TUGAS AKHIR – IF184802

DESAIN DAN ANALISIS *COMPLETE SEARCH ALGORITHM* DENGAN OPTIMASI *PRUNING* BERBASIS *BITWISE* PADA STUDI KASUS SPOJ HOUSEBUY: HOUSE BUYING OPTIMIZATIONS

ALVIN TANUWIJAYA
05111640000021

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

DESAIN DAN ANALISIS *COMPLETE SEARCH ALGORITHM* DENGAN OPTIMASI *PRUNING* BERBASIS *BITWISE* PADA STUDI KASUS SPOJ HOUSEBUY: HOUSE BUYING OPTIMIZATIONS

ALVIN TANUWIJAYA
05111640000021

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*]

FINAL PROJECT – IF184802

DESIGN AND ANALYSIS OF THE COMPLETE SEARCH ALGORITHM WITH BITWISE-BASED PRUNING OPTIMIZATION TO SOLVE SPOJ HOUSEBUY: HOUSE BUYING OPTIMIZATIONS

ALVIN TANUWIJAYA
05111640000021

Supervisor I:
Rully Soelaiman, S.Kom., M.Kom.

Supervisor II:
M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil.

DEPARTMENT OF INFORMATICS
Faculty of Information and Communication Technology
Institut Teknologi Sepuluh Nopember
Surabaya
2020

Surabaya, 2020

[*Halaman ini sengaja dikosongkan*]

LEMBAR PENGESAHAN

DESAIN DAN ANALISIS *COMPLETE SEARCH ALGORITHM* DENGAN OPTIMASI *PRUNING* BERBASIS *BITWISE* PADA STUDI KASUS SPOJ HOUSEBUY: HOUSE BUYING OPTIMIZATIONS

TUGAS AKHIR

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

Oleh:
Alvin Tanuwijaya
NRP: 051116 40000 021

Disetujui oleh Dosen Pembimbing Tugas Akhir:

Rully Soelaiman, S.Kom., M.Kom.
NIP. 197002131994021001 ..... (Pembimbing 1)

M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil.
NIP. 197402092002121001 ..... (Pembimbing 2)

KEMENTERIAN PENDIDIKAN DAN KEBUDAYAAN INSTITUT TEKNOLOGI SEPULUH NOPEMBER FAKULTAS TEKNOLOGI ELEKTRO DAN INFORMATIKA CERDAS DEPARTEMEN TEKNIK INFORMATIKA

SURABAYA
2020

[*Halaman ini sengaja dikosongkan*]

DESAIN DAN ANALISIS *COMPLETE SEARCH ALGORITHM* DENGAN OPTIMASI *PRUNING* BERBASIS *BITWISE* PADA STUDI KASUS SPOJ HOUSEBUY: HOUSE BUYING OPTIMIZATIONS

Nama Mahasiswa : Alvin Tanuwijaya
NRP : 051116 40000 021
Departemen : Informatika FTIK – 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

Tugas Akhir ini berdasarkan permasalahan pada SPOJ (Sphere Online Judge) yaitu House Buying Optimizations (HOUSEBUY), yaitu pencarian kombinasi rumah terjauh dari seluruh kemungkinan kombinasi rumah terdekat dan jarak terjauh yang dapat dibentuk dari seluruh kombinasi pasangan rumah.

Dibutuhkan metode penyelesaian yang cepat dan tepat untuk memenuhi persyaratan dalam menyelesaikan permasalahan HOUSEBUY. Batasan utamanya adalah waktu keseluruhan pencarian maksimal adalah 8 detik untuk mencari semua kombinasi penyelesaian masalahnya.

Pada Tugas Akhir ini diimplementasikan solusi dari permasalahan HOUSEBUY dengan menggunakan algoritma complete search dengan optimasi pruning berbasis bitwise. Solusi tersebut mampu mengurangi jumlah pencarian sehingga mampu memenuhi batasan yang ditentukan.

Kata Kunci: Complete Search, Pruning, Bitwise.

[*Halaman ini sengaja dikosongkan*]

DESIGN AND ANALYSIS OF THE COMPLETE SEARCH ALGORITHM WITH BITWISE-BASED PRUNING OPTIMIZATION TO SOLVE SPOJ HOUSEBUY: HOUSE BUYING OPTIMIZATIONS

Student Name	: Alvin Tanuwijaya
Registration Number	: 051116 40000 021
Department	: Informatics Department, Faculty of IT – : ITS
First Supervisor	: Rully Soelaiman, S.Kom., M.Kom.
Second Supervisor	: M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil.

