_player <- it->first
8.                  middle_path <- getShortestPath(_n_boxes
    (node), _n_player(node), backward_player)
9.
10.                 IF middle_path NOT EQUAL TO "-":
11.                     forward_key <- getPathMapKey(_n_box
    es(node), _n_player(node))
12.                     forward_path <- getFullPath(forward
    _key, true)
13.
14.                     backward_key <- getPathMapKey(_n_bo
    xes(node), backward_player)
15.                     backward_path <- getFullPath(backwa
    rd_key, false)
16.
17.                     ans <- forward_path + middle_path +
     backward_path
18.                     PRINT ans
19.
20.                     win <- true
21.                     BREAK
22.                 ENDIF
23.             ENDFOR
24.         ENDIF
25.     ELSE:
26.         ...
27.     ENDIF
28. ENDFUNCTION
```

Kode Sumber 3.36 *Pseudocode* Check Meet In The Middle untuk *Forward Move*

3.20 Desain Fungsi Run Backwad Move

Fungsi inilah yang menjalankan pencarian *graph* pada *backward move*. *Pseudocode* dari fungsi ini ditunjukkan oleh Kode Sumber 3.37.

```
1.  FUNCTION runBackwardMove(&win):
2.      IF NOT backward_tracks.empty():
3.          node <- backward_tracks.front()
4.
5.          backward_tracks.pop_front()
6.
7.          generateSuccessors(node, successors, false)
8.
9.          FOREACH successor IN successors:
10.             markState(successor, false)
11.
12.             IF isGameOver(successor , false):
13.                 boxes <- _n_boxes(successor)
14.                 player <- _n_player(successor)
15.                 prefix_path <- getShortestPath(boxes, i
    nitial_forward_player, player)
16.
17.                 IF prefix_path NOT EQUAL TO "-":
18.                     key <- getPathMapKey(boxes, player)

19.                     ans <- prefix_path + getFullPath(ke
    y, false)
20.
21.                     PRINT ans
22.
23.                     win <- true
24.                     BREAK
25.                 ELSE
26.                     CONTINUE
27.                 ENDIF
28.             ENDIF
29.
```

Kode Sumber 3.37a *Pseudocode* Fungsi Run Backward Move

```
30.             checkMeetInTheMiddle(successor, win, false)
31.
32.             IF NOT win:
33.                 backward_tracks.push_back(successor)
34.             ELSE
35.                 BREAK
36.             ENDIF
37.         ENDFOREACH
38.     ENDIF
39. ENDFUNCTION
```

Kode Sumber 3.37b *Pseudocode* Fungsi Run Backward Move

3.20.1 Desain Fungsi Generate Backward Box Successors

Node yang merupakan *successor* berasal dari *action pull* dari setiap *box* yang ada. Ada beberapa hal yang harus diperhatikan sebelum suatu *node* dapat di-*generate* sebagai sebuah *successor* pada *backward move*, yaitu:

1. Nilai atribut *pull* pada *node successor* adalah atribut *pull* saat ini ditambah dengan satu. Sebuah *action pull* merupakan hal yang sama *action push* pada *forward move*. Apabila nilai *pull node successor* telah melebihi batas yang diberikan, maka *node* tidak akan di-*generate*.
2. Kedua indeks yang searah dengan *action pull* dari *box* haruslah bukan berupa hambatan/*obstacle*. Untuk memastikan bahwa indeks tempat *box* setelah di-*pull* dan indeks *player* setelah melakukan *pull* dapat ditempati. Sebagai contoh, apabila suatu *box* akan di-*pull* ke atas, maka dua indeks di atas *box* haruslah bukan berupa *obstacle*.
3. *State* akan di-*generate* belum pernah dikunjungi sama sekali.
4. Posisi *player* sebelum melakukan *action pull* bisa mencapai posisi *player* pada *node* sebelumnya. Hal ini dapat dideteksi dengan mencari *path* dari kedua indeks tersebut. Apabila *path* tidak ditemukan, maka *node*

successor tidak bisa didapat dengan cara melakukan *action pull* pada arah yang diinginkan.

5. Memeriksa posisi *box* apakah berada di dalam *tunnel* atau tidak. Apabila kondisi terpenuhi, maka *node* saat ini akan diperbarui dengan menjalankan *macro tunnel move*. Apabila *state* pada *node* yang telah diperbarui masih merupakan sebuah *node* yang valid, maka *node* tersebut resmi menjadi sebuah *successor node*.

Pseudocode dari fungsi Generate Backward Box Successors ditunjukkan oleh Kode Sumber 3.38.

```
1.  FUNCTION generateBackwardBoxSuccessors(&box, &node, &su
    ccessors):
2.      heuristic <- 0
3.      push <- _n_push(node) + 1
4.
5.      IF push GREATER THAN MAX_PUSH_BACKWARD:
6.          RETURN
7.      ENDIF
8.
9.      parent_key <- getPathMapKey(_n_boxes(node), _n_play
    er(node))
10.     directions <- [U_DIR, R_DIR, D_DIR, L_DIR]
11.     push_paths <- ['D', 'L', 'U', 'R']
12.
13.     FOR i = 0 to len(directions):
14.         dir <- directions[i]
15.
16.         f1_side <- getIndexMoved(box, dir)
17.         f2_side <- getIndexMoved(f1_side, dir)
18.
19.         IF NOT isObstacle(_n_boxes(node), f1_side)
20.             AND NOT isObstacle(_n_boxes(node), f2_side)
    ):
21.             boxes <- getBoxesMoved(_n_boxes(node), box,
     dir)
```

Kode Sumber 3.38a *Pseudocode* Fungsi Generate Backward Box Successors

```
22.             player <- f2_side
23.
24.             IF NOT isStateSeen(boxes, player, false):
25.                 f1s_to_p <- getShortestPath(_n_boxes(no
    de), f1_side, _n_player(node));
26.
27.                 IF f1s_to_p NOT EQUAL TO "-":
28.                     path <- push_paths[i] + f1s_to_p
29.                     node_new <- getNewNode(heuristic, p
    ush, boxes, player)
30.                     box_pos_new <- getIndexMoved(box, d
    ir)
31.                     valid <- doMacroTunnel(node_new, bo
    x_pos_new, dir, path, false)
32.
33.                     IF valid AND NOT isStateSeen(_n_box
    es(node_new), _n_player(node_new), false):
34.                         key <- getPathMapKey(_n_boxes(n
    ode_new), _n_player(node_new))
35.                         markPath(key, parent_key, path,
     false)
36.
37.                         successors.push_back(node_new)

38.                     ENDIF
39.                 ENDIF
40.             ENDIF
41.         ENDIF
42.     ENDFOR
43. ENDFUNCTION
```

Kode Sumber 3.38b *Pseudocode* Fungsi Generate Backward Box Successors

3.20.2 Desain Fungsi Check Meet In The Middle untuk Backward Move

Sama halnya dengan *forward move*, *node* saat ini pada *backward move* akan diperiksa keterhubungannya dengan *node-node* pada *graph* lainnya. Setiap *node* pada *forward move* yang memiliki nilai *boxes* yang sama dengan *node* saat ini, dilakukan

pencarian *path* dari posisi *player forward move* pada tiap *player* pada *node-node backward move*. Apabila *path* ditemukan, maka kedua *node* tersebut dapat terhubung dan solusi ditemukan. *Pseudocode* dari fungsi Check Backward Meet The Middle untuk *backward move* ditunjukkan oleh Kode Sumber 3.39.

```
1.  FUNCTION checkMeetInTheMiddle(&node, &win, forward_chec
    k <- true):
2.      IF forward_check:
3.          ...
4.      ELSE:
5.          IF forward_state_map.find(_n_boxes(node)) NOT E
    QUAL TO forward_state_map.end():
6.              temp <- &forward_state_map[_n_boxes(node)]

7.
8.              FOR it = temp->begin() to temp->end():
9.                  forward_player <- it->first
10.                 middle_path <- getShortestPath(_n_boxes
    (node), forward_player, _n_player(node))
11.
12.                 IF middle_path NOT EQUAL TO "-":
13.                     forward_key <- getPathMapKey(_n_box
    es(node), forward_player)
14.                     forward_path <- getFullPath(forward
    _key, true)
15.
16.                     backward_key <- getPathMapKey(_n_bo
    xes(node), _n_player(node))
17.                     backward_path <- getFullPath(backwa
    rd_key, false)
18.
19.                     ans <- forward_path + middle_path +
     backward_path
20.                     PRINT ans
21.
22.                     win <- true
23.                     BREAK
24.                 ENDIF
25.             ENDFOR
26.         ENDIF
27.     ENDIF
28. ENDFUNCTION
```

Kode Sumber 3.39 *Pseudocode* Check Meet In The Middle untuk *Backward Move*

3.21 Desain Solusi Alternatif

Solusi alternatif dibutuhkan untuk membandingkan sistem saat ini dengan sistem lain yang mungkin memberikan hasil yang lebih baik dengan algoritma yang berbeda. Solusi yang akan dijelaskan pada subbab berikutnya sempat diimplementasikan. Perbandingan hasil uji coba menggunakan *dataset* yang dimiliki oleh penulis akan dicantumkan pada bab 5 nantinya.

3.21.1 Desain Bidirectional Search DFS dan BFS

Solusi ini menggunakan algoritma DFS pada *graph* milik *forward move*. Dikarenakan algoritma DFS tidak membutuhkan *heuristic*, struktur *node* untuk *forward move* dari solusi ini sama seperti milik *backward move*-nya. Serta struktur data yang digunakan adalah *stack* bukan *priority queue*.

3.21.2 Desain Bidirectional Search Greedy DFS dan BFS

Sama halnya dengan *bidirectional search* DFS dan BFS, solusi ini juga menggunakan struktur data *stack* untuk *forward move*. Namun, atribut *heuristic* masih digunakan. *Node successors* yang sudah di-*generate*, akan diurutkan terlebih dahulu sehingga *node* yang memiliki nilai *heuristic* terbesar akan dimasukkan ke dalam *stack* sebelum *node* dengan *heuristic* yang lebih kecil. Hasilnya, *node* dengan *heuristic* yang lebih kecil akan diproses terlebih dahulu.

BAB IV
IMPLEMENTASI

Bab ini menjelaskan mengenai implementasi perangkat lunak dari rancangan sistem yang telah dibahas pada Bab 3 meliputi kode program dalam perangkat lunak.

4.1 Lingkungan Implementasi

Lingkungan implementasi dan pengembangan yang dilakukan adalah sebagai berikut.

1. Perangkat Keras
    a. *Processor* Intel Core i7-7700HQ CPU @ 2.80GHz (8 CPUs), ~2.8GHz
    b. *Random Access Memory* 8GB
2. Perangkat Lunak
    a. Sistem Operasi Windows 10 Pro 64-bit
    b. DevC++
    c. C++ 4.9.2
    d. Python 3.6

4.2 Implementasi Program Utama

Subbab ini menjelaskan implementasi program utama. Program ini merupakan program yang digunakan untuk menyelesaikan permasalahan 1589 - Sokoban. Program diimplementasikan dengan menggunakan bahasa pemrograman C++ karena dalam pengimplementasian algoritma solusi dibutuhkan batas waktu dan memori yang sangat terbatas untuk komputasi yang sangat besar.

4.2.1 Penggunaan *Library*

Program ini menggunakan *library* seperti yang ditunjukkan pada Kode Sumber 4.1.

```
#include <algorithm>
#include <cassert>
#include <climits>
#include <cstdio>
#include <cstring>
#include <iostream>
#include <list>
#include <map>
#include <queue>
#include <string>
#include <vector>
```

Kode Sumber 4.1 *Header* Program Utama

4.2.2 Penggunaan Preprocessor Directives

Penggunaan *preprocessor directives* ditujukan untuk mempermudah dan mempersingkat implementasi. Penggunaan *preprocessor directives* dapat dilihat pada Kode Sumber 4.2.

```
#define oo INT_MAX

#define _N_DIR 4
#define MAX_LEN 8
#define MAX_PUSH_FORWARD 37
#define MAX_PUSH_BACKWARD oo
#define OVERLAP_VALUE 1000
#define ULL_BIT_SIZE 64

#define mask(n) (1ULL << (n))
```

Kode Sumber 4.2 *Preprocessor Directives* Program Utama

4.2.3 Penggunaan Reserved Keyword

Reserved keyword digunakan untuk mengganti nama sebuah tipe data. Sama halnya dengan *preprocessor directives*, *reserved keyword* juga digunakan untuk mempermudah dan mempersingkat implementasi. Penggunaan *preprocessor directives* dapat dilihat pada Kode Sumber 4.3

```
typedef unsigned long long ULL;
```

Kode Sumber 4.3 *Reserved Keyword* Program Utama

4.2.4 Implementasi Fungsi Main

Fungsi Main nantinya menjadi fungsi yang dijalankan pertama kali oleh sistem. Fungsi ini hanya berisikan pemanggilan fungsi lainnya yang berguna untuk mendapatkan input standar dan menyelesaikan *puzzle*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.4.

```
int main()
{
    runInput();
    solve();

    return 0;
}
```

Kode Sumber 4.4 Implementasi Fungsi Main

4.3 Implementasi Fungsi Struktur *Node*

Subbab ini menjelaskan implementasi fungsi-fungsi yang mempengaruhi kedudukan sebuah *node* dalam *graph* pencarian, khususnya pada nilai *heuristic*. Program yang terdapat pada subbab ini menggunakan desain yang dijelaskan pada pada bagian 3.3.

4.3.1 Implementasi Fungsi Heuristic Value

Subbab ini menjabarkan implementasi fungsi Heuristic Value yang sudah dijelaskan pada subbab 3.3.1. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.5.

```
1.  int heuristicValue(ULL &boxes, int &player, bool forwar
    d_check = true) {
2.      if (isDeadlock(boxes))
3.          return oo;
4.
5.      static vector<int> box_list;
6.      convertToVector(boxes, box_list);
7.
8.      return greedyBipartiteHeuristic(box_list, forward_c
    heck);
9.  }
```

Kode Sumber 4.5 Implementasi Fungsi Heuristic Value

4.3.1.1 Implementasi Fungsi Greedy Bipartite Heuristic

Subbab ini menjabarkan implementasi fungsi Greedy Bipartite Heuristic yang sudah dijelaskan pada subbab 3.3.1. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.6.

```
1.  int greedyBipartiteHeuristic(vector<int> &box_list,
    bool forward_check) {
2.      static vector<BGEDGE> bgedges;
3.      clearTemplate(bgedges);
4.
5.      for (int i = 0; i < n_attr; i++)
6.          searchBoxGoals(box_list[i], i, bgedges, forward
    _check);
7.      sort(bgedges.begin(), bgedges.end(), bgStateComp);

8.
9.      static bool box_marked[ULL_BIT_SIZE];
10.     static bool goal_marked[ULL_BIT_SIZE];
11.
12.     memsetTemplate(box_marked, false, n_attr);
13.     memsetTemplate(goal_marked, false, n_attr);
14.
15.     static int counter;
16.     static int score;
```

Kode Sumber 4.6a Implementasi Fungsi Greedy Bipartite Heuristic

```
17.     static int overlap_count;
18.
19.     counter = 0;
20.     score = 0;
21.     overlap_count = 0;
22.
23.     for (BGEDGE bgedge: bgedges) {
24.         static int dist;
25.         static int box_idx;
26.         static int goal_idx;
27.
28.         dist = _bge_weight(bgedge);
29.         box_idx = _bge_box_idx(bgedge);
30.         goal_idx = _bge_goal_idx(bgedge);
31.
32.         // If either box_index or goal_index is not tak
    en
33.         if (!box_marked[box_idx] && !goal_marked[goal_i
    dx]) {
34.             box_marked[box_idx] = true;
35.             goal_marked[goal_idx] = true;
36.             score += dist;
37.             counter++;
38.
39.             if (dist == 0)
40.                 overlap_count++;
41.
42.             if (counter == n_attr)
43.                 break;
44.         }
45.     }
46.
47.     return score - (OVERLAP_VALUE * overlap_count);
48. }
```

Kode Sumber 4.6b Implementasi Fungsi Greedy Bipartite Heuristic

4.3.1.2 Implementasi Fungsi Search Box Goals

Subbab ini menjabarkan implementasi fungsi Search Box Goals yang sudah dijelaskan pada subbab 3.3.1.2. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.7.

```
1.  void searchBoxGoals(int &box_index, int &box_idx, vecto
    r<BGEDGE> &bgedges, bool forward_check) {
2.      static vector<int> *goal_list;
3.      goal_list = forward_check ? &forward_goal_list : &b
    ackward_goal_list;
4.
5.      for (int i = 0; i < n_attr; i++) {
6.          static int goal_index;
7.          goal_index = (*goal_list)[i];
8.
9.          bgedges.push_back(
10.             make_pair(
11.                 table[box_index][goal_index],
12.                 make_pair(box_idx, i)
13.             )
14.         );
15.     }
16. }
```

Kode Sumber 4.7 Implementasi Fungsi Search Box Goals

4.3.1.3 Implementasi Fungsi Bgedge Comp

Subbab ini menjabarkan implementasi fungsi Bgedge Comp yang sudah dijelaskan pada subbab 3.3.1.3. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.8.

```
1.  bool bgedgeComp (const BGEDGE &bgea, const BGEDGE &bgeb
    ) {
2.      return (
3.          _bge_weight(bgea) != _bge_weight(bgeb) ? _bge_w
    eight(bgea) < _bge_weight(bgeb) :
4.              _bge_box_idx(bgea) != _bge_box_idx(bgeb) ?
    _bge_box_idx(bgea) < _bge_box_idx(bgeb) :
5.                  _bge_goal_idx(bgea) < _bge_goal_idx(bge
    a)
6.      );
7.  }
```

Kode Sumber 4.8 Implementasi Fungsi Bgedge Comp

4.3.2 Implementasi Struct Node Compare

Struct Node Compare adalah sebuah tipe data yang digunakan sebagai paramater pada *graph forward move*. Fungsi inilah yang mengatur agar *node* dengan nilai *heuristic* terkecil lalu *node* dengan nilai *push* terbesar akan ditaruh pada posisi teratas dalam *priority queue*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.9.

```
1.  bool NodeCompare::operator()(const NODE &na, const NODE
     &nb) {
2.      return _n_heuristic(na) == _n_heuristic(nb) ?
3.          _n_push(na) < _n_push(nb)
4.          : _n_heuristic(na) > _n_heuristic(nb);
5.  }
```

Kode Sumber 4.9 Implementasi Fungsi Operator Node Compare

4.4 Implementasi Mark State

Subbab ini menjabarkan implementasi fungsi Mark State yang sudah dijelaskan pada subbab 3.4. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.10.

```
1.  void markState(NODE &node, bool forward_check = true) {

2.      if (forward_check)
3.          forward_state_map[_n_boxes(node)][_n_player(nod
    e)] = true;
4.      else
5.          backward_state_map[_n_boxes(node)][_n_player(no
    de)] = true;
6.  }
```

Kode Sumber 4.10 Implementasi Fungsi Mark State

4.5 Implementasi *Path Graph*

Subbab ini menjabarkan implementasi semua fungsi yang membentuk *path graph* yang sudah dijelaskan pada subbab 3.5.

4.5.1 Implementasi Fungsi Get Path Map Key

Subbab ini menjabarkan implementasi fungsi Get Path Map Key yang sudah dijelaskan pada subbab 3.5.1. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.11.