ABSTRACT

This thesis based on the SPOJ (Sphere Online Judge): House Buying Optimizations problem (HOUSEBUY), i.e., finding a solution of longest distance of all possible closest distance of a pair of a house and the longest distance a pair of a house can be made.

The fast and proper solution has been needed to fulfil the requirement in solving HOUSEBUY. The main constraint is that the maximum total searching is 8 seconds to find all of the combinations to solve the problem.

The solution of HOUSEBUY will be implemented in this thesis by using the complete search algorithm with bitwise-based pruning optimization. This solution has been able to reduce the amount of search needed so that it has complied the time limit given.

Keyword: Complete Search, Pruning, Bitwise.

[*Halaman ini sengaja dikosongkan*]

KATA PENGANTAR

Puji syukur penulis ucapkan kepada Tuhan Yang Maha Esa atas pimpinan, penyertaan, dan karunia-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul:

DESAIN DAN ANALISIS COMPLETE SEARCH ALGORITHM DENGAN OPTIMASI PRUNING BERBASIS BITWISE PADA STUDI KASUS SPOJ HOUSEBUY: HOUSE BUYING OPTIMIZATIONS

Pengerjaan Tugas Akhir ini dilakukan untuk memenuhi salah satu syarat meraih gelar Sarjana di Departemen Informatika, Fakultas Teknologi Informasi dan Komunikasi, Institut Teknologi Sepuluh Nopember.

Dengan selesainya Tugas Akhir ini diharapkan apa yang telah dikerjakan penulis dapat memeberikan manfaat bagi perkembangan ilmu pengetahuan terutama di bidang teknologi informasi serta bagi diri penulis sendiri selaku peneliti.

Penulis mengucapkan terima kasih kepada semua pihak yang telah memberikan dukungan baik secara langsung maupun tidak langsung selama penulis mengerjakan Tugas Akhir maupun selama menempuh masa studi antara lain:

1. Terimakasih kepada Tuhan Yang Maha Esa, di mana penulis masih diberi kesempatan, kesehatan dan umur untuk menempuh kuliah di sini dan menjalani hidup dengan baik.
2. Ayah, Ibu, dan Kakak penulis yang selalu memberikan perhatian, bantuan, dan kasih sayang beserta seluruh tunjangan dalam berbagai hal selama menempuh perkuliahan maupun pengerjaan Tugas Akhir ini.
3. Bapak Rully Soelaiman, S.Kom., M.Kom. selaku Dosen Pembimbing yang telah membimbing saya tidak hanya dalam pengerjaan Tugas Akhir tetapi juga selama perkuliahan dari tahun pertama hingga terakhir. Terima kasih atas seluruh perhatian, kasih sayang, dan juga kesabarannya dalam menghadapi mahasiswa seperti saya.

4. Bapak M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil. selaku dosen pembimbing yang telah memberikan perhatian, didikan, pengajaran, tuntunan, dan ceritanya. Berkat beliau buku TA ini dapat diselesaikan dengan baik.
5. Ferdinand, Jonathan, Indra, Nurlita, Yolanda, Dandy, Edwin, dan seluruh teman-teman angkatan 2016 yang telah menemani perkuliahan sejak masuk di ITS.
6. Mas Pras, Daniel, Ka Adrian, Irene, Icon, Mayshel, Reni dan teman-teman organisasi kerohanian yang juga selalu membei semangat dan pelajaran selama saya di ITS.

Penulis mohon maaf apabila masih ada kekurangan pada Tugas Akhir ini. Penulis juga mengharapkan kritik dan saran yang membangun untuk pembelajaran dan perbaikan di kemudian hari. Semoga melalui Tugas Akhir ini penulis dapat memberikan kontribusi dan manfaat yang sebaik-baiknya.

Surabaya, 19 Desember 2019

Alvin Tanuwijaya

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

[*Halaman ini sengaja dikosongkan*]

DAFTAR KODE SUMBER

[*Halaman ini sengaja dikosongkan*]

BAB I
PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar tugas akhir yang meliputi latar belakang, tujuan, rumusan masalah, batasan permasalahan, metodologi pembuatan tugas akhir, dan sistematika penulisan buku tugas akhir ini.

1.1 Latar Belakang

Latar belakang dari tugas akhir ini adalah adanya permasalahan *House Buying Optimizations* pada SPOJ (*Sphere Online Judge*), yang selanjutnya disebut HOUSEBUY. Permasalahan tersebut menggambarkan sebuah peta yang kemudian direpresentasikan dalam bentuk graf untuk mencari jarak terjauh dari dua titik diantara kombinasi jarak (*cost*) titik-titik yang membentuk graf dimana informasi yang diberikan untuk membentuk graf tersebut tidak lengkap dan dapat terjadi redundansi.

Ketidaklengkapan dan redundansi informasi dalam pembentukan graf menimbulkan masalah yaitu banyaknya kemungkinan pembentukan graf dikarenakan setiap informasi yang diberikan akan menyebabkan adanya percabangan baru pada graf yang dibentuk. Selain itu, masih ada masalah utama yang muncul, yaitu bagaimana mendapatkan jarak (*cost*) terjauh dari semua kemungkinan graf yang terbentuk.

Hasil dari tugas akhir ini adalah implementasi solusi pencarian jarak terjauh dari berbagai kemungkinan graf dengan waktu yang cepat. Solusi tersebut mempunyai kompleksitas waktu yang optimal untuk setiap *query* pengambilan jarak (*cost*) terjauh dari setiap graf.

1.2 Rumusan Permasalahan

Rumusan masalah yang diangkat dalam tugas akhir ini adalah sebagai berikut:

1. Bagaimana kinerja metode *Pruning* yang berbasis *Bitwise* diimplementasikan pada saat menyelesaikan permasalahan penelusuran berupa pencarian jarak dari berbagai kombinasi anggota graf pada HOUSEBUY?
2. Bagaimana uji coba untuk mengetahui kebenaran dan kinerja dari implementasi yang dilakukan?

1.3 Batasan Permasalahan

Ada dua topik bahasan dalam tugas akhir ini, yaitu batasan tugas akhir dan batasan SPOJ. Permasalahan yang dibahas pada Tugas Akhir ini memiliki beberapa batasan, yaitu sebagai berikut:

1. Permasalahan yang dibahas pada pencarian jarak dari seluruh kombinasi rumah terjauh yang berbentuk graf.
2. Metode yang digunakan adalah *Complete Search* dengan *Pruning* berbasis *Bitwise*.

Sementara batasan SPOJ adalah sebagai berikut:

1. Implementasi dilakukan dengan menggunakan bahasa pemrograman C++.
2. Batas nilai *m* adalah 2 sampai 10.
3. Batas nilai *n* adalah 2 sampai 10.
4. Jumlah maksimal gedung dalam permasalahan adalah 25 gedung berbeda.
5. Jumlah lokasi maksimal yang diberikan adalah 50 untuk setiap kasus.
6. Batas waktu yang diberikan adalah 8 detik.
7. Batasan memori yang diberikan adalah 50000 B.
8. Batasan memori yang diberikan adalah 1536 MB

1.4 Tujuan Pembuatan Tugas Akhir

Melakukan optimasi kinerja pada penelusuran anggota graf untuk jarak (*cost*) terjauh pada sebuah *tree*.

1.5 Manfaat Tugas Akhir

Manfaat tugas akhir ini adalah membantu memahami penggunaan *Pruning* dalam menyelesaikan permasalahan penelusuran anggota graf secara efektif dan efisien.

1.6 Metodologi

Langkah-langkah yang ditempuh dalam pengerjaan tugas akhir ini dapat dijelaskan seperti di bawah ini.

1.6.1 Penyusunan Proposal Tugas Akhir

Tahap awal untuk memulai pengerjaan tugas akhir adalah penyusunan proposal tugas akhir. Pada proposal ini, penulis mengajukan gagasan untuk menyelesaikan permasalahan pada studi kasus SPOJ klasik HOUSEBUY dengan *Complete Search algorithm* dengan *Pruning* berbasis *Bitwise*.