```
1.  string getPathMapKey(ULL &boxes, int &player) {
2.      return to_string(boxes) + "-
    " + to_string(player);
3.  }
```

Kode Sumber 4.11 Implementasi Fungsi Get Path Map Key

4.5.2 Implementasi Fungsi Get Path Map Value

Subbab ini menjabarkan implementasi fungsi Get Path Map Value yang sudah dijelaskan pada subbab 3.5.2. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.12.

```
1.  string getPathMapValue(string &parent_key, string &path
    ) {
2.      return parent_key + ";" + path;
3.  }
```

Kode Sumber 4.12 Implementasi Fungsi Get Path Map Value

4.5.3 Implementasi Fungsi Mark Path

Subbab ini menjabarkan implementasi fungsi Mark Path yang sudah dijelaskan pada subbab 3.5.3. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.13.

```
1.  void markPath(string &key, string &parent_key, string &
    path, bool forward_check = true) {
2.      if (forward_check)
3.          forward_path_map[key] = getPathMapValue(parent_
    key, path);
4.      else
5.          backward_path_map[key] = getPathMapValue(parent
    _key, path);
6.  }
```

Kode Sumber 4.13 Implementasi Fungsi Mark Path

4.5.4 Implementasi Fungsi Split Path Map Value

Subbab ini menjabarkan implementasi fungsi Split Path Map Value yang sudah dijelaskan pada subbab 3.5.4. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.14.

```
1.  void splitPathMapValue(string &path_map_value, string &
    parent, string &path) {
2.      parent = "";
3.      path = "";
4.
5.      static bool parent_build;
6.      parent_build = true;
7.
8.      for (int i = 0, len = path_map_value.length(); i <
    len; i++) {
9.          static char c;
10.         c = path_map_value[i];
11.
12.         if (c == ';') {
13.             parent_build = false;
14.         } else {
15.             if (parent_build)
16.                 parent += c;
17.             else
18.                 path += c;
19.         }
20.     }
21. }
```

Kode Sumber 4.14 Implementasi Fungsi Split Path Map Value

4.5.5 Implementasi Fungsi Get Full Path

Subbab ini menjabarkan implementasi fungsi Get Full Path yang sudah dijelaskan pada subbab 3.5.5. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.15.

```
1.  string getFullPath(string &start_key, bool forward_chec
    k = true) {
2.      static string path_map_value;
3.      static string key;
4.      static string parent;
5.      static string path;
6.
7.      static string full_path;
8.      static string root_key;
9.
10.     full_path = "";
11.     root_key = forward_check ? forward_path_map_root :
    backward_path_map_root;
12.
13.     key = start_key;
14.     while (key != root_key) {
15.         path_map_value = forward_check ? forward_path_m
    ap[key] : backward_path_map[key];
16.         splitPathMapValue(path_map_value, parent, path)
    ;
17.
18.         if (forward_check)
19.             full_path = path + full_path;
20.         else
21.             full_path = full_path + path;
22.
23.         key = parent;
24.     }
25.
26.     return full_path;
27. }
```

Kode Sumber 4.15 Implementasi Fungsi Get Full Path

4.6 Implementasi Deteksi Deadlock

Subbab ini menjelaskan implementasi fungsi-fungsi yang digunakan untuk mendeteksi *deadlock*. Program yang terdapat pada subbab ini menggunakan desain yang dijelaskan pada pada bagian 3.8.

4.6.1 Implementasi Fungsi Is Simple Deadlock

Subbab ini menjabarkan implementasi fungsi Is Simple Deadlock yang sudah dijelaskan pada subbab 3.6.1. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.16.

```
1.  bool isSimpleDeadlock(ULL &boxes) {
2.      return ((locks & boxes) != 0ULL);
3.  }
```

Kode Sumber 4.16 Implementasi Fungsi Is Simple Deadlock

4.6.1.1 Implementasi Fungsi Generate Deadlock

Subbab ini menjabarkan implementasi fungsi Generate Deadlock yang sudah dijelaskan pada subbab 3.6.1.1. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.17.

```
1.  void generateDeadlock() {
2.      vector<int> forward_goal_list;
3.      convertToVector(forward_goals, forward_goal_list);
4.
5.      bool visited[ULL_BIT_SIZE];
6.      memset(visited, false, sizeof(visited));
7.
8.      ULL safe_indices = 0ULL;
9.      for (int goal_index: forward_goal_list)
10.         expandSafeIndices(safe_indices, goal_index, vis
    ited);
11.
12.     // Reversed safe indices bits as locks value
13.     locks = ~safe_indices;
14. }
```

Kode Sumber 4.17 Implementasi Fungsi Generate Deadlock

4.6.1.2 Implementasi Fungsi Expand Safe Indices

Subbab ini menjabarkan implementasi fungsi Expand Safe Indices yang sudah dijelaskan pada subbab 3.6.1.2. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.18.

```
1.  void expandSafeIndices(ULL &safe_indices, int index,
    bool visited[]) {
2.      if (!isValidIndex(index) || visited[index])
3.          return;
4.
5.      visited[index] = true;
6.      safe_indices |= mask(index);
7.
8.      static int f1_side; // One index ahead
9.      static int f2_side; // Two index ahead
10.
11.     static int directions[_N_DIR] = {U_DIR, R_DIR, D_DI
    R, L_DIR};
12.     for (int DIR: directions) {
13.         f1_side = getIndexMoved(index, DIR);
14.         f2_side = getIndexMoved(f1_side, DIR);
15.
16.         if (isValidIndex(f1_side) && !isWall(f1_side))

17.             if (isValidIndex(f2_side) && !isWall(f2_sid
    e))
18.                 expandSafeIndices(safe_indices, f1_side
    , visited);
19.     }
20. }
```

Kode Sumber 4.18 Implementasi Fungsi Expand Safe Indices

4.6.2 Implementasi Fungsi Is Freeze Deadlock

Subbab ini menjabarkan implementasi fungsi Is Freeze Deadlock yang sudah dijelaskan pada subbab 3.6.1.2. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.19.

```
1.  bool isFreezeDeadlock(ULL &boxes) {
2.      static vector<int> box_list;
3.      convertToVector(boxes, box_list);
4.
5.      static bool deadlock;
6.      deadlock = false;
7.
8.      for (int i = 0; !deadlock && i < n_attr; i++) {
9.          // Set index as center of winds
10.         assignWindsDirection(box_list[i]);
11.
12.         deadlock = false
13.                    || isFDTwoBoxes1(boxes, box_list[i])
14.                    || isFDTwoBoxes2(boxes, box_list[i])
15.                    || isFDThreeBoxes1(boxes, box_list[i
    ])
16.                    || isFDThreeBoxes2(boxes, box_list[i
    ])
17.                    || isFDFourBoxes(boxes, box_list[i])
    ;
18.     }
19.
20.     return deadlock;
21. }
```

Kode Sumber 4.19 Implementasi Fungsi Is Freeze Deadlock

4.7 Implementasi Fungsi Validasi

Subbab ini menjelaskan implementasi fungsi-fungsi yang digunakan untuk memvalidasi sebuah kondisi. Program yang terdapat pada subbab ini menggunakan desain yang dijelaskan pada pada bagian 3.7.

4.7.1 Implementasi Fungsi Is Box

Subbab ini menjabarkan implementasi fungsi Is Box yang sudah dijelaskan pada Tabel 3.1. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.20.

```
1.  bool isBox(ULL &boxes, int &index) {
2.      return ((mask(index) & boxes) != 0ULL);
3.  }
```

Kode Sumber 4.20 Implementasi Fungsi Is Box

4.7.2 Implementasi Fungsi Is Deadlock

Subbab ini menjabarkan implementasi fungsi Is Deadlock yang sudah dijelaskan pada Tabel 3.1. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.21.

```
1.  bool isDeadlock(ULL &boxes) {
2.      return isSimpleDeadlock(boxes) || isFreezeDeadlock(
    boxes);
3.  }
```

Kode Sumber 4.21 Implementasi Fungsi Is Deadlock

4.7.3 Implementasi Fungsi Is Goal

Subbab ini menjabarkan implementasi fungsi Is Goal yang sudah dijelaskan pada Tabel 3.1. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.22.

```
1.  bool isGoal(int &index, bool forward_check = true) {
2.      return forward_check ?
3.          ((mask(index) & forward_goals) != 0ULL)
4.          : ((mask(index) & backward_goals) != 0ULL);
5.  }
```

Kode Sumber 4.22 Implementasi Fungsi Is Goal

4.7.4 Implementasi Fungsi Is Goals

Subbab ini menjabarkan implementasi fungsi Is Goals yang sudah dijelaskan pada Tabel 3.1. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.23.

```
1.  bool isGoals(ULL &goals, bool forward_check = true) {
2.      return forward_check ?
3.          ((goals & forward_goals) == goals)
4.          : ((goals & backward_goals) == goals);
5.  }
```

Kode Sumber 4.23 Implementasi Fungsi Is Goals

4.7.5 Implementasi Fungsi Is Game Over

Subbab ini menjabarkan implementasi fungsi Is Game Over yang sudah dijelaskan pada Tabel 3.1. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.24.

```
1.  bool isGameOver(NODE &node, bool forward_check = true)
    {
2.      return forward_check ?
3.          ((_n_boxes(node) ^ forward_goals) == 0ULL)
4.          : ((_n_boxes(node) ^ backward_goals) == 0ULL);

5.  }
```

Kode Sumber 4.24 Implementasi Fungsi Is Game Over

4.7.6 Implementasi Fungsi Is Game Over Without Move

Subbab ini menjabarkan implementasi fungsi Is Game Over Without Move yang sudah dijelaskan pada Tabel 3.1. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.25.

```
1.  bool isGameOverWithoutMove() {
2.      return ((initial_forward_boxes ^ forward_goals) ==
    0ULL);
3.  }
```

Kode Sumber 4.25 Implementasi Fungsi Is Game Over Without Move

4.7.7 Implementasi Fungsi Is Lock

Subbab ini menjabarkan implementasi fungsi Is Lock yang sudah dijelaskan pada Tabel 3.1. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.26.

```
1.  bool isLock(int &index) {
2.      return ((mask(index) & locks) != 0ULL);
3.  }
```

Kode Sumber 4.26 Implementasi Fungsi Is Lock

4.7.8 Implementasi Fungsi Is Obstacle

Subbab ini menjabarkan implementasi fungsi Is Obstacle yang sudah dijelaskan pada Tabel 3.1. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.27.

```
1.  bool isObstacle(ULL &boxes, int &index) {
2.      return (isBox(boxes, index) || isWall(index));
3.  }
```

Kode Sumber 4.27 Implementasi Fungsi Is Obstacle

4.7.9 Implementasi Fungsi Is On Same Column

Subbab ini menjabarkan implementasi fungsi Is On Same Column yang sudah dijelaskan pada Tabel 3.1. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.28.

```
1.  bool isOnSameColumn(int &index1, int &index2) {
2.      return ((index1 % n_col) == (index2 % n_col));
3.  }
```

Kode Sumber 4.28 Implementasi Fungsi Is On Same Column

4.7.10 Implementasi Fungsi Is On Same Row

Subbab ini menjabarkan implementasi fungsi Is On Same Row yang sudah dijelaskan pada Tabel 3.1. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.29.

```
1.  bool isOnSameRow(int &index1, int &index2) {
2.      return ((index1 / n_col) == (index2 / n_col));
3.  }
```

Kode Sumber 4.29 Implementasi Fungsi Is On Same Row

4.7.11 Implementasi Fungsi Is State Seen

Subbab ini menjabarkan implementasi fungsi Is State Seen yang sudah dijelaskan pada Tabel 3.1. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.30.

```
1.  bool isStateSeen(ULL &boxes, int &player, bool forward_
    check = true) {
2.      return forward_check ?
3.          forward_state_map[boxes].find(player) != forwar
    d_state_map[boxes].end()
4.          : backward_state_map[boxes].find(player) != bac
    kward_state_map[boxes].end();
5.  }
```

Kode Sumber 4.30 Implementasi Fungsi Is State Seen

4.7.12 Implementasi Fungsi Is Tunnel

Subbab ini menjabarkan implementasi fungsi Is Tunnel yang sudah dijelaskan pada Tabel 3.1. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.31.

```
1.  bool isTunnel(int &index) {
2.      return ((mask(index) & tunnels) != 0ULL);
3.  }
```

Kode Sumber 4.31 Implementasi Fungsi Is Tunnel

4.7.13 Implementasi Fungsi Is Valid Index

Subbab ini menjabarkan implementasi fungsi Is Valid Index yang sudah dijelaskan pada Tabel 3.1. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.32.

```
1.  bool isValidIndex(int &index) {
2.      return (index >= 0 && index < max_index);
3.  }
```

Kode Sumber 4.32 Implementasi Fungsi Is Valid Index

4.7.14 Implementasi Fungsi Is Valid Index Move

Subbab ini menjabarkan implementasi fungsi Is Valid Index Move yang sudah dijelaskan pada Tabel 3.1. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.33.

```
1.  bool isValidIndexMove(int &curr_index, int &next_index,
     int direction) {
2.      switch(direction) {
3.          case U_DIR:
4.              return (next_index >= 0);
5.          case R_DIR:
6.              return ((next_index < max_index) && isOnSam
    eRow(curr_index, next_index));
7.          case D_DIR:
8.              return (next_index < max_index);
9.          case L_DIR:
10.             return ((next_index >= 0) && isOnSameRow(cu
    rr_index, next_index));
11.         default:
12.             return true;
13.     }
14. }
```

Kode Sumber 4.33 Implementasi Fungsi Is Valid Index Move

4.7.15 Implementasi Fungsi Is Wall

Subbab ini menjabarkan implementasi fungsi Is Wall yang sudah dijelaskan pada Tabel 3.1. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.34.

```
1.  bool isWall(int &index) {
2.      return ((mask(index) & walls) != 0ULL);
3.  }
```

Kode Sumber 4.34 Implementasi Fungsi Is Wall

4.8 Implementasi Fungsi Global

Subbab ini menjelaskan implementasi fungsi-fungsi yang digunakan secara global dalam pencarian solusi. Program yang terdapat pada subbab ini menggunakan desain yang dijelaskan pada pada bagian 3.8.

4.8.1 Implementasi Fungsi Clear Template

Subbab ini menjabarkan implementasi fungsi Clear Template yang sudah dijelaskan pada Tabel 3.2. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.35.

```
1.  template <class C> void clearTemplate(C &c) {
2.      c = C();
3.  }
```

Kode Sumber 4.35 Implementasi Fungsi Clear Template

4.8.2 Implementasi Fungsi Memset Template

Subbab ini menjabarkan implementasi fungsi Memset Template yang sudah dijelaskan pada Tabel 3.2. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.36.

```
1.  template <typename T> void memsetTemplate(T arr[], T va
    lue, int limit) {
2.      for (int i = 0; i < limit; i++)
3.          arr[i] = value;
4.  }
```

Kode Sumber 4.36 Implementasi Fungsi Memset Template

4.8.3 Implementasi Fungsi Assign To Vector

Subbab ini menjabarkan implementasi fungsi Assign To Vector yang sudah dijelaskan pada Tabel 3.2. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.37.

```
1.  template <typename T> void assignToVector(T arr[], int
    start, int size_, vector<T> &v) {
2.      clearTemplate(v);
3.
4.      for (int i = 0; i < size_; i++)
5.          v.push_back(arr[start + i]);
6.  }
```

Kode Sumber 4.37 Implementasi Fungsi Assign To Vector

4.8.4 Implementasi Fungsi Get Index Moved

Subbab ini menjabarkan implementasi fungsi Get Index Moved yang sudah dijelaskan pada Tabel 3.2. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.38.

```
1.  int getIndexMoved(int &index, int direction) {
2.      switch(direction) {
3.          case U_DIR:
4.              return index - n_col;
5.          case R_DIR:
6.              return index + 1;
7.          case D_DIR:
8.              return index + n_col;
9.          case L_DIR:
10.             return index - 1;
11.         default:
12.             return index;
13.     }
14. }
```

Kode Sumber 4.38 Implementasi Fungsi Get Index Moved

4.8.5 Implementasi Fungsi Get New Node

Subbab ini menjabarkan implementasi fungsi Get New Node yang sudah dijelaskan pada Tabel 3.2. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.39.

```
1.  NODE getNewNode(int &heuristic, int &push, ULL &boxes,
    int &player) {
2.      return make_pair(
3.          make_pair(heuristic, push),
4.          make_pair(boxes, player)
5.      );
6.  }
```

Kode Sumber 4.39 Implementasi Fungsi Get New Node

4.8.6 Implementasi Fungsi Get Shortest Path

Subbab ini menjabarkan implementasi fungsi Get Shortest Path yang sudah dijelaskan pada Tabel 3.2. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.40.

```
1.  string getShortestPath(ULL &boxes, int &from, int &dest
    ) {
2.      static bool visited[ULL_BIT_SIZE];
3.      memsetTemplate(visited, false, max_index);
4.
5.      static pair<int,string> initial_visit;
6.      static list<pair<int,string> > visits;
7.
8.      initial_visit = make_pair(from, "");
9.      clearTemplate(visits);
10.
11.     visits.push_back(initial_visit);
12.     while (!visits.empty()) {
13.         static pair<int,string> visit;
14.         visit = visits.front();
15.
16.         #define visit_index visit.first
17.         #define visit_path visit.second
18.
19.         if (visit_index == dest) {
20.             return visit_path;
21.         } else {
22.             static int D_index;
23.             static int L_index;
24.             static int U_index;
25.             static int R_index;
26.
27.             D_index = getIndexMoved(visit_index, D_DIR)
    ;
28.             if (!visited[D_index] && !isObstacle(boxes,
     D_index))
29.                 visits.push_back(make_pair(D_index, vis
    it_path + "d"));
30.
31.             L_index = getIndexMoved(visit_index, L_DIR)
    ;
32.             if (!visited[L_index] && !isObstacle(boxes,
     L_index))
33.                 visits.push_back(make_pair(L_index, vis
    it_path + "l"));
```

Kode Sumber 4.40a Implementasi Fungsi Get Shortest Path

```
34.
35.             U_index = getIndexMoved(visit_index, U_DIR)
    ;
36.             if (!visited[U_index] && !isObstacle(boxes,
     U_index))
37.                 visits.push_back(make_pair(U_index, vis
    it_path + "u"));
38.
39.             R_index = getIndexMoved(visit_index, R_DIR)
    ;
40.             if (!visited[R_index] && !isObstacle(boxes,
     R_index))
41.                 visits.push_back(make_pair(R_index, vis
    it_path + "r"));
42.         }
43.
44.         visited[visit_index] = true;
45.         visits.pop_front();
46.
47.         #undef visit_index
48.         #undef visit_path
49.     }
50.
51.     return "-";
52. }
```

Kode Sumber 4.40b Implementasi Fungsi Get Shortest Path

4.8.7 Implementasi Fungsi Get Boxes Moved

Subbab ini menjabarkan implementasi fungsi Get Boxes Moved yang sudah dijelaskan pada Tabel 3.2. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.41.