1.6.2 Studi Literatur

Pada tahap ini dilakukan pencarian informasi dan studil iteratur yang relevan untuk dijadikan referensi dalam melakukan pengerjaan tugas akhir. Informasi didapatkan dari materi-materi yang berhubungan dengan struktur data. Informasi tersebut didapatkan dari buku, internet, dan materi kuliah yang berhubungan dengan metode yang akan digunakan.

1.6.3 Implementasi Perangkat Lunak

Tahapan ini merupakan tahapan untuk membangun algoritma yang akan digunakan. Pada tahap ini dilakukan implementasi dari rancangan struktur data yang akan dimodelkan

sesuai dengan permasalahan. Implementasi ini dilakukan dengan menggunakan bahasa pemrograman C++.

1.6.4 Pengujian dan Evaluasi

Pada tahap ini dilakukan uji coba kebenaran dan kinerja solusi yang telah diimplementasikan dengan melakukan pengiriman sumber kode sistem ke situs penilaian *Sphere Online Judge(SPOJ)* pada permasalahan yang terkait untuk mendapatkan umpan balik. Pengujian dilakukan dengan membandingkan kompleksitas hasil uji coba dengan kompleksitas hasil analisis.

1.6.5 Penyusunan Buku Tugas Akhir

Pada tahap ini dilakukan penyusunan laporan yang menjelaskan dasar teori dan metode yang digunakan dalam tugas akhir ini serta hasil dari implementasi aplikasi perangkat lunak yang telah dibuat.

1.7 Sistematika Penulisan

Buku tugas akhir ini merupakan laporan secara lengkap mengenai tugas akhir yang telah dikerjakan baik dari sisi teori, rancangan, maupun implementasi sehingga memudahkan bagi pembaca dan juga pihak yang ingin mengembangkan lebih lanjut. Sistematika penulisan buku tugas akhir secara garis besar dapat dilihat seperti dibawah ini.

Bab I Pendahuluan

Bab ini berisi penjelasan latar belakang, rumusan masalah, batasan masalah dan tujuan pembuatan tugas akhir. Selain itu, metodologi pengerjaan dan sistematika

penulisan laporan tugas akhir juga dijelaskan di dalamnya.

Bab II Dasar Teori

Bab ini berisi penjelasan secara detil mengenai dasar-dasar penunjang dan teori-teori yang digunakan untuk mendukung pembuatan tugas akhir ini.

Bab III Analisis dan Perancangan Sistem

Bab ini berisi penjelasan tentang rancangan dari sistem yang akan dibangun.

Bab IV Implementasi

Bab ini berisi penjelasan implementasi dari rancangan yang telah dibuat pada bab III. Implementasi disajikan dalam bentuk *pseudocode* disertai dengan penjelasannya.

Bab V Pengujian dan Evaluasi

Bab ini berisi penjelasan mengenai data hasil percobaan dan pembahasan mengenai hasil percobaan yang telah dilakukan.

Bab VI Kesimpulan dan Saran

Bab ini merupakan bab terakhir yang menjelaskan kesimpulan dari hasil uji coba yang dilakukan dan saran untuk pengembangan perangkat lunak ke depannya.

[*Halaman ini sengaja dikosongkan*]

BAB II
DASAR TEORI

Pada bab ini akan dijelaskan mengenai dasar-dasar teori yang akan digunakan guna menyelesaikan permasalahan HOUSEBUY. Dasar teori yang digunakan meliputi pencarian dengan metode *complete search*, Operasi *Bitwise*, metode *pruning* yang digunakan, beserta penjabaran mengenai teori yang digunakan sebagai solusi pada permasalahan.

2.1 Deskripsi Permasalahan SPOJ HOUSEBUY

HOUSEBUY [1] – *House Buying Optimizations*, merupakan sebuah permasalahan yang diunggah pada situs penilaian *online Sphere Online Judge* (SPOJ) dengan deskripsi permasalahan yang diterjemahkan ke dalam Bahasa Indonesia sebagai berikut:

"Bill dan Scott merupakan rival bisnis. Setiap dari mereka ingin membeli sebuah rumah di Javaville, tetapi mereka ingin tinggal sejauh mungkin antara yang satu dengan yang lain. Karena Javaville merupakan kota yang baru, maka belum ada peta yang tersedia. Namun, informasi tentang rumah dan bangunan lainnya telah dikumpulkan dari mulut ke mulut dan disediakan kepada Bill dan Scott. Informasi ini terdiri atas alamat bangunan dan jarak antar bangunan. Seluruh informasi adalah konsisten, meskipun bisa saja tidak lengkap atau terulang.