```
1.  ULL getBoxesMoved(ULL &boxes, int &index, int direction
    ) {
2.      static ULL delete_box_index;
3.      static ULL put_box_index;
4.
5.      delete_box_index = ULLONG_MAX ^ mask(index);
6.      put_box_index = mask(getIndexMoved(index, direction
    ));
7.
8.      return ((boxes & delete_box_index) | put_box_index)
    ;
9.  }
```

Kode Sumber 4.41 Implementasi Fungsi Get Boxes Moved

4.8.8 Implementasi Fungsi Assign Winds Direction

Subbab ini menjabarkan implementasi fungsi Assign Winds Direction yang sudah dijelaskan pada Tabel 3.2. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.42.

```
1.  void assignWindsDirection(int &index) {
2.      N_index = getIndexMoved(index, U_DIR);
3.      E_index = getIndexMoved(index, R_DIR);
4.      S_index = getIndexMoved(index, D_DIR);
5.      W_index = getIndexMoved(index, L_DIR);
6.
7.      NE_index = getIndexMoved(N_index, R_DIR);
8.      NW_index = getIndexMoved(N_index, L_DIR);
9.      SE_index = getIndexMoved(S_index, R_DIR);
10.     SW_index = getIndexMoved(S_index, L_DIR);
11. }
```

Kode Sumber 4.42 Implementasi Fungsi Assign Winds Direction

4.8.9 Implementasi Fungsi Convert To Vector

Subbab ini menjabarkan implementasi fungsi Convert To Vector yang sudah dijelaskan pada Tabel 3.2. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.43.

```
1.  void convertToVector (ULL &bit_rep, vector<int> &indice
    s) {
2.      clearTemplate(indices);
3.
4.      for (int i = 0; i < max_index; i++) {
5.          if ((mask(i) & bit_rep) != 0ULL)
6.              indices.push_back(i);
7.      }
8.  }
```

Kode Sumber 4.43 Implementasi Fungsi Convert To Vector

4.8.10 Implementasi Fungsi Generate Successors

Subbab ini menjabarkan implementasi fungsi Generate Successors yang sudah dijelaskan pada Tabel 3.2. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.44.

```
1.  void generateSuccessors(NODE &node, vector<NODE> &succe
    ssors, bool forward_check = true) {
2.      clearTemplate(successors);
3.
4.      static vector<int> box_list;
5.      convertToVector(_n_boxes(node), box_list);
6.
7.      for (int i = 0; i < n_attr; i++) {
8.          if (forward_check)
9.              generateForwardBoxSuccessors(box_list[i], n
    ode, successors);
10.         else
11.             generateBackwardBoxSuccessors(box_list[i],
    node, successors);
12.     }
13. }
```

Kode Sumber 4.44 Implementasi Fungsi Generate Successors

4.9 Implementasi Fungsi Run Input

Subbab ini menjabarkan implementasi fungsi Run Input yang sudah dijelaskan pada bagian 3.10. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.45.

```
1.  void runInput() {
2.      string s;
3.      vector<string> inp;
4.
5.      clearTemplate(inp);
6.      while (getline(cin, s)) {
7.          inp.push_back(s);
8.
9.          n_row++;
10.         n_col = max(n_col, (int)s.length());
11.     }
12.
13.     initPuzzle(inp);
14. }
```

Kode Sumber 4.45 Implementasi Fungsi Run Input

4.10 Implementasi Fungsi Reinit Walls

Subbab ini menjabarkan implementasi fungsi Reinit Walls yang sudah dijelaskan pada bagian 3.13. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.46.

```
1.  void reinitWalls() {
2.      ULL blocked_area = ULLONG_MAX;
3.      bool visited[ULL_BIT_SIZE];
4.      memsetTemplate(visited, false, ULL_BIT_SIZE);
5.
6.      list<int> tracks;
7.      tracks.push_back(initial_forward_player);
8.
9.      while (!tracks.empty()) {
10.         static int index;
11.         index = tracks.front();
12.
13.         if (!visited[index] && !isWall(index)) {
14.             blocked_area ^= mask(index);
15.
16.             tracks.push_back(getIndexMoved(index, U_DIR
    ));
17.             tracks.push_back(getIndexMoved(index, R_DIR
    ));
18.             tracks.push_back(getIndexMoved(index, D_DIR
    ));
19.             tracks.push_back(getIndexMoved(index, L_DIR
    ));
20.
21.             visited[index] = true;
22.         }
23.
24.         tracks.pop_front();
25.     }
26.
27.     // Set all outside play area indices as walls
28.     walls |= blocked_area;
29. }
```

Kode Sumber 4.46 Implementasi Fungsi Reinit Walls

4.11 Implementasi Fungsi Generate Tunnels

Subbab ini menjabarkan implementasi fungsi Generate Tunnels yang sudah dijelaskan pada bagian 3.14. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.47.

```
1.  void generateTunnels() {
2.      for (int index = 0; index < max_index; index++) {
3.          if (!isWall(index)) {
4.              assignWindsDirection(index);
5.
6.              if (isHorizontalTunnel() || isVerticalTunne
    l())
7.                  tunnels |= mask(index);
8.          }
9.      }
10. }
```

Kode Sumber 4.47 Implementasi Fungsi Generate Tunnels

4.11.1 Implementasi Fungsi Is Horizontal Tunnel

Subbab ini menjabarkan implementasi fungsi Is Horizontal Tunnel yang sudah dijelaskan pada bagian 3.14.1. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.48.

```
1.  bool isHorizontalTunnel() {
2.      return (
3.          isWall(NW_index) && isWall(N_index) && isWall(N
    E_index) &&
4.          isWall(SW_index) && isWall(S_index) && isWall(S
    E_index)
5.      );
6.  }
```

Kode Sumber 4.48 Implementasi Fungsi Is Horizontal Tunnel

4.11.2 Implementasi Fungsi Is Vertical Tunnel

Subbab ini menjabarkan implementasi fungsi Is Vertical Tunnel yang sudah dijelaskan pada bagian 3.14.2. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.49.

```
1.  bool isVerticalTunnel() {
2.      return (
3.          isWall(NW_index) && isWall(W_index) && isWall(S
    W_index) &&
4.          isWall(NE_index) && isWall(E_index) && isWall(S
    E_index)
5.      );
6.  }
```

Kode Sumber 4.49 Implementasi Fungsi Is Vertical Tunnel

4.12 Implementasi Tabel Jarak

Subbab ini menjelaskan implementasi fungsi-fungsi yang digunakan untuk mengkonstruksi tabel jarak antar indeks. Program yang terdapat pada subbab ini menggunakan desain yang dijelaskan pada pada bagian 3.15.

4.12.1 Implementasi Fungsi Init Table

Subbab ini menjabarkan implementasi fungsi Init Table yang sudah dijelaskan pada bagian 3.15.1. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.50.

```
1.  void initTable() {
2.      // Initial value
3.      table = vector<vector<int> >(max_index, vector<int>
    (max_index, oo));
4.
5.      // Value zero for same index
6.      for (int i = 0; i < max_index; i++)
7.          table[i][i] = 0;
8.
9.      // Change initial table based on its adjacent
10.     for (int i = 0; i < max_index; i++) {
11.         static int U_index;
12.         static int R_index;
13.         static int D_index;
14.         static int L_index;
15.
16.         if (!isWall(i)) {
17.             U_index = getIndexMoved(i, U_DIR);
18.             if (isValidIndexMove(i, U_index, U_DIR) &&
    !isWall(U_index))
19.                 table[i][U_index] = 1;
20.
21.             R_index = getIndexMoved(i, R_DIR);
22.             if (isValidIndexMove(i, R_index, R_DIR) &&
    !isWall(R_index))
23.                 table[i][R_index] = 1;
24.
25.             D_index = getIndexMoved(i, D_DIR);
26.             if (isValidIndexMove(i, D_index, D_DIR) &&
    !isWall(D_index))
27.                 table[i][D_index] = 1;
28.
29.             L_index = getIndexMoved(i, L_DIR);
30.             if (isValidIndexMove(i, L_index, L_DIR) &&
    !isWall(L_index))
31.                 table[i][L_index] = 1;
32.         }
33.     }
34. }
```

Kode Sumber 4.50 Implementasi Fungsi Init Table

4.12.2 Implementasi Fungsi Create Distance Table

Subbab ini menjabarkan implementasi fungsi Create Distance Table yang sudah dijelaskan pada bagian 3.15.2. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.51.

```
1.  void createDistanceTable() {
2.      initTable();
3.
4.      for (int i = 0; i < max_index; i++)
5.          updateShortestPath(i);
6.  }
```

Kode Sumber 4.51 Implementasi Fungsi Create Distance Table

4.13 Implementasi Fungsi Solve

Subbab ini menjabarkan implementasi fungsi Solve sebagai fungsi utama untuk mencari solusi dari *puzzle*. Alur dari jalannya fungsi sudah dijelaskan pada Gambar 3.7. Implementasi fungsi Solve dapat dilihat pada Kode Sumber 4.52.

```
1.  void solve() {
2.      initGlobalVariables();
3.
4.      if (isGameOverWithoutMove()) {
5.          cout << endl;
6.      } else {
7.          addInitialNode();
8.
9.          bool win = false;
10.         while (!win) {
11.             assert(!(forward_tracks.empty() && backward
    _tracks.empty()));
12.
13.             runForwardMove(win);
14.             if (!win)
15.                 runBackwardMove(win);
16.         }
17.     }
18. }
```

Kode Sumber 4.52 Implementasi Fungsi Solve

4.13.1 Implementasi Fungsi Init Global Variables

Subbab ini menjabarkan implementasi fungsi Init Global Variables sebagai fungsi untuk menginisialisasi nilai variabel global yang nilainya tergantung dari *puzzle*. Desain dari fungsi telah diberikan pada bagian 3.12. Implementasi fungsi Solve dapat dilihat pada Kode Sumber 4.53.

```
1.  void initGlobalVariables() {
2.      for (int i = 0; i < puzzle.length(); i++) {
3.          char c = puzzle[i];
4.
5.          switch(c) {
6.              case '#':
7.                  walls = walls | mask(i);
8.                  break;
9.              case '.':
10.                 forward_goals = forward_goals | mask(i)
    ;
11.                 forward_goal_list.push_back(i);
12.                 break;
13.             case '@':
14.                 initial_forward_player = i;
15.                 break;
16.             case '+':
17.                 initial_forward_player = i;
18.                 forward_goals = forward_goals | mask(i)
    ;
19.                 forward_goal_list.push_back(i);
20.                 break;
21.             case '$':
22.                 initial_forward_boxes = initial_forward
    _boxes | mask(i);
23.                 break;
24.             case '*':
25.                 initial_forward_boxes = initial_forward
    _boxes | mask(i);
26.                 forward_goals = forward_goals | mask(i)
    ;
27.                 forward_goal_list.push_back(i);
28.                 break;
29.             default:
30.                 break;
31.         }
32.     }
```

Kode Sumber 4.53b Implementasi Fungsi Init Global Variables

```
33.     n_attr = forward_goal_list.size();
34.     max_index = n_col * n_row;
35.     reinitWalls();
36.
37.     generateDeadlock();
38.     generateTunnels();
39.     createDistanceTable();
40.
41.     initForwardGlobalVariables();
42.     initBackwardGlobalVariables();
43. }
```

Kode Sumber 4.53b Implementasi Fungsi Init Global Variables

4.13.1.1 Implementasi Fungsi Init Forward Global Variables

Subbab ini menjabarkan implementasi fungsi Init Forward Global Variables yang sudah dijelaskan pada subbab 3.16. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.54.

```
1.  void initForwardGlobalVariables() {
2.      forward_path_map_root = getPathMapKey(initial_forwa
    rd_boxes, initial_forward_player);
3.
4.      clearTemplate(forward_path_map);
5.      clearTemplate(forward_tracks);
6.  }
```

Kode Sumber 4.54 Implementasi Fungsi Init Forward Global Variables

4.13.1.2 Implementasi Fungsi Init Backward Global Variables

Subbab ini menjabarkan implementasi fungsi Init Backward Global Variables yang sudah dijelaskan pada subbab 3.17. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.55.

```
1.  void initBackwardGlobalVariables() {
2.      int initial_backward_player = 0;
3.
4.      backward_goals = initial_forward_boxes;
5.      initial_backward_boxes = forward_goals;
6.      backward_path_map_root = getPathMapKey(initial_back
    ward_boxes, initial_backward_player);
7.
8.      clearTemplate(backward_path_map);
9.      clearTemplate(backward_tracks);
10.
11.     convertToVector(backward_goals, backward_goal_list)
    ;
12. }
```

Kode Sumber 4.55 Implementasi Fungsi Init Backwars Global Variables

4.13.2 Implementasi Fungsi Add Initial Node

Subbab ini menjabarkan implementasi fungsi Add Initial Node yang sudah dijelaskan pada subbab 3.18. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.56.

```
1.  void addInitialNode() {
2.      addForwardInitialNode();
3.      addBackwardInitialNode();
4.  }
```

Kode Sumber 4.56 Implementasi Fungsi Add Initial Node

4.13.2.1 Implementasi Fungsi Add Forward Initial Node

Subbab ini menjabarkan implementasi fungsi Add Forward Initial Node yang sudah dijelaskan pada subbab 3.18.1. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.57.

```
1.  void addForwardInitialNode() {
2.      int heuristic = 0;
3.      int push = 0;
4.      ULL boxes = initial_forward_boxes;
5.      int player = initial_forward_player;
6.
7.      NODE node = getNewNode(heuristic, push, boxes, play
    er);
8.      markState(node, true);
9.      forward_tracks.push(node);
10. }
```

Kode Sumber 4.57 Implementasi Fungsi Add Forward Initial Node

4.13.2.2 Implementasi Fungsi Add Backward Initial Node

Subbab ini menjabarkan implementasi fungsi Add Backward Initial Node yang sudah dijelaskan pada subbab 3.18.2. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.58.

```
1.  void addBackwardInitialNode() {
2.      vector<int> box_list;
3.      convertToVector(initial_backward_boxes, box_list);

4.
5.      int heuristic = 0;
6.      int push = 1;
7.      string key;
8.
9.      for (int i = 0; i < n_attr; i++) {
10.         static NODE node;
11.         static int box;
12.         box = box_list[i];
13.
14.         static string path;
15.         static ULL boxes;
16.         static int player;
17.
```

Kode Sumber 4.58a Implementasi Fungsi Add Backward Initial Node

```
18.         static int directions[_N_DIR] = {U_DIR, R_DIR,
    D_DIR, L_DIR};
19.         static char push_paths[_N_DIR] = {'D', 'L', 'U'
    , 'R'};
20.
21.         for (int i = 0; i < _N_DIR; i++) {
22.             static int dir;
23.             dir = directions[i];
24.
25.             static int f1_side;
26.             static int f2_side;
27.
28.             f1_side = getIndexMoved(box, dir);
29.             f2_side = getIndexMoved(f1_side, dir);
30.
31.             if (!isObstacle(initial_backward_boxes, f1_
    side)
32.                 && !isObstacle(initial_backward_boxes,
    f2_side)) {
33.                 path = push_paths[i];
34.                 boxes = getBoxesMoved(initial_backward_
    boxes, box, dir);
35.                 player = f2_side;
36.
37.                 node = getNewNode(heuristic, push, boxe
    s, player);
38.                 backward_tracks.push_back(node);
39.
40.                 // Mark Path
41.                 key = getPathMapKey(boxes, player);
42.                 markPath(key, backward_path_map_root, p
    ath, false);
43.                 markState(node, false);
44.             }
45.         }
46.     }
47. }
```

Kode Sumber 4.58b Implementasi Fungsi Add Backward Initial Node

4.13.3 Implementasi Fungsi Run Forward Move

Subbab ini menjabarkan implementasi fungsi Run Forward Move yang sudah dijelaskan pada subbab 3.19. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.59.

```
1.  void runForwardMove(bool &win) {
2.      if (!forward_tracks.empty()) {
3.          static NODE node;
4.          node = forward_tracks.top();
5.
6.          forward_tracks.pop();
7.
8.          static vector<NODE> successors;
9.          generateSuccessors(node, successors, true);
10.
11.         for (NODE successor: successors) {
12.             markState(successor, true);
13.
14.             if (isGameOver(successor , true)) {
15.                 static ULL boxes;
16.                 static int player;
17.
18.                 boxes = _n_boxes(successor);
19.                 player = _n_player(successor);
20.
21.                 string key = getPathMapKey(boxes, playe
    r);
22.                 string ans = getFullPath(key, true);
23.
24.                 cout << ans << endl;
25.
26.                 win = true;
27.                 break;
28.             }
29.
```

Kode Sumber 4.59a Implementasi Fungsi Run Forward Move

```
30.             checkMeetInTheMiddle(successor, win, true);
31.
32.             if (!win)
33.                 forward_tracks.push(successor);
34.             else
35.                 break;
36.         }
37.     }
38. }
```

Kode Sumber 4.59b Implementasi Fungsi Run Forward Move

4.13.3.1 Implementasi Fungsi Generate Forward Box Successors

Subbab ini menjabarkan implementasi fungsi Generate Forward Successors yang sudah dijelaskan pada subbab 3.19.1. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.60.

```
1.  void generateForwardBoxSuccessors(int &box, NODE &node,
     vector<NODE> &successors) {
2.      static int heuristic;
3.      static int push;
4.      static int player;
5.      static ULL boxes;
6.
7.      push = _n_push(node) + 1;
8.      if (push > MAX_PUSH_FORWARD)
9.          return;
10.
11.     static string key;
12.     static string parent_key;
13.     static string path;
14.
15.     parent_key = getPathMapKey(_n_boxes(node), _n_playe
    r(node));
16.
```

Kode Sumber 4.60a Implementasi Fungsi Generate Forward Box Successors

```
17.     static NODE node_new;
18.     static int box_pos_new;
19.     static bool valid;
20.
21.     static int directions[_N_DIR] = {D_DIR, L_DIR, U_DI
    R, R_DIR};
22.     static char push_paths[_N_DIR] = {'D', 'L', 'U', 'R
    '};
23.
24.     for (int i = 0; i < _N_DIR; i++) {
25.         static int f_dir; // Front side of direction
26.         static int b_dir; // Back side of direction
27.
28.         f_dir = directions[i];
29.         b_dir = directions[(i + 2) % _N_DIR];
30.
31.         static int f_side;
32.         static int b_side;
33.
34.         f_side = getIndexMoved(box, f_dir);
35.         b_side = getIndexMoved(box, b_dir);
36.
37.         // If both sides on one straight line to box ar
    en't obstacles \
38.            then box is possible to push
39.         if (!isObstacle(_n_boxes(node), f_side)
40.             && !isObstacle(_n_boxes(node), b_side)) {
41.             static string p_to_bs;
42.             p_to_bs = getShortestPath(_n_boxes(node), _
    n_player(node), b_side);
43.
44.             // If player can reach b_side then box can
    be pushed to f_side
45.             if (p_to_bs != "-") {
46.                 boxes = getBoxesMoved(_n_boxes(node), b
    ox, f_dir);
```

Kode Sumber 4.60b Implementasi Fungsi Generate Forward Box Successors

```
47.                 player = box;
48.
49.                 // If state never seen before then node
    _new could be valid
50.                 if (!isStateSeen(boxes, player, true))
    {
51.                     heuristic = heuristicValue(boxes, p
    layer, true);
52.                     path = p_to_bs + push_paths[i];
53.
54.                     // Make sure generated successor is
     valid
55.                     if (heuristic != oo) {
56.                         node_new = getNewNode(heuristic
    , push, boxes, player);
57.                         box_pos_new = getIndexMoved(box
    , f_dir);
58.                         valid = doMacroTunnel(node_new,
     box_pos_new, f_dir, path, true);
59.
60.                         // Make sure generated successo
    r is valid after macro tunnel move is done
61.                         if (valid && !isStateSeen(_n_bo
    xes(node_new), _n_player(node_new), true)) {
62.                             key = getPathMapKey(_n_boxe
    s(node_new), _n_player(node_new));
63.                             markPath(key, parent_key, p
    ath, true);
64.
65.                             successors.push_back(node_n
    ew);
66.                         }
67.                     }
68.                 }
69.             }
70.         }
71.     }
72. }
```

Kode Sumber 4.60c Implementasi Fungsi Generate Forward Box Successors

4.13.3.2 Implementasi Fungsi Check Forward Meet In The Middle untuk Forward Move

Subbab ini menjabarkan implementasi fungsi Check Forward Meet In The Middle yang sudah dijelaskan pada subbab 3.19.2. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.61.