Jalanan di Javaville tertata dalam bentuk kotak segi empat dengan **m** merupakan jalan timur/barat (bernama 'A', 'B', 'C', ...) dan **n** merupakan jalan utara/selatan (bernomor '0', '1', '2', ...), dimana masing-masing **m** dan **n** bernilai antara 2 dan 10. Setiap bangunan (rumah atau bangunan lain, seperti kantor pos atau sekolah) di Javaville berada pada perpotongan dua jalan, dan tidak ada bangunan yang terletak pada perpotongan yang sama. Beberapa perpotongan tidak terdapat bangunan sama sekali. Seluruh jarak diukur dari angka terkecil dari seluruh blok yang

harus dilalui utara, selatam, timur, dan/atau barat untuk menuju dari perpotongan yang satu ke perpotongan lainnya.

Berikut merupakan contoh informasi yang disediakan kepada Bill dan Scott oleh berbagai sumber terpercaya:

- Terdapat 5 jalan timur/barat dan 5 jalan utara/selatan
- House1 terletak pada perpotongan A0
- Kantor Pos terletak pada perpotongan A4
- Sekolah berjarak 4 blok dari house1
- House2 berjarak 6 blok dari kantor pos
- Sekolah berjarak 6 blok dari kantor pos
- House3 berjarak 6 blok dari kantor pos

Dari Gambar 2.1 berikut dapat dilihat bahwa ada dua kemungkinan peta dari Javavile

	0	1	2	3	4
A	H1				PO
B					
C	H2				
D		S			
E			H3		

	0	1	2	3	4
A	H1				PO
B					
C	H3				
D		S			
E			H2		

Gambar 2.1 Tata letak kota

Bill dan Scott ingin anda untuk membuat sebuah program yang akan menerima informasi lokasi dan jarak dari bangunan dan menentukan dua kuantitas, **D** dan **D'**. **D** adalah minimum, dari seluruh kemungkinan rancangan bangunan yang sesuai, dari jarak rumah terjauh dalam setiap rancangan. **D'** adalah nilai maksimum dimana terdapat paling tidak sepasang rumah **i** dan **j** yang dipastikan terpisah dengan jarak setidaknya **D'**. Pada kasus terakhir, anda dimohon untuk mencatatkan seluruh pasangan rumah yang dipastikan terpisah dengan jarak setidaknya **D'** blok.

Untuk lebih tepatnya, tentukan diameter **d(S)** dari tata kota **S** sebagai jarak antara dua rumah huni terjauh dalam tata kota

tersebut. Untuk kedua rumah huni **i**, **j**, tentukan faktor keamanan mereka: **e[i, j]** yang merupakan nilai minimum dari jarak rumah huni **i** dan **j** pada tata kota yang layak. Bill dan Scott ingin anda untuk membuat program yang menghitung **D** dan **D'** berdasarkan informasi yang mereka dapatkan, dimana

D = min {d (S)} dari seluruh tata letak S;

D' = max {e [i, j]} dari seluruh pasangan (i, j).

Pada saat yang sama anda juga harus memberikan rekomendasi pembelian terbaik: dimana seluruh pasangan **(i, j)** memenuhi **e[i, j] = D'** untuk setiap rumah huni **i** dan **j**."

Pada permasalahan HOUSEBUY diceritakan bahwa Scott dan Bill merupakan rival bisnis yang ingin membeli rumah di kota yang sama dengan jarak antar rumah sejauh mungkin. Diketahui pada soal bahwa tata letak kota berbentuk petak segiempat dengan setiap bangunan berada pada perpotongan jalan. Mereka ingin agar mereka dapat menemukan seluruh kombinasi rumah terjauh dari seluruh kemungkinan tata letak kota yang dapat dibuat.

Pada permasalahan ini, terdapat batasan permasalahan antara lain,

1. Batas nilai m adalah 2 sampai 10.
2. Batas nilai n adalah 2 sampai 10.
3. Jumlah maksimal gedung dalam permasalahan adalah 25 gedung berbeda.
4. Jumlah maksimal lokasi yang diberitahukan adalah 50 untuk setiap kasus.
5. Batas waktu yang diberikan adalah 8 detik.
6. Batasan ukuran program adalah 50000 B.
7. Batasan memori adalah 1536 MB.