```
1.  void checkMeetInTheMiddle(NODE &node, bool &win, bool f
    orward_check = true) {
2.      static map<int,bool> *temp;
3.
4.      if (forward_check) {
5.          if (backward_state_map.find(_n_boxes(node)) !=
    backward_state_map.end()) {
6.              temp = &backward_state_map[_n_boxes(node)];
7.
8.              for (map<int,bool>::iterator it = temp-
    >begin(); it != temp->end(); it++) {
9.                  static int backward_player;
10.                 static string middle_path;
11.
12.                 backward_player = it->first;
13.                 middle_path = getShortestPath(_n_boxes(
    node), _n_player(node), backward_player);
14.
15.                 if (middle_path != "-") {
16.                     string forward_key = getPathMapKey(
    _n_boxes(node), _n_player(node));
17.                     string forward_path = getFullPath(f
    orward_key, true);
18.
19.                     string backward_key = getPathMapKey
    (_n_boxes(node), backward_player);
20.                     string backward_path = getFullPath(
    backward_key, false);
21.
```

Kode Sumber 4.61a Implementasi Fungsi Check Meet In The Middle untuk *Forward Move*

```
                    string ans = forward_path + middle_
path + backward_path;
                    cout << ans << endl;

                    win = true;
                    break;
                }
            }
        } else {
            ...
        }
}
```

Kode Sumber 4.61b Implementasi Fungsi Check Meet In The Middle untuk *Forward Move*

4.13.4 Implementasi Fungsi Run Backward Move

Subbab ini menjabarkan implementasi fungsi Run Backward Move yang sudah dijelaskan pada subbab 3.20. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.62.

```
void runBackwardMove(bool &win) {
    if (!backward_tracks.empty()) {
        static NODE node;
        node = backward_tracks.front();

        backward_tracks.pop_front();

        static vector<NODE> successors;
        generateSuccessors(node, successors, false);

        for (NODE successor: successors) {
            markState(successor, false);

            if (isGameOver(successor , false)) {
                static ULL boxes;
                static int player;
```

Kode Sumber 4.62a Implementasi Fungsi Run Backward Move

```
17.                 static string prefix_path;
18.
19.                 boxes = _n_boxes(successor);
20.                 player = _n_player(successor);
21.                 prefix_path = getShortestPath(boxes, in
    itial_forward_player, player);
22.
23.                 if (prefix_path != "-") {
24.                     string key = getPathMapKey(boxes, p
    layer);
25.                     string ans = prefix_path + getFullP
    ath(key, false);
26.
27.                     cout << ans << endl;
28.
29.                     win = true;
30.                     break;
31.                 } else {
32.                     continue;
33.                 }
34.             }
35.
36.             checkMeetInTheMiddle(successor, win, false)
    ;
37.
38.             if (!win)
39.                 backward_tracks.push_back(successor);
40.             else
41.                 break;
42.         }
43.     }
44. }
```

Kode Sumber 4.62b Implementasi Fungsi Run Backward Move

4.13.4.1 Implementasi Fungsi Generate Backward Move Box Successors

Subbab ini menjabarkan implementasi fungsi Generate Backward Move Successors yang sudah dijelaskan pada subbab 3.20.1. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.63.

```
1.  void generateBackwardBoxSuccessors(int &box, NODE &node
    , vector<NODE> &successors) {
2.      static int heuristic = 0;
3.      static int push;
4.      static int player;
5.      static ULL boxes;
6.
7.      push = _n_push(node) + 1;
8.      if (push > MAX_PUSH_BACKWARD)
9.          return;
10.
11.     static string key;
12.     static string parent_key;
13.     static string path;
14.
15.     parent_key = getPathMapKey(_n_boxes(node), _n_playe
    r(node));
16.
17.     static NODE node_new;
18.     static int box_pos_new;
19.     static bool valid;
20.
21.     static int directions[_N_DIR] = {U_DIR, R_DIR, D_DI
    R, L_DIR};
22.     static char push_paths[_N_DIR] = {'D', 'L', 'U', 'R
    '};
23.
24.     for (int i = 0; i < _N_DIR; i++) {
25.         static int dir;
26.         dir = directions[i];
27.
28.         static int f1_side; // One index ahead
29.         static int f2_side; // Two index ahead
30.
31.         f1_side = getIndexMoved(box, dir);  // Player p
    osition after push action
32.         f2_side = getIndexMoved(f1_side, dir);
33.
```

Kode Sumber 4.63a Implementasi Fungsi Generate Backward Move Box Successors

```
34.         if (!isObstacle(_n_boxes(node), f1_side)
35.             && !isObstacle(_n_boxes(node), f2_side)) {

36.             boxes = getBoxesMoved(_n_boxes(node), box,
    dir);
37.             player = f2_side;   // Player position afte
    r pull up
38.
39.             if (!isStateSeen(boxes, player, false)) {
40.                 static string f1s_to_p; // Path from pl
    ayer position after push \
41.                                                 to next pre-
    push player position
42.                 f1s_to_p =  getShortestPath(_n_boxes(no
    de), f1_side, _n_player(node));
43.
44.                 // If player position after push can re
    ach next pre-push player position
45.                 if (f1s_to_p != "-") {
46.                     path = push_paths[i] + f1s_to_p;
47.                     node_new = getNewNode(heuristic, pu
    sh, boxes, player);
48.                     box_pos_new = getIndexMoved(box, di
    r);
49.                     valid = doMacroTunnel(node_new, box
    _pos_new, dir, path, false);
50.
51.                     if (valid && !isStateSeen(_n_boxes(
    node_new), _n_player(node_new), false)) {
52.                         key = getPathMapKey(_n_boxes(no
    de_new), _n_player(node_new));
53.                         markPath(key, parent_key, path,
     false);
54.
55.                         successors.push_back(node_new);

56.                     }
57.                 }
```

Kode Sumber 4.63b Implementasi Fungsi Generate Backward Move Box Successors

```
            }
        }
    }
}
```

Kode Sumber 4.63c Implementasi Fungsi Generate Backward Move Box Successors

4.13.4.2 Implementasi Fungsi Check Forward Meet In The Middle untuk Backward Move

Subbab ini menjabarkan implementasi fungsi Check Backward Meet In The Middle yang sudah dijelaskan pada subbab 3.20.2. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.64.

```
void checkMeetInTheMiddle(NODE &node, bool &win, bool forward_check = true) {
    static map<int,bool> *temp;

    if (forward_check) {
        ...
    } else {
        if (forward_state_map.find(_n_boxes(node)) != forward_state_map.end()) {
            temp = &forward_state_map[_n_boxes(node)];

            for (map<int,bool>::iterator it = temp->begin(); it != temp->end(); it++) {
                static int forward_player;
                static string middle_path;

                forward_player = it->first;
                middle_path = getShortestPath(_n_boxes(node), forward_player, _n_player(node));

```

Kode Sumber 4.64a Implementasi Fungsi Check Meet In The Middle untuk *Backward Move*

```
17.                 if (middle_path != "-") {
18.                     string forward_key = getPathMapKey(
    _n_boxes(node), forward_player);
19.                     string forward_path = getFullPath(f
    orward_key, true);
20.
21.                     string backward_key = getPathMapKey
    (_n_boxes(node), _n_player(node));
22.                     string backward_path = getFullPath(
    backward_key, false);
23.
24.                     string ans = forward_path + middle_
    path + backward_path;
25.                     cout << ans << endl;
26.
27.                     win = true;
28.                     break;
29.                 }
30.             }
31.         }
32.     }
33. }
```

Kode Sumber 4.64b Implementasi Fungsi Check Meet In The Middle untuk *Backward Move*

4.14 Implementasi Solusi Alternatif

Subbab ini menjelaskan implementasi dari modifikasi fungsi yang digunakan oleh sistem untuk mendapatkan sebuah sistem baru dengan bentuk solusi yang berbeda. Program yang terdapat pada subbab ini menggunakan desain yang dijelaskan pada pada bagian 3.22.

4.14.1 Implementasi Bidirectional Search DFS dan BFS

Subbab ini menjabarkan implementasi modifikasi dari solusi utama untuk mendapatkan solusi berupa *bidirectional search* DFS dan BFS yang sudah dijelaskan pada subbab 3.21.1. Variabel untuk *graph* pencarian untuk *forward move* solusi ini dapat dilihat pada Kode Sumber 4.65.

```
1.  stack<NODE> forward_tracks;
```

Kode Sumber 4.65 Variabel Graph Pencarian *Forward Move* pada Solusi *Bidirectional Search* DFS dan BFS

Meskipun tidak membutuhkan nilai *heuristic*, atribut *heusritic* pada *node* tetap diperlukan sebagai indikator dalam mendeteksi *deadlock*. Untuk itu, fungsi yang dimodifikasi pada solusi ini adalah fungsi Heuristic Value. Implementasi dapat dilihat pada Kode Sumber 4.66.

```
1.  int heuristicValue(ULL &boxes, int &player, bool forwar
    d_check = true) {
2.      if (isDeadlock(boxes))
3.          return oo;
4.
5.      return 0;
6.  }
```

Kode Sumber 4.66 Implementasi Modifikasi Fungsi Heuristic Value pada Solusi *Bidirectional Search* DFS dan BFS

4.14.2 Implementasi Bidirectional Search Greedy DFS dan BFS

Subbab ini menjabarkan implementasi modifikasi dari solusi utama untuk mendapatkan solusi berupa *bidirectional search greedy* DFS dan BFS yang sudah dijelaskan pada subbab 3.21.2. Bentuk variabel dari *graph* pencarian pada solusi ini sama dengan solusi pada subbab sebelumnya. Perbedaan dari solusi utama adalah adalah adanya penambahan sebaris kode pada fungsi Run Forward Move yang digunakan untuk mengurutkan semua *successor* yang di-*generate* pada tiap *state*. Implementasi fungsi dapat dilihat pada Kode Sumber 4.67.

```
1.  void runForwardMove(bool &win) {
2.      if (!forward_tracks.empty()) {
3.          ...
4.          static vector<NODE> successors;
5.          generateSuccessors(node, successors, true);
6.
7.          // Additional line
8.          sort(successors.begin(), successors.end(), succ
    essorComp);
9.          ...
10.     }
11. }
```

Kode Sumber 4.67 Implementasi Modifikasi Fungsi Run Forward Move pada Solusi *Bidirectional Search Greedy* DFS dan BFS

Pada solusi ini juga terdapat sebuah tambahan berupa fungsi yang bernama Successor Comp, yang digunakan untuk mengurutkan semua *successor* sebelum dimasukkan ke dalam *stack graph* pencarian. Implementasi fungsi dapat dilihat pada Kode Sumber 4.68.

```
1.  bool successorComp(const NODE &na, const NODE &nb) {
2.      return _n_heuristic(na) == _n_heuristic(nb) ?
3.          _n_push(na) < _n_push(nb)
4.          : _n_heuristic(na) > _n_heuristic(nb);
5.  }
```

Kode Sumber 4.68 Implementasi Fungsi Successor Comp pada Solusi *Bidirectional Search Greedy* DFS dan BFS

(Halaman ini sengaja dikosongkan)

BAB V
UJI COBA DAN EVALUASI

Bab ini akan membahas mengenai hasil uji coba sistem yang telah dirancang dan dibuat. Uji coba dilakukan untuk mengetahui kinerja sistem dengan lingkungan uji coba yang telah ditentukan.

5.1 Lingkungan Uji Coba

Uji coba program dilakukan pada perangkat dengan spesifikasi berikut.

1. Perangkat Keras
    a. *Processor* Intel Core i7-7700HQ CPU @ 2.80GHz (8 CPUs), ~2.8GHz
    b. *Random Access Memory* 8GB
2. Perangkat Lunak
    a. Sistem Operasi Windows 10 Pro 64-bit
    b. DevC++
    c. C++ 4.9.2
    d. Python 3.6

5.2 Uji Coba Kebenaran

Subbab ini menjelaskan pengujian program C++ untuk penyelesaian permasalahan Timus Online Judge 1589 - Sokoban. Uji coba dilakukan untuk mendapatkan status *Accepted* pada situs penilaian daring Timus Online Judge dangan cara mengirimkan program ke Timus Online Judge 1589 - Sokoban [1] dan membuktikan bahwa program dapat berjalan sesuai yang diharapkan dengan menerapkan algoritma yang ditawarkan.

5.2.1 Uji Coba Kebenaran Lokal

Uji coba lokal dilakukan dengan menjalankan program untuk menguji apakah dapat berjalan tanpa *error* dan menghasilkan output yang diharapkan.

5.2.1.1 Data Uji Coba Kebenaran Lokal

Data yang digunakan adalah data yang berasal dari contoh input pada Timus Online Judge 1589 – Sokoban [1] dan data uji yang didapat dari semua koleksi *puzzle* yang terdapat pada *website* [2] yang memenuhi *constraint*.

5.2.1.2 Skenario Uji Coba Kebenaran Lokal

Uji coba dilakukan dengan menjalankan program dengan input data uji pada Timus Online Judge 1589 – Sokoban [1] dan melihat apakah program berjalan dan menghasilkan solusi yang diharapkan.

5.2.1.3 Evaluasi Uji Coba Kebenaran Lokal

Evaluasi dilakukan dengan memeriksa hasil output program apakah sama dengan contoh output pada Timus Online Judge 1589 – Sokoban [1]. Tabel kebenaran dapat dilihat pada Tabel 5.1. Output program untuk data uji penulis dapat dilihat pada lampiran A6.

Tabel 5.1 Tabel Data Uji Coba Kebenaran Lokal dengan Data Sampel

Input	**Output**
######## #@ $.# ########	rrRR
###### ## .# #@ ### # * # # $ # # # #######	dddrrrruLdlUUUluRR

5.2.2 Uji Coba Kebenaran Situs Penilaian Daring Timus Online Judge

Uji coba pada situs penilaian daring *Timus Online Judge* dilakukan dengan mengirim kode sumber ke situs Timus Online Judge 1589 – Sokoban [1].

5.2.2.1 Data Uji Coba Kebenaran Situs Penilaian Daring Timus Online Judge

Data yang digunakan pada uji coba ini adalah data yang ada pada situs Timus Online Judge 1589 - Sokoban[1] sehingga data input yang digunakan untuk pengujian tidak dapat diketahui.

5.2.2.2 Skenario Uji Coba Kebenaran Situs Penilaian Daring Timus Online Judge

Uji coba dilakukan dengan mengirim program dengan algoritma *bidirectional search* dengan beberapa variasi untuk *forward move* ke situs penilaian daring Timus Online Judge. Sedangkan untuk *backward move*, akan selalu digunakan algoritma BFS. Variasi dari *forward move* antara lain dengan menggunakan algoritma *A** (yang merupakan pilihan utama), *Greedy DFS*, dan *DFS*. Program yang diuji juga menggunakan beberapa variasi pendekatan heuristik, antara lain *manhattan distance*, *euclidean distance*, dan *goal pull distance*.

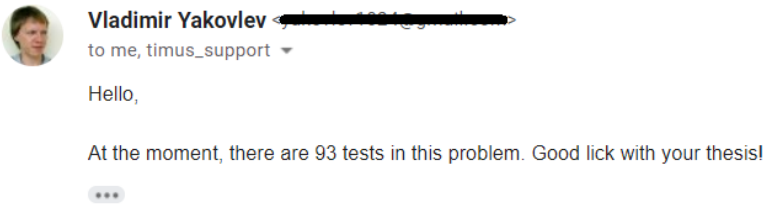
5.2.2.3 Evaluasi Uji Coba Kebenaran Situs Penilaian Daring Timus Online Judge

Evaluasi dilakukan dengan melihat status pengumpulan yang didapatkan pada Timus Online Judge 1589 – Sokoban [1]. Hasil uji kebenaran situs penilaian daring *Timus Online Judge* dapat dilihat pada Gambar 5.1.

ID	Date	Author	Problem	Language	Judgement result	Test #	Execution time	Memory used
8626549	23:27:03 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.007	24 264 KB

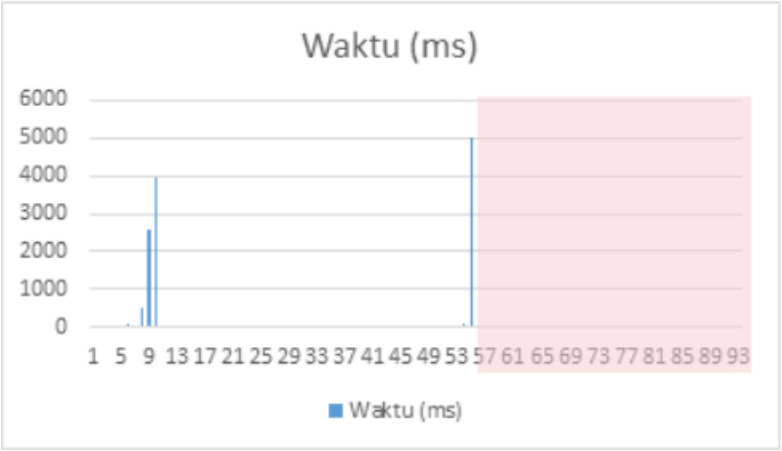
Gambar 5.1 Hasil Uji Coba Kebenaran Situs Penilaian Daring Timus Online Judge

Jumlah *testcase* yang disediakan oleh daring *Timus Online Judge* ada sebanyak 93 *testcase*, seperti yang diutarakan oleh *problem setter* pada Gambar 5.2.



Gambar 5.2 Jumlah *Testcase* oleh *Problem Setter*

Kinerja hasil uji coba kebenaran situs penilaian daring *Timus Online Judge* ditunjukkan oleh Gambar 5.3.



Gambar 5.3 Kinerja Hasil Uji Coba Kebenaran Situs Penilaian Daring Timus Online Judge

5.2.3 Uji Coba Kinerja Lokal

Subbab ini menjelaskan pengujian kinerja program pada Tugas Akhir ini untuk menyelesaikan permasalahan Timus Online Judge 1589 - Sokoban. Uji coba kinerja dilakukan untuk membandingkan kinerja program utama pada Tugas Akhir ini dibandingkan dengan program yang menggunakan algoritma *bidirectional search* lainnya yang tidak menjadi pilihan solusi pada lingkungan sistem lokal.

5.2.4 Data Uji Coba Kinerja Lokal

Data yang digunakan adalah data uji *custom* milik penulis dan beberapa *dataset* yang didapat dari sebuah *website* referensi. Data uji tersebut terdiri dari beberapa koleksi yang memiliki beberapa tingkatan *puzzle* yang memiliki *constraint* sesuai dengan yang disyaratkan *problem.*

5.2.5 Skenario Uji Coba Kinerja Lokal

Uji coba dilakukan dengan menjalankan program dengan algoritma *bidirectional search*. Untuk *backward move*, akan selalu digunakan algoritma BFS. Sedangkan variasi dari *forward move* antara lain dengan menggunakan algoritma *A** (yang merupakan pilihan utama), *Greedy DFS*, dan *DFS*. Setiap program yang dijalankan akan menggunakan data input yang sama.

5.2.6 Evaluasi Uji Coba Kinerja Lokal

Evaluasi dilakukan dengan membandingkan waktu yang dibutuhkan masing-masing program. Di samping waktu, jumlah *state* yang dikunjungi dari hasil percobaan juga akan ditampilkan.

5.2.6.1 Evaluasi Uji Coba Kinerja Lokal Tiap Algoritma

Data hasil uji kinerja lokal dari tiap algoritma dapat dilihat pada Tabel 5.2. Perbandingan jumlah *state* yang di-*generate* dapat dilihat pada Tabel 5.3.

Tabel 5.2 Hasil Uji Coba Kinerja Lokal untuk Tiap Variasi Algoritma *Bidirectional Search*

No	Puzzle		Waktu (detik)		
	Koleksi	**Level**	**A* - BFS**	**DFS - BFS**	**Greedy DFS - BFS**
1.	Microcosmos	6	0.062	0.804	0.762
2.		8	0.250	0.194	0.236
3.		13	0.078	0.042	0.062
4.		23	0.053	0.040	0.056
5.		39	0.029	0.037	0.045
6.		35	0.155	0.152	0.157
7.		39	0.101	0.028	0.111
8.	Nabokosmos	1	0.126	0.133	0.137
9.		6	0.330	0.187	0.515
10.		17	0.212	0.284	0.337
11.		29	1.494	1.406	1.471
12.		30	1.120	1.970	2.076
13.	Sasquatch5	2	0.030	0.018	0.019

Tabel 5.3 Perbandingan Jumlah *State* Tiap Variasi Algoritma *Bidirectional Search*

No	Puzzle		Jumlah State		
	Koleksi	**Level**	**A* - BFS**	**DFS - BFS**	**Greedy DFS - BFS**
1.	Microcosmos	6	917	6785	6228
2.		8	2318	1954	2301
3.		13	863	575	776
4.		23	623	532	692
5.		39	408	496	525
6.		35	1441	1549	1548
7.		39	1108	352	1247
8.	Nabokosmos	1	1410	1522	1510
9.		6	3621	2320	4967
10.		17	2078	2523	2853
11.		29	10456	10231	10215
12.		30	8529	14464	14463
13.	Sasquatch5	2	453	299	292

Dari hasil percobaan di atas, solusi utama dan solusi berupa *bidirectional search* DFS dan BFS sama-sama unggul jauh sebanyak 1 *testcase*. Hasil yang didapatkan dari *dataset* yang bisa dikatakan sebagai *easy case* belum dapat memastikan algoritma mana yang lebih baik. Untuk itu, dilakukan pengujian kembali menggunakan *dataset hard case*. Batas waktu eksekusi tiap *puzzle*-nya adalah 10 menit (atau 600 detik). Data hasil uji dari *case* ini untuk tiap algoritma dapat dilihat pada Tabel 5.4. Perbandingan jumlah *state* yang di-*generate* dapat dilihat pada Tabel 5.5.

Tabel 5.4 Hasil Uji Coba Tiap Variasi Algoritma *Bidirectional Search* *Dataset Hard Case*

No	A* - BFS			DFS - BFS			Greedy DFS - BFS		
	Waktu (detik)	Push		Waktu	Push		Waktu (detik)	Push	
		F	B		F	B		F	B
1.	93,863	28	10	INF			57,613	20	10
2.	INF			INF			INF		
3.	266,354	16	10	INF			INF		
4.	159,889	3	30	165,639	1	32	171,649	2	31
5.	31,734	40	11	125,583	48	13	14,381	116	9
6.	32,193	38	9	10,471	139	8	1,593	39	6
7.	161,907	19	8	INF			228,668	18	9
8.	160,058	19	8	INF			236,953	18	9
9.	153,870	20	8	INF			0,088	23	3

Tabel 5.5 Perbandingan Jumlah *State* Tiap Variasi Algoritma *Bidirectional Search Dataset Hard Case*

No	Jumlah State		
	A* - BFS	**DFS - BFS**	**Greedy DFS - BFS**
1.	628062		381874
2.			
3.	1640916		
4.	918378	956289	934735
5.	235003	734194	104374
6.	253574	100278	16664
7.	1021064		1308517
8.	1021064		1308517
9.	905508		1319

Dalam *graph* pencarian, kedalaman tingkat pencarian juga menjadi sebuah faktor penting. Untuk itu, dari percobaan menggunakan *hard case* dilakukan kembali dengan konfigurasi batasan *push* masing-masing *graph* adalah jumlah *push* maksimal dari semua *testcase* yang dapat diselesaikan. Konfigurasi *push* ini dapat dilihat pada Tabel 5.6.

Tabel 5.6 Konfigurasi *Push* Tiap Algoritma *Dataset Hard Case*

Algoritma	Forward Push	Backward Push
A* - BFS	40	30
DFS – BFS	139	32
Greedy DFS - BFS	116	31

Data hasil uji tiap algoritma menggunakan konfigurasi *push* dapat dilihat pada Tabel 5.7. Perbandingan jumlah *state* yang di-*generate* dapat dilihat pada Tabel 5.8.

Tabel 5.7 Hasil Uji Coba Kinerja Lokal untuk Tiap Variasi Algoritma *Bidirectional Search* Menggunakan Konfigurasi *Push*

No	A* - BFS			DFS - BFS			Greedy DFS - BFS		
	Waktu	Push		Waktu	Push		Waktu	Push	
		F	B		F	B		F	B
1.	72,226	28	10	INF			42,868	20	10
2.	INF			INF			INF		
3.	247,474	16	10	INF			INF		
4.	138,271	11	30	163,144	1	32	147,780	2	31
5.	30,843	40	11	128,841	48	13	11,643	116	9
6.	32,175	38	9	11,415	139	8	1,457	39	6
7.	163,596	19	8	INF			180,182	18	9
8.	157,018	19	8	INF			184,035	18	9
9.	137,797	20	8	INF			0.091	23	3

Tabel 5.8 Perbandingan Jumlah *State* Tiap Variasi Algoritma *Bidirectional Search* Menggunakan Konfigurasi *Push*

No	Jumlah State		
	A* - BFS	**DFS - BFS**	**Greedy DFS - BFS**
1.	575385		359807
2.			
3.	1628852		
4.	918573	954247	930116
5.	234669	703494	97493
6.	259253	99889	16664
7.	1017632		1231145
8.	1017632		1231145
9.	879064		1319

5.2.6.2 Evaluasi Uji Coba Kinerja Lokal Tiap *Heuristic*

Algoritma yang digunakan untuk uji coba *heuristic* adalah algoritma yang digunakan untuk solusi utama. Data hasil uji kinerja lokal dari tiap variasi *heuristic* dapat dilihat pada Tabel 5.9. Perbandingan jumlah *state* yang di-*generate* dapat dilihat pada Tabel 5.10.

Tabel 5.9 Hasil Uji Coba Kinerja Lokal untuk Tiap *Heuristic*

No	Puzzle		Waktu (detik)		
	Koleksi	**Level**	**Manhat-tan**	**Eucli-dean**	**Goal Pull**
1.	Microcosmos	6	0.063	0.057	0.061
2.		8	0.241	0.244	0.233
3.		13	0.077	0.071	0.077
4.		23	0.060	0.052	0.052
5.		39	0.032	0.040	0.029
6.		35	0.159	0.151	0.153
7.		39	0.104	0.104	0.104
8.	Nabokosmos	1	0.131	0.118	0.130
9.		6	0.388	0.302	0.331
10.		17	0.218	0.244	0.223
11.		29	1.650	1.550	1.570
12.		30	1.208	1.167	1.177
13.	Sasquatch5	2	0.034	0.033	0.033

Tabel 5.10 Grafik Perbandingan Jumlah *State* Tiap Variasi Heuristic

No	Puzzle		Jumlah State		
	Koleksi	**Level**	**Manha-ttan**	**Euclid-ean**	**Goal Pull**
1.	Microcosmos	6	917	821	917
2.		8	2318	2348	2318
3.		13	867	823	863
4.		23	690	621	623
5.		39	408	509	408
6.		35	1441	1229	1441
7.		39	1141	1110	1108
8.	Nabokosmos	1	1410	1213	1410
9.		6	3749	3216	3621
10.		17	2078	2291	2078
11.		29	10451	10218	10456
12.		30	8529	8403	8529
13.	Sasquatch5	2	453	439	453

Dari hasil percobaan di atas, hasil yang didapatkan dari *dataset easy case* belum dapat memastikan *heuristic* mana yang lebih baik, dikarenakan hasil yang tidak jauh mengungguli satu sama lainnya. Untuk itu, kembali dilakukan pengujian menggunakan *dataset hard case*. Data hasil uji dari *dataset* ini untuk tiap *heuristic* dapat dilihat pada Tabel 5.11. Perbandingan jumlah *state* yang di-*generate* dapat dilihat pada Tabel 5.12.

Tabel 5.11 Hasil Uji Coba Tiap Variasi Heurtistik *Dataset Hard Case*

No	Manhttan			Euclidean			Goal Pull		
	Waktu (detik)	Push		Waktu	Push		Waktu (detik)	Push	
		F	B		F	B		F	B
1.	INF			INF			93,863	28	10
2.	INF			INF			INF		
3.	259,559	16	10	INF			266,354	16	10
4.	139,702	13	28	159,009	2	31	159,889	3	30
5.	33,450	42	11	34,396	92	11	31,734	40	11
6.	15,399	39	8	21,688	34	9	32,193	38	9
7.	161,798	19	8	0.032	26	3	161,907	19	8
8.	163,496	19	8	0.031	26	3	160,058	19	8
9.	164,272	20	8	0,143	29	3	153,870	20	8

Tabel 5.12 Grafik Perbandingan Jumlah *State* Tiap Variasi *Heuristic Dataset Hard Case*

No	Jumlah State		
	Manhattan	Euclidean	Goal Pull
1.			628062
2.			
3.	1460916		1640916
4.	854613	947408	918378
5.	237526	259700	235003
6.	133299	193076	253574
7.	1021064	558	1021064
8.	1021064	558	1021064
9.	905508	1712	905508

Percobaan menggunakan *hard case* dilakukan kembali dengan konfigurasi batasan *push* masing-masing *graph*, untuk mengetahui hasil optimal yang dihasilkan oleh tiap *heuristic*. Konfigurasi *push* ini dapat dilihat pada Tabel 5.13.

Tabel 5.13 Konfigurasi *Push* Tiap *Heuristic Dataset Hard Case*

Heuristic	Forward Push	Backward Push
Manhattan	42	28
Euclidean	92	31
Goal Pull	40	30

Data hasil uji tiap algoritma menggunakan konfigurasi *push* dapat dilihat pada Tabel 5.14. Perbandingan jumlah *state* yang di-*generate* dapat dilihat pada Tabel 5.15.

Tabel 5.14 Hasil Uji Coba Kinerja Lokal untuk Tiap Variasi *Heuristic* Menggunakan Konfigurasi *Push*

No	Manhttan			Euclidean			Goal Pull		
	Waktu (detik)	Push		Waktu	Push		Waktu (detik)	Push	
		F	B		F	B		F	B
1.	510,052	18	32	INF			72,226	28	10
2.	INF			INF			INF		
3.	211,301	16	26	INF			247,474	16	10
4.	115,813	16	41	153,004	946445	2	138,271	11	30
5.	25,135	41	51	33,168	256711	92	30,843	40	11
6.	15,646	41	49	22,154	193047	34	32,175	38	9
7.	165,357	19	27	0,033	558	26	163,596	19	8
8.	169,154	19	27	0,004	558	26	157,018	19	8
9.	149,977	20	28	0,151	1712	29	137,797	20	8

Tabel 5.15 Grafik Perbandingan Jumlah *State* Tiap Variasi *Heuristic* Menggunakan Konfigurasi *Push*

No	Jumlah State		
	Manhattan	**Euclidean**	**Goal Pull**
1.	2982234		575385