Masukan dari permasalahan ini adalah jumlah jalan timur/barat (**m**) dan jumlah jalan utara/selatan (**n**). Setelah itu masukan berikutnya adalah salah satu diantara dua format berikut:

- nama LOCATION r c

- nama DISTANCE d nama2

Dimana nama dan nama2 merupakan *string* berisi angka dan alfabet huruf kecil, dengan panjang maksimal 10. r adalah huruf kapital antara ‘A’ dan ‘J’. c adalah angka dan d adalah bilangan bulat positif. Berikut merupakan contoh masukan pada permasalahan ini yang dapat dilihat pada Gambar 2.2.

```
Input:
5 5
house1 LOCATION A 0
postoffice LOCATION A 4
school DISTANCE 4 house1
house2 DISTANCE 6 postoffice
school DISTANCE 6 postoffice
house3 DISTANCE 6 postoffice
END
```

Gambar 2.2 Contoh masukan

Dari masukan pada Gambar 2.2 akan menghasilkan keluaran berupa **D**, **D’**, dan pasangan rumah yang memenuhi persyaratan. Keluaran dari contoh yang diberikan dapat dilihat pada Gambar 2.3,

```
Output:
6 4
house2 house3
```

Gambar 2.3 Contoh keluaran

Pada contoh masukan pada Gambar 2.2 diketahui bahwa terdapat bentuk dari kotta tersebut adalah persegi berukuran 5x5 dan terdapat 5 bangunan (house1, house2, house3, *postoffice*, *school*) dengan keterangan informasi yang berbeda-beda. Permasalahan ini dimodelkan dengan menggunakan koordinat kartesius. Masukan 5 5 merupakan m = 5 dan n = 5 sehingga direpresentasikan seperti Gambar 2.4,

	0	1	2	3	4
A					
B					
C					
D					
E					

Gambar 2.4 Representasi koordinat kartesius 5x5

Setelah itu, masukan selanjutnya adalah ‘house1 LOCATION A 0’ dan ‘postoffice LOCATION A 4’, kedua bangunan tersebut berada pada perpotongan yang berbeda dan dapat dipastikan bahwa perpotongan A0 dan A4 hanya ditempati oleh kedua bangunan tersebut. Sehingga representasi selanjutnya dapat dilihat seperti pada Gambar 2.5,

	0	1	2	3	4
A	H1				PO
B					
C					
D					
E					

Gambar 2.5 Representasi lokasi house1 dan postoffice

Setelah kedua bangunan tersebut telah diletakkan, dilanjutkan dengan masukan berikutnya yaitu ‘school DISTANCE 4 house1’, ‘house2 DISTANCE 6 postoffice’, ‘school DISTANCE 6 postoffice’, ‘house3 DISTANCE 6 postoffice’ dengan representasi kemungkinan lokasi setiap bangunan terlepas dari hubungannya dengan bangunan lain yang belum memiliki lokasi pasti seperti pada Gambar 2.6, Gambar 2.7, dan Gambar 2.8.

	0	1	2	3	4
A	H1				PO
B				S	
C			S		
D		S			
E	S				

Gambar 2.6 Representasi kemungkinan lokasi *school*

	0	1	2	3	4
A	H1				PO
B					
C	H2				
D		H2			
E			H2		

Gambar 2.7 Representasi kemungkinan lokasi house2

	0	1	2	3	4
A	H1				PO
B					
C	H3				
D		H3			
E			H3		

Gambar 2.8 Represenasi kemungkinan lokasi house3

Setelah seluruh kemungkinan diletakkan, dilakukan eliminasi sehingga menyisakan kemungkinan tata letak yang sah (Gambar 2.9 dan Gambar 2.10).