2.			
3.	1391480		1628852
4.	758772	946445	918573
5.	199688	256711	234669
6.	133737	193047	259253
7.	1019267	558	1017632
8.	1019267	558	1017632
9.	887673	1712	879064

5.2.7 Uji Coba Kinerja Situs Penilaian Daring Timus Online Judge

Subbab ini menjelaskan pengujian kinerja program pada Tugas Akhir ini untuk menyelesaikan permasalahan Timus Online Judge 1589 - Sokoban. Uji coba kinerja dilakukan untuk membandingkan kinerja program utama pada Tugas Akhir ini dibandingkan program dengan variasi algoritma *bidirectional search* lainnya pada situs penilaian daring Timus Online Judge.

5.2.7.1 Data Uji Coba Kinerja Situs Penilaian Daring Timus Online Judge

Data yang digunakan pada uji coba ini adalah data yang ada pada situs Timus Online Judge 1589 – Sokoban [1] sehingga data input yang digunakan untuk pengujian tidak dapat diketahui.

5.2.7.2 Skenario Uji Coba Kinerja Situs Penilaian Daring Timus Online Judge

Uji coba dilakukan dengan mengirim program untuk semua variasi algoritma *bidirectional search* ke situs penilaian daring Timus Online Judge sebanyak 10 kali.

5.2.7.3 Evaluasi Uji Coba Kinerja Situs Penilaian Daring Timus Online Judge

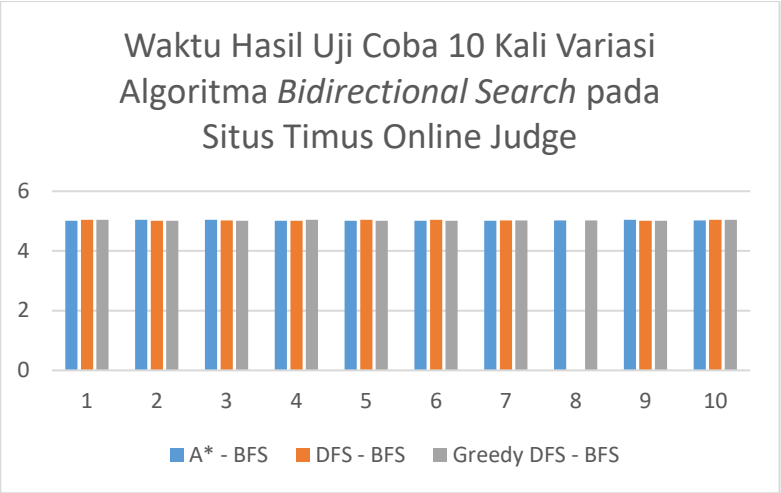
Evaluasi dilakukan dengan membandingkan waktu yang dibutuhkan dan penggunaan memori masing-masing program. Tiap percobaan yang dilakukan menggunakan batasan *push* yang berbeda untuk *forward move*. Data hasil uji kinerja tiap variasi algoritma *bidirectional search* dapat dilihat pada Tabel 5.16 dan 5.17. Grafik waktu dan memori pada penumpulan kode berkas dapat dilihat pada Gambar 5.4 dan 5.5.

Tabel 5.16 Tabel Data Waktu Hasil Uji Coba Kinerja Variasi Algoritma *Bidirectional Search* Situs Penilaian Daring Timus Online Judge

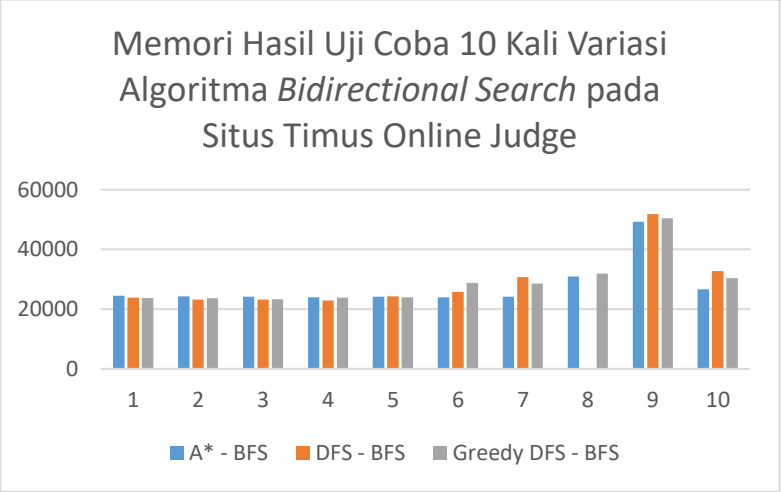
Percobaan ke-	**Batas *Push***	**Waktu (detik)**		
		A* - BFS	**DFS - BFS**	**Greedy DFS - BFS**
1	oo (*infiny*)	5,007	5,038	5,038
2	1000	5,038	5,007	5,007
3	500	5,038	5,023	5,007
4	250	5,007	5,007	5,038
5	150	5,007	5,038	5,007
6	100	5,007	5,038	5,007
7	50	5,007	5,023	5,023
8	25	5,023	3,244	5,023
9	12	5,038	5,007	5,007
10	37	5,023	5,038	5,038

Tabel 5.17 Tabel Data Memori Hasil Uji Coba Kinerja Variasi Algoritma *Bidirectional Search* Situs Penilaian Daring Timus Online Judge

Percobaan ke-	**Batas *Push***	**Memori (KB)**		
		A* - BFS	**DFS - BFS**	**Greedy DFS - BFS**
1	oo (*infiny*)	24464	23844	23720
2	1000	24260	23268	23696
3	500	24136	23204	23376
4	250	23940	22928	23828
5	150	24128	24336	23956
6	100	23972	25792	28768
7	50	24200	30716	28528
8	25	30964	18812	31920
9	12	49276	51888	50460
10	37	26596	32752	30408



Gambar 5.4 Grafik Waktu Hasil Pengumpulan Kode Berkas Variasi Algoritma *Bidirectional Search* Sebanyak 10 Kali



Gambar 5.5 Grafik Memori Hasil Pengumpulan Kode Berkas Variasi Algoritma *Bidirectional Search* Sebanyak 10 Kali

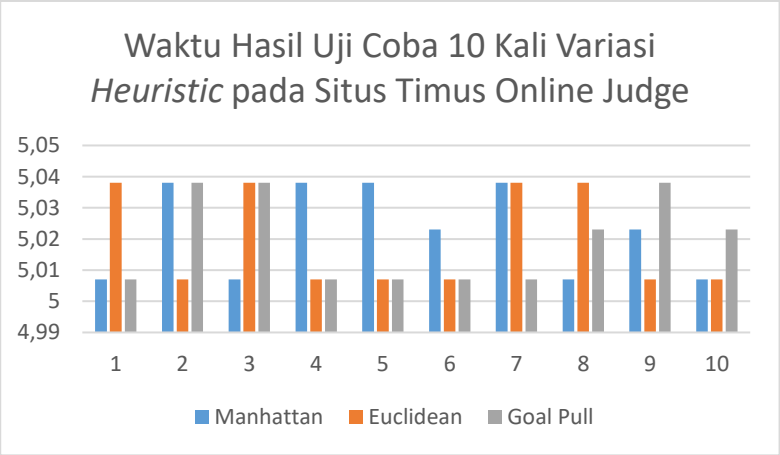
Data hasil uji kinerja tiap variasi *heuristic* dapat dilihat pada Tabel 5.18 dan 5.19. Grafik waktu dan memori pada pengumpulan kode berkas dapat dilihat pada Gambar 5.6 dan 5.7.

Tabel 5.18 Tabel Data Waktu Hasil Uji Coba Kinerja Variasi *Heuristic* Situs Penilaian Daring Timus Online Judge

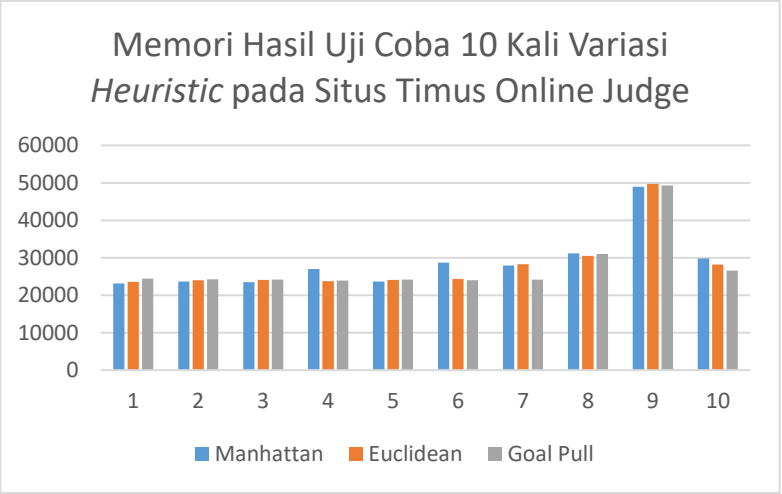
Percobaan ke-	**Batas *Push***	**Waktu (detik)**		
		Manhattan	**Euclidean**	**Goal Pull**
1	oo (*infiny*)	5,007	5,038	5,007
2	1000	5,038	5,007	5,038
3	500	5,007	5,038	5,038
4	250	5,038	5,007	5,007
5	150	5,038	5,007	5,007
6	100	5,023	5,007	5,007
7	50	5,038	5,038	5,007
8	25	5,007	5,038	5,023
9	12	5,023	5,007	5,038
10	37	5,007	5,007	5,023

Tabel 5.19 Tabel Data Memori Hasil Uji Coba Kinerja Variasi *Heuristic* Situs Penilaian Daring Timus Online Judge

Percobaan ke-	Batas *Push*	Memori (KB)		
		Manhattan	Euclidean	Goal Pull
1	oo (*infiny*)	23180	23536	24464
2	1000	23692	24032	24260
3	500	23500	24060	24136
4	250	26964	23776	23940
5	150	23692	24060	24128
6	100	28700	24352	23972
7	50	27916	28240	24200
8	25	31208	30504	30964
9	12	48912	49680	49276
10	37	29820	28212	26596



Gambar 5.6 Grafik Waktu Hasil Pengumpulan Kode Berkas Variasi *Heuristic* Sebanyak 10 Kali



Gambar 5.7 Grafik Memori Hasil Pengumpulan Kode Berkas Variasi *Heuristic* Sebanyak 10 Kali

5.3 Analisis dan Kesimpulan Umum

Berdasarkan hasil uji coba lokal dan hasil pengumpulan kode berkas ke daring *Timus Online Judge* pada subbab 5.2, didapatkan bahwa:

- Hasil output program sudah sesuai dengan yang diharapkan.
- Variasi algoritma *bidirectional search* terbaik dalam menyelesaikan semua *testcase* yang dimiliki oleh penulis adalah algoritma *bidirectional search* yang menggunakan A* sebagai *forward move* dan BFS sebagai *backward move*. Hanya *testcase* ke-2 pada *dataset hard case* yang tidak dapat diselesaikan oleh algoritma ini dalam *constraint* yang diberikan oleh penulis.
- Secara keseluruhan dari hasil uji coba menggunakan *dataset hard case*, algoritma *bidirectional search* yang menggunakan A* sebagai *forward move* dan BFS sebagai

backward move adalah algoritma yang memiliki hasil output dengan jumlah *push* paling sedikit. Sehingga dapat dikatakan bahwa algoritma ini juga yang paling mendekati hasil optimal.

- Pendekatan *heuristic* terbaik dari semua variasi yang telah dibuat adalah pendekatan *heuristic* menggunakan *goal pull distance*. Yang mana pendekatan *heuristic* ini berhasil menyelesaikan *testcase* lebih banyak, dibandingkan dengan pendekatan *heuristic* lain yang menggunakan algoritma yang sama.
- Konfigurasi batasan *push* memiliki andil besar dalam kinerja sistem untuk menyelesaikan *puzzle*. Sebagai bukti, algoritma *bidirectional search* yang menggunakan pendekatan *heuristic manhattan distance* pada akhirnya dapat menyelesaikan *testcase* ke-1 pada *dataset hard case*. Yang mana apabila tanpa menggunakan konfigurasi batasan *push*, *testcase* tersebut tidak dapat diselesaikan oleh algoritma *bidirectional search* dengan pendekatan *heuristic* ini dalam *constraint* waktu yang telah ditetapkan oleh penulis.

Dari poin-poin analisis diatas, dapat disimpulkan bahwa solusi utama, algoritma *bidirectional search* dengan A* sebagai *forward move* dan BFS sebagai *backward move* serta dengan pendekatan *heuristic goal pull distance*, adalah solusi terbaik yang didapatkan dalam menyelesaikan semua *testcase* yang dimiliki oleh penulis.

(Halaman ini sengaja dikosongkan)

BAB VI
KESIMPULAN DAN SARAN

Bab ini membahas tentang kesimpulan yang didasari oleh analisis dan kesimpulan umum dari hasil uji coba yang telah dilakukan pada bab sebelumnya. Kesimpulan nantinya sebagai jawaban dari rumusan masalah yang dikemukakan pada bab 1. Selain kesimpulan, juga terdapat saran yang ditujukan untuk pengembangan penelitian lebih lanjut di masa depan.

6.1 Kesimpulan

Dalam pengerjaan Tugas Akhir ini setelah melalui tahap perancangan aplikasi, implementasi metode, serta uji coba, diperoleh kesimpulan sebagai berikut.

1. Sistem yang diajukan belum dapat menyelesaikan semua studi kasus dengan benar.
2. Sistem yang diajukan dapat menembus hingga *testcase* ke-55 dalam waktu 5 detik dengan penggunaan memori sebesar 24MB pada daring *Timus Online Judge* untuk studi kasus 1589 – Sokoban.
3. Solusi terbaik yang diterapkan dalam sistem adalah solusi yang paling banyak dapat menyelesaikan *testcase* milik penulis, yaitu algoritma *bidirectional search* dengan A* sebagai *forward move* dan BFS sebagai *backward move*.
4. *Heuristic* terbaik dalam menyelesaikan *testcase* yang dimiliki oleh penulis adalah *goal pull distance*, yang berhasil menyelesaikan hampir semua *testcase* pada *dataset hard case*, dan tinggal menyisakan menyelesaikan satu *testcase* pada konfigurasi normal (yang tidak menggunakan batasan *push*).
5. Konfigurasi batasan *push* berpengaruh besar dalam kinerja sistem untuk menyelesaikan *puzzle*.

6.2 Saran

Saran yang diberikan untuk pengembangan sistem penyelesaian masalah pada Timus Online Judge 1589 – Sokoban dapat dilihat di bawah ini.

1. Perlu menemukan algoritma yang lebih efektif untuk menyelesaikan permasalahan.
2. Implementasi deteksi *deadlock* yang lebih beragam dari sistem saat ini, seperti deteksi *corral deadlock* dan *bipartite deadlock*.
3. Implementasi deteksi *simple deadlock* secara dinamis untuk setiap *state*.
4. Perlu dilakukan riset lebih lanjut untuk konfigurasi batasan *push* pada sistem yang telah ditemukan benar nantinya.
5. Perlu implementasi sistem yang lebih efektif, agar pengembangan fitur dari sistem masih berada dalam batas ukuran *file* yang diperbolehkan (yaitu 64KB).

DAFTAR PUSTAKA

[1] Vasilev, Stanislav dan Sergey P. 2011. 1589. Sokoban, [Online], Available: http://acm.timus.ru/problem.aspx?space=1&num=1589, diakses pada 12 Oktober 2018 pukul 15.32.

[2] Sokoban for the Macintosh. 2006. Sokoban Levels, [Online]. Available: http://sneezingtiger.com/sokoban/levels.html, diakses pada 11 Januari 2020 pukul 21.16.

[3] Wikipedia. 2019. Sokoban, [Online]. Available: https://en.wikipedia.org/wiki/Sokoban, diakses pada 22 Januari 2020 pukul 05.24.

[4] Virkkala, Timo. 2011. *Solving Sokoban*. Helsinki: University of Helsinki

[5] Froleyks, Nils. 2016. *Using Algorithm Portfolio to Solve Sokoban*. Karlsruhe: Karlsruher Institut für Technologie.

[6] Wikipedia. 2019. Heuristik, [Online]. Available: https://id.wikipedia.org/wiki/Heuristik, diakses pada 10 Januari 2020 pukul 11.25.

[7] Sokoban Wiki. 2019. Solver, [Online]. Available: http://sokobano.de/wiki/index.php?title=Solver, diakses pada 10 Januari 2020 pukul 10.56.

[8] Sokoban Wiki. 2019. Deadlocks, [Online]. Available: http://sokobano.de/wiki/index.php?title=Deadlocks, diakses pada 10 Januari 2020 pukul 10.57.

(Halaman ini sengaja dikosongkan)

LAMPIRAN A: Hasil Uji Coba pada Situs *Timus Online Judge* Sebanyak 10 Kali

Berikut lampiran hasil uji coba program utama sebanyak 10 kali ditunjukkan oleh tabel A1.

Tabel A1 Hasil Uji Coba Program Utama (*Bidirectional Search* A* dan BFS)

ID	Date	Author	Problem	Language	Judgement result	Test #	Execution time	Memory used
8626072	18:59:29 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.023	26 596 KB
8626069	18:57:48 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	26	5.038	49 276 KB
8626068	18:56:30 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.023	30 964 KB
8626067	18:52:06 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.007	24 200 KB
8626065	18:50:31 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.007	23 972 KB
8626061	18:49:04 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.007	24 128 KB
8626058	18:48:03 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.007	23 940 KB
8626056	18:46:50 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.038	24 136 KB
8626050	18:43:50 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.038	24 260 KB
8626049	18:39:22 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.007	24 464 KB

Berikut lampiran hasil uji coba program tambahan (*bidirectional search* DFS dan BFS) sebanyak 10 kali ditunjukkan oleh tabel A2.

Tabel A2 Hasil Uji Coba Program Tambahan (*Bidirectional Search* DFS dan BFS)

ID	Date	Author	Problem	Language	Judgement result	Test #	Execution time	Memory used
8626124	19:30:04 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.038	32 752 KB
8626122	19:29:02 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	26	5.007	51 888 KB
8626121	19:28:45 13 Nov 2019	adityarev	1589	G++ 7.1	Runtime error (non-zero exit code)	32	3.244	18 812 KB
8626119	19:26:59 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.023	30 716 KB
8626118	19:26:09 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.038	25 792 KB
8626116	19:25:42 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.038	24 336 KB
8626106	19:22:20 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.007	22 928 KB
8626101	19:20:01 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.023	23 204 KB
8626097	19:18:09 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.007	23 268 KB
8626095	19:17:03 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.038	23 844 KB

Berikut lampiran hasil uji coba program tambahan (*bidirectional search greedy* DFS dan BFS) ditunjukkan oleh tabel A3.

Tabel A3 Hasil Uji Coba Program Tambahan (*Bidirectional Search Greedy* DFS dan BFS)

ID	Date	Author	Problem	Language	Judgement result	Test #	Execution time	Memory used
8626223	20:34:53 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.038	30 408 KB
8626221	20:34:33 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	26	5.007	50 460 KB
8626220	20:33:47 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.023	31 920 KB
8626219	20:33:26 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.023	28 528 KB
8626218	20:33:09 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.007	28 768 KB
8626217	20:32:46 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.007	23 956 KB
8626216	20:32:22 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.038	23 828 KB
8626213	20:31:52 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.007	23 376 KB
8626212	20:31:25 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.007	23 696 KB
8626211	20:31:03 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.038	23 720 KB

Berikut lampiran hasil uji coba program tambahan (*manhattan heuristic*) ditunjukkan oleh tabel A4.

Tabel A4 Hasil Uji Coba Program Tambahan (*Manhattan Heuristic*)

ID	Date	Author	Problem	Language	Judgement result	Test #	Execution time	Memory used
8626240	20:43:13 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.007	29 820 KB
8626238	20:42:21 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	26	5.023	48 912 KB
8626236	20:41:51 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.007	31 208 KB
8626235	20:41:22 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.038	27 916 KB
8626233	20:40:39 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.023	28 700 KB
8626230	20:40:02 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.038	23 692 KB
8626229	20:39:34 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.038	26 964 KB
8626228	20:38:58 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.007	23 500 KB
8626227	20:38:29 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.038	23 692 KB
8626226	20:37:57 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.007	23 180 KB

Berikut lampiran hasil uji coba program tambahan (*euclidean heuristic*) ditunjukkan oleh tabel A5.

Tabel A5 Hasil Uji Coba Program Tambahan (*Euclidean Heuristic*)

ID	Date	Author	Problem	Language	Judgement result	Test #	Execution time	Memory used
8626267	20:52:17 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.007	28 212 KB
8626265	20:51:56 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	26	5.007	49 680 KB
8626263	20:51:38 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.038	30 504 KB
8626262	20:51:19 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.038	28 240 KB
8626261	20:51:03 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.007	24 352 KB
8626259	20:50:44 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.007	24 060 KB
8626257	20:50:25 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.007	23 776 KB
8626256	20:50:09 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.038	24 060 KB
8626255	20:49:52 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.007	24 032 KB
8626254	20:49:26 13 Nov 2019	adityarev	1589	G++ 7.1	Time limit exceeded	55	5.038	23 536 KB

(Halaman ini sengaja dikosongkan)

LAMPIRAN B: Data Uji

Berikut data uji coba untuk koleksi *puzzle microban* ditunjukkan oleh tabel B1 hingga tabel B10.

Tabel B1 Data Uji Coba Koleksi *Puzzle Microban* (1)

Level	Puzzle	Output
1	#### # .# # ### #*@ # # $ # # ### ####	dlUrrrdLullddrUluRuulDrddr ruLdlUU
2	###### # # # #@ # # $* # # .* # # # ######	rddLruulDuullddR
4	######## # # # .**$@# # # ##### # ####	ullDullddrRuLurrrdLLrrddlU ruL
5	####### # # # .$. # ## $@$ # # .$. # # # ########	LuRdlddrruLdlluUrrdrruLruu lldDluulDrdrrddllURuullDur rddlUddlluR

Tabel B2 Data Uji Coba Koleksi *Puzzle Microban* (2)

Level	Puzzle	Output
7	``` ####### # # # .$. # # $.$ # # .$. # # $.$ # # @ # ####### ```	``` rruuuulLrrddddlUruuulldDll ddrUdrruruuLululldRRllddRU dddrruruulDLrrddlU ```

9	```
	#####
	#. ##
	#@$$ #
	## #
	 ## #
	  ##.#
	   ###
	``
		``
		urrDulldRdRluurDrDDlUruLdl
		UruL
		``

14	``` ####### # # # # # # #. $*@# # ### ##### ```	``` uullllddrrUlluurrrrddLLLr rruullDurrddlLdllURRuullD ```
17	``` ##### # @ # #...# #$$$## # # # # ###### ```	``` dlDurrDulldRddlUUddrrruLUd dlUU ```

Tabel B3 Data Uji Coba Koleksi *Puzzle Microban* (3)

Level	Puzzle	Output

19	########
	# .. #
	# @$$ #
	##### ##
	  # #
	  # #
	  # #
	  ####
		urrDDDlddrUUUUruLLLdRurDDD
		lddrUUUUllllluRR

21	#### # #### # . . # # $$#@# ## # ######	dlllUdrUlluurDRddlU
23	####### # * # # # ## # ## #$@.# # # #####	ruulluRdrdddllUUdrruulullld RddrruuruLLrdddlluuRurDDul lluR
24	# ##### # # ###$$@# # ### # # # . . # #######	ullDDlddrruLuuurrdLulDDDul lddrRuulD

Tabel B4 Data Uji Coba Koleksi *Puzzle Microban* (4)

Level	Puzzle	Output
25	#### # ### # $$ # ##... # # @$ # # ### #####	urrdLullddrUluRluurDlddrrr uuLLulD
26	##### # @ # # # ###$ # # ...# # $$ # ### # ####	drddlUdrddlUUlldRurrddlUru LruuulldRurDDulD
27	###### # .# # ## ## # $$@# # # # #. ### #####	dlULddlluuuurrrDullldddrr uuLrdrruLuullldDDurRdrUU
28	##### # # # @ # # $$### ##. . # # # ######	rDulldRddrruLUddlUUluurrDD ldRuuulDD

Tabel B5 Data Uji Coba Koleksi *Puzzle Microban* (5)

Level	Puzzle	Output
30	#### # ### # $$ # #... # # @$ # # ## #####	luuurDldddrrUruuLLulD
31	#### ## # ##@$.## # $$ # # . . # ### # #####	RlDururDDrddlluUluR
32	#### ## ### # # #.**$@# # ### ## # ####	ullDurrdLullldrURurrdLLLd drUluRurrdLululD
33	####### #. # # # $ # #. $#@# # $ # #. # # #######	ulLLdlUrrrrddlLrruullDlldd rUluurrrrddlLLruulldDrruL

Tabel B6 Data Uji Coba Koleksi *Puzzle Microban* (6)

Level	Puzzle	Output

40	` #####`
	` #   #`
	`##   ##`
	`# $$$ #`
	`# .+. #`
	`#######`
		`UdrruLuulldDuRurDlDurD`

41	`#######`
	`#     #`
	`#@$$$ ##`
	`#  #...#`
	`##    ##`
	` ######`
		`urrrDullDRurrdddlllUluurrd`
		`RurDllulldRRRllddrrURdlllu`
		`uurrrDulD`

42	`   ####`
	`   #  #`
	`   #@ #`
	`####$.#`
	`#   $.#`
	`# # $.#`
	`#    ##`
	`######`
		`rdddLdllluurrRUruulDDDllld`
		`drrrUrUdldlUrruUlDLdRdlllu`
		`uRRRuuurD`

44	`#####`
	`#@$.#`
	`#####`
		`R`

45	`######`
	`#... #`
	`#  $ #`
	`# #$##`
	`#  $ #`
	`#  @ #`
	`######`
		`lluuuurrrdLulldddrrruLUUd`
		`dLdlUrruuruLdddllUURlddrru`
		`U`

Tabel B7 Data Uji Coba Koleksi *Puzzle Microban* (7)

Level	Puzzle	Output
46	###### ## # # ## # # # $ # # * .# ## #@## # # #####	dlluuRuRDrUdllddrrUdlluur urrdLddlluuluururrrDD
48	###### # @ # # # ## # .# ## # .$$$ # # .# # #### # #####	rddDLruuulldd1ddrUrrurDrdd llUdrruululuulldlddrUluurr rddrdLLruuulllddrDrrddrUru
51	#### # ### # ### # $*@ # ### .# # # # ######	rddllluURuLdddrrruuLrddlll uuurDllluurDldRR
52	#### ### @# # $ # # *.# # *.# # $ # ### # ####	dddLrddlUlluuurRurDDulldRl lddrUluR

Tabel B8 Data Uji Coba Koleksi *Puzzle Microban* (8)

Level	Puzzle	Output

53	` #####`
	`##. .##`
	`# * * #`
	`#  #  #`
	`# $ $ #`
	`## @ ##`
	` #####`
		`lUluuRurrDLrrddLUUddd1luRd`
		`rUl1luuRlddrUUddrrruulLrrd`
		`dlU`

56	`#####`
	`#   ###`
	`#  $  #`
	`##* . #`
	` #   @#`
	` ######`
		`luuLulldRurDrddllURuulD`

58	`####`
	`#  ####`
	`#.*$  #`
	`# .$# #`
	`## @  #`
	` #   ##`
	` #####`
		`luluurDRDulldRddrruruuLLLd`
		`ddrruLUddlUrrruulL`

67	`#####`
	`#   ##`
	`# #  #`
	`#@$*.##`
	`##  . #`
	` # $# #`
	` ##   #`
	`  #####`
		`uurrdrddrddllUdrruulLruulD`
		`uulldd RRDrrddllUlURuuulldd`
		`R`

Tabel B9 Data Uji Coba Koleksi *Puzzle Microban* (9)

Level	Puzzle	Output
82	##### # ### # . ## ##*#$ # # .# $ # # @## ## # # #######	drrruUlUdrruLuLLulldRDDldd rUdrrruuuulLulDrrrddlUruLL
95	######## #@ # # .$$. # # $..$ # # $..$ # # .$$. # # # ########	rrDllddRluurrRllldrUdddrr ruruuLDrddllUL
100	####### # @# # #.$ # #. # $## #.$# # #. # $ # # # # ########	lddrDluuurDrrddrUdrdLdlUru uluurDDDrddlluUUdrrdLuuuul Dllldrr UluRRRurDDullddldd rUluuurrrrdDuullldlddrUluu rrrrddlUdrrddllUUrrdLuuuLL LulDDDrUluRRRurDDDlUruLLLu lDDurrrddrrddlUdlUUUddrruL dlUruuLLLulDrrrrddlUruLLL
103	##### # . ## ### $ # # . $#@# # #$. # # $ ### ## . # #####	ulLulDrrrddllUUdLrdddlluRd rUlURuulDDD

Tabel B10 Data Uji Coba Koleksi *Puzzle Microban* (10)

Level	Puzzle	Output
107	######## # # # $*** # # * * # # * * # # ***. # # @# ########	uuuLDlluRdrruuulllDulldRld dRRRllUdrruLdlluuurrDDrdLu uulldR

Berikut data uji coba untuk koleksi *puzzle microcosmos* ditunjukkan oleh tabel B11 hingga tabel B13.

Tabel B11 Data Uji Coba Koleksi *Puzzle Microcosmos* (1)

Level	Puzzle	Output
6	##### # # # # ## ## * ## # $*$ # # * # # ## . +# #######	uululuullddDuuurrddDuuulld ddRDuluuurrddrdLuuulldddld RdRRRlUULDurUddllluRuRDrru LdlluuurrDDrdL
8	#### ### ### # . # # $ $ # ##.#+#.# # $ $ # ### ## ####	uLurrrdLrddLdlUrruululldRl ulldRddRRUULrddlluUruurDDu rrdLulldlluRdrRurrdddLLrru uullLrrrdddllUUddrruuulLdl lddRluurrurrdddldlUlluurru ulDrrrdddllUURurDllddlluuR uRDllluRdrruulDrdLulDrrrru L

Tabel B12 Data Uji Coba Koleksi *Puzzle Microcosmos* (2)

Level	Puzzle	Output

13	#####
	## ###
	# ...* #
	# #@$# #
	# $ #
	### #$##
	  # #
	  #####
		drrruuLLulldlddRluururDrrr
		ddllUdrruulLulldRRddLrrruu
		LLddlddrrUdlluuruurrddLruu
		llulldRurDrrddlddlluuUdddr
		ruuruullulldlddRluururrdLd
		Duurrrddlddll UUlluuRRRllld
		drrUdlluururrdLddddrruuLUl
		DrrddllUUU

23	##### # ### #.#$ # # @$* # # $ # ##.#.## # # #####	dddrruuUdddlluuuRlluurrDul ldddrddrruuLrddlluUluuurrd DuullddddrddrruuruuLrddlddl luuluuurrDrrddlLrruullDLDR uurrdLDlluRdllU
29	#### ### # ## @$ # # .$ # # .* ## # .$ # ## ## ####	lddRUdlddrUluurrruulDrdLLD lddrUrULrUruulDDlddlluuRDu uRlddrddlUluururDrruulDrdL ulldlddrrrUULDrdLuurruulDr dLulD

35	########
	# * #
	# +*** #
	## $ #
	 ## ##
	  ## #
	  ####
		drdrruruulLLDurrDulldRDuur
		rdLulldldRdRluuRlddrUluurr
		rddLLdlUluluRRRDLddrUrruuL
		DrdLLddrUUdllU

Tabel B13 Data Uji Coba Koleksi *Puzzle Microcosmos* (3)

Level	Puzzle	Output
39	##### ### ## # $$.. # # . @# # ## # # # *$ ## ### # ####	drddlUUUllddRRurruulullDur rdrddlldlluuluRddrruuLrdr ruulullDRllDurrdLrdrruuLrd dllddrUluuluurrdrddLruulul ldRdDrddlUUUlulldRururDDDr ruuLulldldRuurrdLLrrrddllU luRdllluRRdrdrddlUllUUrrDr dLuuluurDDDullluRdrruulDrd Lrdrddl UUU

Berikut data uji coba untuk koleksi *puzzle minicosmos* ditunjukkan oleh tabel B14 hingga tabel B17.

Tabel B14 Data Uji Coba Koleksi *Puzzle Minicosmos* (1)

Level	Puzzle	Output
1	##### ### # # $ # ## # # . # # # # ## # # #@ ### #####	uuluuRlddrrurruullDDullddr RlddrruUluR
2	##### ### # # $ # ## # # . # # # # ##$#. # #@ ### #####	rruullluuRlddrrurruullDlld drrrddllUdrruuuruulldlldd R RlddrrurruullDllluurrDulld drddrrUUddlluuluurrdRdddll uuRurD

Tabel B15 Data Uji Coba Koleksi *Puzzle Minicosmos* (2)

Level	Puzzle	Output

3	<pre> #####
	### #
	# $ # ##
	# # . #
	# . # #
	##$#.$ #
	 #@ ###
	 #####</pre>
		<pre>rruullluuRlddrrrddllUdrruu
		urrddLruuluullDllddRddrrUU
		ddlluuRlluurrDRDL</pre>

9	<pre> #####
	 # #
	##$# ###
	# $@ #
	# # # #
	# #. . #
	# ####
	#####</pre>
		<pre>rddllluuRuullDurrddddldllu
		uuRlddrrurrruuLrddllldllu
		uuruurrdDDurrddlLuuuullddR
		luurrdDDlullddrrURuulD</pre>

10	<pre> #####
	 # #
	##$# ###
	# .$@ #
	# # # #
	# #..$ #
	# ####
	#####</pre>
		<pre>rddLLuluRlllddrrUruuuullD
		urrddddldlluuuRlddrrurrru
		uLrddllldlluuuruurrdDDuuul
		lddlddrrruRldlluuurRluurrd
		Dllddrrruuddlluuuruurrddr
		rddLLrruullD</pre>

15	<pre> ####
	#### #
	# $ #
	# .# ##
	## #. #
	# @ $ #
	# ####
	#####</pre>
		<pre>uuluRdddrrurrdLuuulLdlluRR
		dlddrRurrdLLLdlUUUruRurDDu
		lldlddrrrruLdllluuluRRRurD
		DDrdLLLdlUUdrrruuulllldR</pre>

Tabel B16 Data Uji Coba Koleksi *Puzzle Minicosmos* (3)

Level	Puzzle	Output
16	#### #### # # $ $ # # .# ## ## #. # # @ $ # #. #### #####	rrurrdLLLdlluRRluuruRurDll dlddrRurrdLLLdlluRuururrDD ldLruruulldlluRddDrdLurrur rdLuuulLdlluRRdlddrRurrdLL LdlUUUruRurDDulldlddrrrruL dllluuluRRRurDDDrdLLLdlUUd rrruuulllldR
31	##### # ## # # ## #. #$ # # @ # #.##$## # # ######	lluuurrdrDrdLLruulullddddd rrrUUlLulDuuurrdrDrdLddlll UUUdddrrruuLLulDrrruulullD
32	#### # ### # ## # .#$ # ## @ # #.#$## # # #####	luurrDrdLuulldddrrUULrddl luUUluurDrrDrdLddlluuUluRd dddrruuLrddlluUUluurDDDrru uLrddlluluurD
35	#### ### ### # *$ # # # #@# # # *. # # #### #####	dlLulullddrrUdlluuurrurDl llddrrurrruuLLLrrrddlllUd rrruullDldRuuulDrddldlluuu RRurDDullldddrruUddlluuurR

Tabel B17 Data Uji Coba Koleksi *Puzzle Minicosmos* (4)

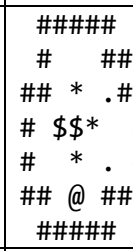
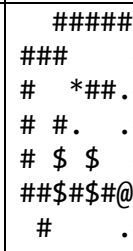
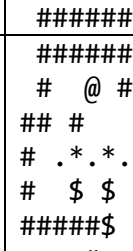
Level	Puzzle	Output

36	#####
	# ##
	# # ###
	# *$ #
	### #@#
	  # *. #
	  # ###
	  ####
		dlLdlUUUlluurrDrDDuLdddrUr
		ruuLLDldRuuulDDuuulldd RRdr
		ddlUUUlluurrDrDDuulullddrR

38	#### # ### ## . # ##@$$$ # # . . ## # ### # ## ####	drrrUruLLDrdLLdlluRRluRRur rdLdLLrruululDDlDururrddlU lldRllddrU
40	#### ## # ## . ## #@$$$ # #. .# # # # # # ### #####	drrUdlluRuRDldlddrrurruuLL ulDururDldllddd rrurruuLLrr ddllULrdrruulullDlDururrdr ddlluUlldRurrrddlldllU

Berikut data uji coba untuk koleksi *puzzle nabokosmos* ditunjukkan oleh tabel B18 hingga tabel B19.

Tabel B18 Data Uji Coba Koleksi *Puzzle Nabokosmos* (1)

Level	Puzzle	Output
1		ruruulLrrddldlluRURlddrUrU dlluRdllluRdrruuullDDldRdr rurruLdldlluRRlluuurrdDLru ulldDldRdrrU
6		ululldRurrdddllUURurDllluR drruuulllDlldd RRRddrrUUUdd dlluuuRldddrruuLulDllluurr DullddrRurrdrddllllUdrrrru ululldRluullddRddrrUUddrru uLuLDrrddllllluuRUrrdLrrddl lUUruLdddlluuRlddrrrruulul DDurrddlLuullddRRR
17		dddLruurrddLruulldRdrddlUU UluurDrDDlddrUluUruLdddrUl uuullddRdRluluurrrddLruul llddldRRluuurrrddlLrrDulld lUrrruulDrdLrdrddlUUUluurD rDDlddrUluUruLdddrUlulLdRu rDuuulDrddlllluRdrrruulull Durrdrddrddl UU

Tabel B19 Data Uji Coba Koleksi *Puzzle Nabokosmos* (2)

Level	Puzzle	Output
29	#### # @## ## . ### # $$$. # #..$.# # # $ $ # ## . ## ######	dlDlddRURUdlddrUdrruLrruul LDLrurrddldlUdlluRdrruruul luLDDurrrddldlllluluuRRdLdd rrruruulLululDDDrUrrrddldl lUUluRRldddrruLdllUdrruUdd lluRurDluulDuuurDrDulldrd dlUrrrdLuluU
30	####### # # # .$. # ##$. $## # * # # $.$## ##@. # #####	UrUrrUdlldlluRURuulDDDldRd rrULUlldRurURDlluuurrDulld ddddrUluuuurrrrdLullldrDr rULdlluuurrDDuulldRlddrUrr dLullddrrUULDlluRurDrruLd lluuurrDDrdL

Berikut data uji coba untuk koleksi *puzzle sasquatch5* ditunjukkan oleh tabel B20.

Tabel B20 Data Uji Coba Koleksi *Puzzle Sasquatch5*

Level	Puzzle	Output
2	##### # @#### # $. # ###$.# # # $.# # # #$. # # ### ######	ldRRdddrruuuLrdddlldllluur Rllddrr Urrruuul LdDLdRuuuLD DrddllluuRRlldrrruUluurrr dddLruuulllddllddrrUdlluur RuuulldRR

Berikut data uji coba untuk koleksi *puzzle sokogen* ditunjukkan oleh tabel B21 hingga tabel B28.

Tabel B21 Data Uji Coba Koleksi *Puzzle Sokogen* (1)

Level	Puzzle	Output
1	##### #@ ## #.$* # # # # # # ######	rDlddrrruuLLulDrrrddllUdrr uululldRddlU
2	###### #@ ## # $$ # # #. .# # # #######	rrDulldRRurDrddllURuulD
3	#### ### # #@ .$## # $ # # #. # # # #######	drrdrruLddllluuurrDuurDDrd dllUUrrdL
4	##### ##@ ## # # # # $*$ # # . # ## . ## #####	dDldRuuurrdDLDlluRuurrddrd L

Tabel B22 Data Uji Coba Koleksi *Puzzle Sokogen* (2)

Level	Puzzle	Output
7	####### #@* ## # $# # # $. # #. ## #######	ddRluuRDlddrrruuuLrdddlllu urDRdL
9	####### #.@* ## #$ $ # # # # # . # #######	dddrrruuLrddllluuulDrddrrr uuluLDDluluRdddlUU
10	####### #@ ..## # #$ ## # # # # $# # # * # #######	rrrddlUdrrddlLUlldRluuuurR llddrrUU
11	##### # # # # ### #*.$ # # #$ # #@ .# #######	rrrruulLrDrdLuuluulldDRddR RuuLrddlllU
12	####### ### @# #. *# # # $ # # $. ## #######	ddLLrruullDllddRURuurrddLd LruruulldLrurrddldlUlldR

Tabel B23 Data Uji Coba Koleksi *Puzzle Sokogen* (3)

Level	Puzzle	Output
14	###### # . # # #. ## # . # # $$$ # # # @# #######	uullllddrUluRluurrdrdrddlU UdLUllddrUluuurrrDullldddr ruUddlluRdrUdrruL
15	####### # #### # *.# # $# # # .$@# #######	LruuLLrrddlLUlldRluuurDRRl llddrUluRR
17	##### #@ ### # $$$ # ## ...# # # # # # #######	rrDulldRdRluurDDlddrrruuuL rdddllluurRllddrUluRluurD
18	####### # #### # ### #$ $@## #. #$ # # . .# #######	DrdLuuLullDRddRRuuLulDrrdd lllUdrrruullDldRuuulD

Tabel B24 Data Uji Coba Koleksi *Puzzle Sokogen* (4)