	0	1	2	3	4
A	H1				PO
B					
C	H2				
D		S			
E			H3		

Gambar 2.9 Representasi tata letak 1

	0	1	2	3	4
A	H1				PO
B					
C	H3				
D		S			
E			H2		

Gambar 2.10 Representasi tata letak 2

Masalah utama dalam penyelesaian HOUSEBUY adalah waktu komputasi untuk mencari seluruh kemungkinan lokasi yang sah dari berbagai masukan. Hal ini dikarenakan, pencarian *complete search* mencari seluruh kemungkinan yang dapat dibentuk, sehingga apabila tidak dioptimasi akan menjadi tidak efisien. *Pruning* berbasis *bitwise* dipilih karena *pruning* berfungsi untuk mengurangi kemungkinan pencarian yang tersedia dengan memotong/menghilangkan kemungkinan yang tidak sah dan operasi *bitwise* memiliki waktu komputasi yang lebih cepat daripada komputasi lainnya.

2.2 Graf

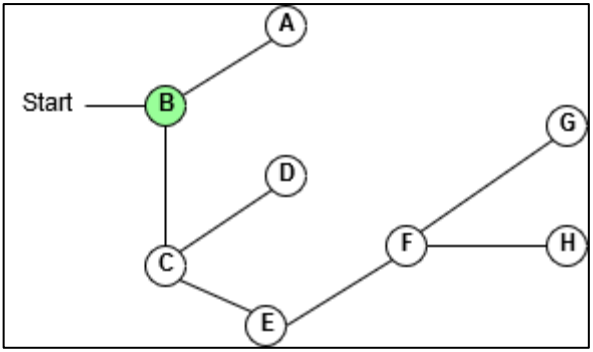
Graf adalah sebuah himpunan dari objek-objek bernama titik, simpul, dan sudut (*node*) yang dihubungan oleh penghubung dengan nama garis (*edge*) sehingga setiap titik memiliki tetangga (*neighbour*) yaitu titik yang terhubung dengan titik tersebut. Terdapat 2 jenis graf, yaitu graf berarah dan graf tidak berarah. Graf berarah merupakan graf dengan garis penghubung yang

memiliki arah dari titik A ke titik B, sedangkan untuk garis penghubung graf tidak berarah dari titik A ke B dan titik B ke A dianggap sama. Dalam penyelesaian permasalahan HOUSEBUY, graf digunakan sebagai struktur data yang akan merepresentasikan solusi dari permasalahan tersebut.

2.3 Algoritma Complete Search

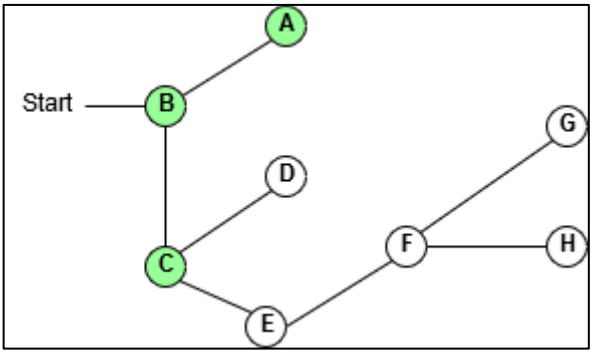
Complete Search atau pencarian menyeluruh merupakan pencarian dimana apabila terdapat jalan untuk mencapai hasil yang diinginkan, maka pencarian akan dilakukan hingga menemukan hasil tersebut. Namun, *Complete Search* tidak mengutamakan hasil pencarian atau langkah yang optimal. Contoh dari *Complete Search Algorithm* adalah *Breadth-First Search* (BFS) yang berfungsi untuk mencari hasil yang dituju dengan cara melintasi seluruh jalan yang dapat dilintasi hingga mencapai tujuan.

Penelusuran *Breadth-First Search* dimulai dari titik awal dilanjutkan dengan penelusuran berdasarkan tingkat (*layer*/*level*) selanjutnya ke titik yang menjadi tetangga dari titik sebelumnya. Berikut merupakan contoh penelusuran *Breadth-First Search*.