Level	Puzzle	Output
19	###### #@ .# #$$$ # #. # # # . # ######	DurrrdddlluUluRRldddrruuLu lDrrddlllUdrrruullDldRuuul D
21	###### # @## # #* # # $ # # $..# #######	llddrRdLuluurrDrDulullddrd rUlllldRR
25	####### #. ### # # $ # # * # # ## # # * @# #######	uulLLdlUddRuuruullDDrrrruL dllluurrDullddrddlUUUdrRur rdLrddlL
28	##### # # # # ### #. $$ # #. # # # *@ # #######	LuUddrruuLLdlldRluuuurrDDL rrrddlLUllUdrrdrruulLdlldR uruuullDDuurrddL
30	####### ##. ## #+$$$ # # # .# # ## #######	ddrruUdrruLuLDrddllluuRRur DllldrrURuLL

Tabel B25 Data Uji Coba Koleksi *Puzzle Sokogen* (5)

Level	Puzzle	Output
31	##### # ## #. ## # #.$ # # $$# # ##@. # ######	rUUddrruuLuLDDuuulldddRdRU lluuurrdrdrddLruululL
32	####### ## ## ## #.## # $.## # # $ # # * @# #######	uLuuulldd Rlld drRUrUdldlluu rRdrrdL
33	##### # ## # #$ ## #. * # # $# # #+ * # #######	uurRlluurrDrDrddlLLrrruulL Lrrrddlll UlURddrrruulDrdLu uulullD
43	####### #### @# ##.$$.# # $ ## # # .## # $.## #######	lDLDlldd RRUrUdldlluurRuRD

Tabel B26 Data Uji Coba Koleksi *Puzzle Sokogen* (6)

Level	Puzzle	Output
44	####### ###+ ## # $$ # # #$ # #. .# #######	rDLrrddllUdrruulDldRuuulDr ddllluuRRDrdLLrurruLdlU
46	####### ## ### ## .$ # # $$ # # #. .# #@ ### #######	rruUddlluuRRluurDldlddrrur ruuLDLddlluuRuRldlddrruRld lluururDrruL
48	####### #@* ## #$$$ # #. # # # ..# #######	DuRDRddrruuLuLDlluRdrrrddl lllUdrrrruulllDldRuuulDrrr uLdlddRRlluurDldR
52	##### ##@ ## # $*$ # #. # # # #. # # ## ######	DurDDulldddrrruruuLLdLrurr ddldlUUdrruulullDlDRuurrdL DlluRurrdrddldlllU

Tabel B27 Data Uji Coba Koleksi *Puzzle Sokogen* (7)

Level	Puzzle	Output
55	####### ## .## ## #$## # $ ## # # $.# # . @# #######	llllluuRuurrDullddlddrrrruL dllluuruurrdDuullddRDRdLur UU
56	####### # .$@## # .$$.# #$ # # #. # #######	LDllDrdrrruuLLrrddllUlluuR DRRlddlUruruL
59	####### ##. ## #*.# ## #.$ $ # #$ # # # *@# #######	uuLuullldDuurrddLddllURUUdr rrddLLUdrruulLdlldRRluU
60	####### ## ### ##$. # # # # #** # #@ ### #######	rrUdlluurRurrddLruulldlldd rrUrruullulDrrrddllUdrruul DlddlluuRuRDrruL

Tabel B28 Data Uji Coba Koleksi *Puzzle Sokogen* (8)

Level	Puzzle	Output
62	####### # $.@## # $ # # $# # # .*. # #######	dLrrddlLUlluuRDRddrruuLuLD llddRRlluurrDulDluuR
70	####### ## ### ## # # $. .# # #$$ # #@ .### #######	rrUUddlluuRuRldlddrruUrrdL ulDuluurDldRdrruuLrddlluRl luurD

Berikut data uji coba untuk koleksi *puzzle yoshio* ditunjukkan oleh tabel B29 hingga tabel B42.

Tabel B29 Data Uji Coba Koleksi *Puzzle Yoshio* (1)

Level	Puzzle	Output
1	###### ## . # # * # # # .$ # # #$## ## @ # #####	luluuRurrrddlLrruulllldlddr drrUdlluluururrrddLddlluUd drruuLUdrddlluluuRuRDllddr UddrruuL

Tabel B30 Data Uji Coba Koleksi *Puzzle Yoshio* (2)

Level	Puzzle	Output
2	####### # .@ # # #.# # # $ # #.$$ ## # ### ####	rddLruullllddddrUluuurrrrd dldLUrruullllddRldRluuurrr rddlLLrrruullllldDrrUdlluur RddlddlUUrrrdLullddrUruuul lDDRdrUrruuL
3	#### #### @# # *$ # # # ## .### #$ # # .# ####	ddllddddlUruurruulDrdLuLDll uRRdDDlUruulldRurDrruL
4	### ### #.###.# # # .# # $$ @# # $ # # # # # #### ####	dlLruulDLDlUUddrRurrddlUUl uRdrUddllllddrUUluRRRdrU
5	#### # @## #### # #. #$$ # # ## #. $## ##. # #####	dlDurrdLdLLdlluurDDurrruul DrdLLdlluurDrddrUUllldRurr ruulDrdLLLdlUrrrdLL

Tabel B31 Data Uji Coba Koleksi *Puzzle Yoshio* (3)

Level	Puzzle	Output
6	##### # ..#### # $ # # #$# # # @ .$ # ########	luuurrdrrrddLLLrrruulllulD rrrrddllUdrruulLLulldRlddR RRllUUddrruuLrddllluuuRldd rUrrrrddL
7	##### ### .# # $ # # # *$ # # .#@ # # ## # ## #####	dllluuuRDDlddrruruuLrruull DlDururrddlLullдddRluuRRRd rUUdllllddrUluurrdrddLdlU
8	####### #. @.# # $# ## # # $. # # $# # #### # #####	lDDuurrddLrrddllULURlUluRd llddRluurrdrddrruuLuuLLLdl ddrRuuurrddLruulldllddrrUU luRddrrrddllUluRluuR
9	##### #. .### #.#$$ # # @ # # $# # ## ## #####	ddllUluRRlluurrDullдddrdrr uruuLDLUdLdlUdrdrrUruLLrdd lluluUddrdrruulLdlU

Tabel B32 Data Uji Coba Koleksi *Puzzle Yoshio* (4)

Level	Puzzle	Output
10	##### #. ### # # # # . # # # $*$ # ##@ ### # # ####	UdrUlluuurrdrrddLLrruullDL DlUUddrRddlUUruurrddL

11	########
	#. . #
	# # # #
	#@$ $.#
	##### $#
	    # #
	    ####
		uurrrrdrdLddrUUluullddRluu
		rrdDrddlUUlluurrrDullllldd
		RRuurrrdDlUdddrUluuruLLLLr
		rrddddrUluullddRluurrdDrddl
		UUlluurrrDullldrrU

12	#### # # # ##### # .* # ##$ # # #$### #. @# #####	lluUddrrUdlluurRurrdLLddll uuURlddrruuLulDDurrruLdll uluurD
13	##### # @ ### ## . # #. $.$ # ##$# ### # # #####	drdddllUURuulDrdLurrDurrdL LullddrrUUruLullddR

Tabel B33 Data Uji Coba Koleksi *Puzzle Yoshio* (5)

Level	Puzzle	Output
14	##### ## # # $# # # . @## # * # ## #$ # #. ## #####	uullDlddRddrrUUddlluuluRdd drruuLrddlluUrrUdllluururr DDuulldlddrddrruruLuuulldD DDuuuurrdddLruL
15	#### ## #### #..$.# # #$ $ # #@ # # ##### # ####	uurRllddrrUrRurDlllldlluuru rDRllddrrUruLdrrruLdddrUU llluLrdrruLL
16	###### # .@## # $.# ###*# # ## # # $ ## # ### #####	ldDDrruuLLddllddrUluRRuuul ldRurDrrddLLUdrdLLrurruulu lldRRlddldlluRuRlddrUrUUU dddlluRdrU
17	#### #@ # # # ##. #### # $$. .# # $ ### ### # ####	ddDRRRlddlUUlldRurrddlUruL UluurDDDrddlUlUR

Tabel B34 Data Uji Coba Koleksi *Puzzle Yoshio* (6)

Level	Puzzle	Output
18	##### #. # # # ### # *$ # # $. # # @### #####	luRlluuurrdDLruulldDrddrUU rrdLLullluurrDullddrRdrruLL ddllURRllUU
19	##### # # # #.# ### .# #@ $$ # # .$ # #######	drrUUrrddLruuuullDurrddddl UlldRurruuulldDRdrUdlLdllu RRurrddLruulldRdrU
20	###### # @# # $# ### # * $ # # ## # ##. . # ## ## #####	llldddrruRldlluRluurDDuurr ddLruulldlddRluuurrrddlDld RluluurDDrddrruruuLLuulldd drdRluluuurrddL
21	###### # @## # # # #. $ # # $$#.# ### .# ####	drdLLrrddllUdrruuluulldldd RUluurrrddLLrruullDurrddrd dllUlURddrruuulullDurrdrdd dlluuRurDDulldllURRRurDlll uulD

Tabel B35 Data Uji Coba Koleksi *Puzzle Yoshio* (7)

Level	Puzzle	Output
22	#### ###. # # . ### # $$ # ## . $@# #######	LLUdrruLLLulldRdRRuLdlUrrr rdLLuluurDldlluRdrUdRdrruL LullddRUrrdLuuL
23	###### ##@. # # $$* # # # ## # # .# #### # # # # #####	rrDDDuuullDRllddrUluRurrDL rrddlUdrrddllUUruuulldRurD llulDlddrUluRurrdDDrrddllU UUUrDldRluuLulDrrruLdllldd rUluRR
24	#### # # ###$.# # . # ### #.# # $ $ # # #@ # ########	ruuullddldlluRdrUruurruulD rdddLLLdlluRRRRdrUlluulDrd rrUUddllLdlluRRRRdrU
25	##### # .### # $.. # # ##$## ## # # #$ @# # #### ####	llluluUluRdddrddlUUUluurDr rrDDrdLLLulUluRRlddrddlUUU luR

Tabel B36 Data Uji Coba Koleksi *Puzzle Yoshio* (8)

Level	Puzzle	Output
26	#### # # # ### ### .. # # $# # # .$$ # #### @ # #####	ULrUdlLUlldRRRdrruLLruruLd dlLulldRRRdrUlluURuulDDDld RRdrruLLrruuLrddlU
27	##### # ### # # *@## # * # ###$ # # .# ######	dlLuullddRRRdrruLuLDllluur rDullddrRurrdddllURRurDllU
28	###### ### . # # $@#. # # $# ## # * # ## # # ## # #####	urrdddddlluluRUdddrruuLrdd lluluurDrruuullDDuurrdddlL rrddllUlUluuRlddrdrdrruuuu ullDllddrRRllluurrDullddrd rdrruUUdddlluluRlluRurDldd rdrruuuUruLddddddlluURlddrr uUllluurDlddrdrruuU
29	#### ## ### # ## # #$$@# # . *.# ########	dLLUdrruLuLLulDlddrUluRRld dRRRuuLDuLulDrrdrdLuullldd rRRuullDldR

Tabel B37 Data Uji Coba Koleksi *Puzzle Yoshio* (9)

Level	Puzzle	Output
30	####### ##@ # #. # # # $$$.## # .# # # #### ####	dDuurrdrddlUruulllddRRdrUl lluurrDullddrRllddrUluuru rrddlLdlUruurrrrdLullddrr drUllluurrDurrdLdLLrruulll dDuurrDullddrR
31	######## # # # # ##*# # #@ $ # #.$. # ##### # # # ####	drrruLdlluuulldddRRuuurrrD DlLrruullllldddrrUrrddrUUl lldRllluuurrDDrdLurrrdLull dL
32	###### #@ ## ##$ # ### . # # $ #$## # . .# #### # ####	rrddlddrrUUddlllluRdrrruul lDurrddlLuuruulDurrDrdLddl lulldRRRdrUUUruLulldDDuuur rddddlLuuRluurrdDDuuulD
33	##### # ### # $ # ##$$.# #@ . # ## # # # .# #####	rruuLulldRRDDlUruulldRurDD ldRurruLdlluurDldRurrddddl lUUURlluurDldRurrDDDuuulld R

Tabel B38 Data Uji Coba Koleksi *Puzzle Yoshio* (10)

Level	Puzzle	Output
34	##### # #### # $$ # # .#. # # ## ## # ##$# # @ .# #######	luuuuurDRRdrruLLLulldddddr rrrUUddlllluuuuurrdrrrdLul lulldRRRllDDlddrrrruuuruLL LullddrDldRRRllluuuuurD
35	###### # . # # .# ### # @$$ # # $. # ########	ldRRRULdlUrrrrdLLullluurrr DullldrrRdrruLLuullldrUd dRRurrdLulLdlUrrrdL

36	########
	# @.# #
	# .$. #
	# #$ #
	# $ ##
	### ##
	  # #
	  ####
		dRRDDLddrUUruruLLLulDrrrdd
		lUruLLullddRRdrUUdlllluuur
		rdRlulldRlddrUUddrruuLrddl
		lluuuR

37	####### ## . # # $ $@# #.$.#### # ## # # # # ####	ulldldLruulDurrrrdLulldlDu urrrdLullddrUluRRdldlDlddr UUUruulDlDRddlU

Tabel B39 Data Uji Coba Koleksi *Puzzle Yoshio* (11)

Level	Puzzle	Output
38	###### # . # # #@# # $ ## ##$# # # # # #. * # #######	dLrdrddlLrruuluuulldldRuur rdddrddllulldRuUUluurrrddL rdrddlLrruuluuulllddRUdddr drruuluLrdrddlluluUluuRldd rddldRRluuuluurDDDDrdLuuur rdrddLruuluuuL
39	##### #### . # # *@ . # # $ # # # # $ # # #### #####	dddlluuRluRRRurDlllldlddrru UluRRurrdddLLrruuulldldlld drrUUdrrruuulldLrurrdLrddl UUddlluuR
40	#### ### ### # .. $.# # $$ @# #### # #####	dllULuurDRddlUlulldRRRdrru LuLLrrddlUruL
41	#### #@ # ##### .# # $ $ $# # . # ### . # ######	drDldlluRdllluRRdrrUruulDD LdlluRddrrrUUdLuuurDDlddll uuRlddrruUruulDDllddrrrUdl lluurrDrdLurU

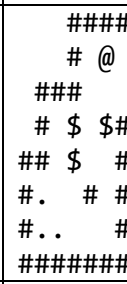
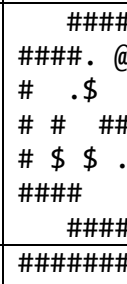
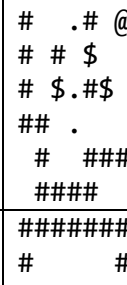
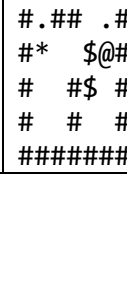
Tabel B40 Data Uji Coba Koleksi *Puzzle Yoshio* (12)

Level	Puzzle	Output

42	########
	# # #
	# #.$ $#
	# $ #
	#####. #
	  # @#
	  # .#
	  ######
		luuuurDDDDluuLLuullddRluur
		rdRdrruulDDlulullddrRurrdD
		rdLuuulldRurrddlUdddlluRdr
		UruuuLL

43	#### ##@ ## ## ..# ## $#$## # $. # # # # # ### ######	drDDLLrruullDlDRuurrddrdLu uullddRllldrrrUrUUddldlll uurrRluurRlldd lldrUluRRld drrurUUddldlluurR
44	###### # @# # $$#### # $. # ## #.# # #. # # # # ########	lDulldRDRlDurRdddrruuuLrdd dllllluRRdrUdrruuulLLrrrdd dllllluUluurDDDurrrrdddlll uLuuuurDldR
45	#### # # #### $## # @$. # # ## # # ## # # * .# ########	RRdrruLLuurDDrdddlLLulluur RllddRRlluuurrRdRUrDDullu uurDlddrruL

Tabel B41 Data Uji Coba Koleksi *Puzzle Yoshio* (13)

Level	Puzzle	Output
46		dlddrUluurrdLulDDrdddllluu uRDlddrrruuuLrdLrddllluuuR lddddrrruulLrruulDllDDuurrr dLLulDrrrddllLrrruullDLurr rddlL
47		llDlddRUlullddRluurrdrddrr uLdlULuullddRRRluuRurrdLLL ddrRlUdllluuRlddrrurU
48		dlDrdLLLuuRlddrruruulDllul lddRluurrdrrrddlUdlldlUrrr uruulDllDuullddRdRUUrrrddl Lrruullldlddr Urrruul DrdLL
49		LLrrddlUUruullllldDRRRllluu rrrrdDlllddlUUrrrruulDrdLd drUluLLrrdrU

Tabel B42 Data Uji Coba Koleksi *Puzzle Yoshio* (14)

Level	Puzzle	Output
50	#### #. # # $# # ##### # .$ @ # # .$ # # ### # ######	lLrddlUlluurDldRRdrrruullL ddrUluRdlllluuuurDldddrruLd lUUUrDDrddrrruuLLL
51	######## # # # #$ # # $ @#.# ##$#. # # .# ########	uullddRRlluurrrdDurrdddll llUdrrruruullullddRluurrr ddDuuullddrRlluurrrdDlldd rRRuruullulDrrrddldlUUddll uuRuRDuRurD
52	###### # . # # ### # #$$. # #. ## # #@$ ## # ### # ######	urrUUdRuuLdddddrrruuuLLull lddrrUdlluurRddlldRururrrd ddlllUlluuurrurDllluRdrdRl ullddrrUUruLLulDD

Berikut data uji coba untuk koleksi *puzzle hard case* ditunjukkan oleh tabel B43 hingga tabel B45.

Tabel B43 Data Uji Coba Koleksi *Puzzle Hard Case* (1)

Level	Puzzle	Output
1	######## #. # #$ $ $ # # $ $ # #@ $ $ # ###*** # #......# ########	uUrurrdRurDDullLulDrrddLru urrdDDuuullddDuuurrdddLruu ullllldRUluRRurrdrdddldLL LrrruruuulLullddRDuluuurr drrdddldlLuuluuurrdLulDDur DDuurrrdLDDuuullddRDulDuuu rrdDuulDDlulldRurD
2	######## ##.* $.# # * . # # *. # ## $.$ # # $*$$$# #.. @# ########	*<Tidak dapat diselesaikan>*
3	######## #......# # # #***** # # # #$$$$$$# # @# ########	UUUUddlUUddrddlUUllUUddrrd dllUUddrUruruulllluRdrrrdd dlllulUUddlUdrrUUdddrruLdd llUdlUrrUrrruullldlUrrrrr dddllllUdlU

Tabel B44 Data Uji Coba Koleksi *Puzzle Hard Case* (2)

Level	Puzzle	Output
4	######## #..... # # ## # # * # # # # # #$$$$$$# # +# ########	llllUUrUrrrrdDuullllldlddr rrUdllluuruUluRRRlldddlddr UUddrruLdlluurUrrrrddLLdll Udrrurruullll UluRRlddrrrrd dldlllluUddrruLdlluurUdlddr Urrrruullll UluRddrrrrddllU drruulLddllUUUdddrruuLrddl lluUU
5	######## #. ....# ##### # #.@$ $# #$ $ $ # # $ $ # #.$.# ########	ddlUruRRdddlLrrrruuUdddlll luRdrrruuLDLdRllluRRdrUruu LuurDldDrdLdllluuurRllddd r rruuUUruLLLLrrrdddrUddldll uUluRRldRlddrUrruuLDuuurDl ddrdLdlluuuRlddd rruuUUruLL rddlldRlddrruUlldRurUdlddr UluurUruLddlddrUrUdlUrUdlU
6	######## #. . ..# ##### # #.@$ $# #$ $ $ # # $ $ # #.$ ..# ########	ddlUruRRdddlLrrrruuUdddlll luRdrrruuLDLdlluRRRurDllll uurDldRuRlddrUrruuLDuuurDl ddLdlluurRdrUUddldlUluRdrr rdLullddrUruuLrddlluRururD lldlluRddrrUUruuLLLLrrrrdd lUdddrUluuruLLrdddlluRdrrU dlUrUdlU
7	######## #......# #. $@# #. $ $# #$ $ $ # # $$ $ # #.$.# ########	ullddrUdlddd lLrruuuurruLdd lLrdddrruuUUddddllllUlUUdd rdrruuulLrrdddlluUUUruLddr rdLrddlluUUUddddrUUdrRluur rdDlUUdllUrrUdllU

Tabel B45 Data Uji Coba Koleksi *Puzzle Hard Case* (3)

Level	Puzzle	Output
8	######## #......# #. $ # #.@$ $# #$ $ $ # # $$ $ # #.$.# ########	urrdrUdlddd lLrruuuurruLddl LrdddrruuUUddddllllUlUUddr drruuulLrrdddlluUUUruLddrr dLrddlluUUUddddrUUdrRluurr dDlUUdllUrrUdllU
9	######## #. # #$ $ $ # #@ $ $ # # $ $ # # ****# #......# ########	UrurrdddLruuLulDurrrrdLull dDLulDDDuurDDuuuurrrdrddDu uulullddrRuulDrrDDulLdDuuu llddRluurrdDurrdLDuulullddd rDRlluuurrdDuulDD

(Halaman ini sengaja dikosongkan)

BIODATA PENULIS

Aditya Pratama, lahir di Padang pada tanggal 29 Desember 1998. Penulis menempuh pendidikan mulai dari TK LKMD Pekanbaru (2003-2004), SD 015 Senapelan (2004-2007), SD 163 Pekanbaru (2007-2010), SMP Negeri 4 Pekanbaru (2010-2012), SMA Negeri 8 Pekanbaru (2012-2015), dan sekarang sedang menjalani pendidikan S1 Informatika di ITS. Penulis aktif dalam organisasi dan kepanitiaan Himpunan Mahasiswa Teknik Computer (HMTC) dan Schematics. Diantaranya adalah menjadi staff Departemen Kaderisasi dan Pemetaan Mahasiswa (KDPM) HMTC ITS 2016-2017, staff *National Programming Contest* (NPC) Schematics ITS 2016 dan badan pengurus harian *National Programming Contest* (NPC) Schematics ITS 2017. Penulis juga merupakan salah satu penerima beasiswa dari Dinas Pendidikan Provinsi Riau selama menempuh pendidikan S1 di ITS. Komunikasi dengan penulis dapat melalui telepon: +6285363547381 dan *email*: **aditya.vritalia@gmail.com**.