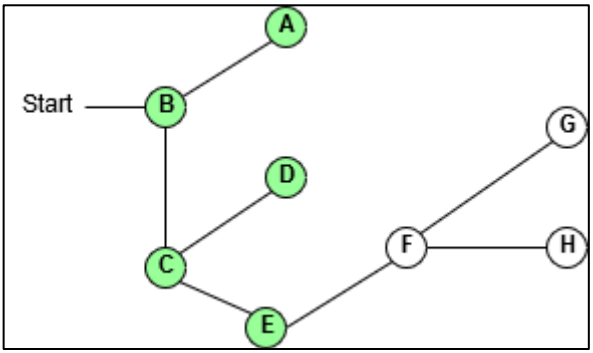
Gambar 2.11 Penelusuran BFS titik B

Setelah menelusuri dari *start* ke B (Gambar 2.11), tetangga yang berada pada tingkat selanjutnya adalah A dan C. Sehingga penelusuran dilanjutkan ke kedua titik tersebut (Gambar 2.12)

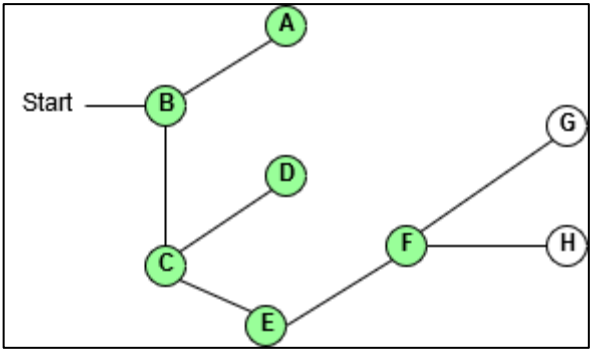


Gambar 2.12 Penelusuran BFS titik A dan C

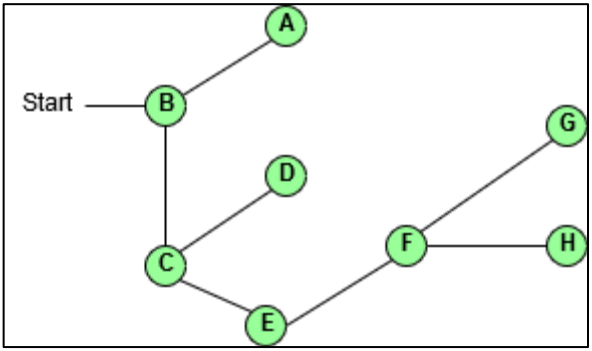
Tahapan penelusuran dilakukan secara terus menerus hingga tiba pada titik terujung (*leaf node*) dengan penelusuran selanjutnya adalah titik D dan E. Dilanjutkan dengan penelusuran ke F dan penelusuran terakhir berhenti pada titik G dan H sebagai *leaf* nodes. Tahapan-tahapan tersebut digambarkan pada Gambar 2.13, Gambar 2.14, dan Gambar 2.15.



Gambar 2.13 Penelusuran BFS titik D dan E



Gambar 2.14 Penelusuran BFS titik F



Gambar 2.15 Penelusuran BFS titik G dan H

2.4 Operasi Bitwise OR

OR (|) merupakan salah satu operator *bitwise* yang digunakan untuk *Bit Manipulation* yang mengeksekusi bilangan logika bernilai 0 dan 1, dimana hasil dari operasi akan mengeluarkan nilai *False* (0) apabila kedua masukan bernilai 0, dan menghasilkan *True* (1) apabila terdapat nilai 1 pada salah satu masukan. Dapat dilihat pada Tabel 2.1

Masukan		Keluaran
0	0	0
0	1	1
1	0	1
1	1	1

Tabel 2.1 Tabel Kebenaran OR

2.5 Operasi Bitwise AND

AND (&) merupakan salah satu operator *bitwise* yang digunakan untuk *Bit Manipulation* yang mengeksekusi bilangan logika bernilai 0 dan 1, dimana hasil dari eksekusi akan mengeluarkan nilai *False* (0) apabila terdapat masukan bernilai 0, dan menghasilkan *True* (1) apabila seluruh masukan bernilai 1. Dapat dilihat pada Tabel 2.2.

Masukan		Keluaran
0	0	0
0	1	0
1	0	0
1	1	1

Tabel 2.2 Tabel Kebenaran AND

2.6 Pruning

Pruning merupakan sebuah teknik yang berguna untuk memperkecil ukuran dari sebuah *graf* dengan memangkas bagian dari *graf* yang tidak memiliki dampak pada hasil yang dituju. Pada permasalahan HOUSEBUY, batasan utama yang harus dilampaui adalah batasan waktu, sehingga membutuhkan metode untuk mempersingkat waktu pencarian. Teknik *pruning* yang digunakan dalam tugas akhir ini akan berbasis pada operasi *bitwise* selama proses pencarian hasil.

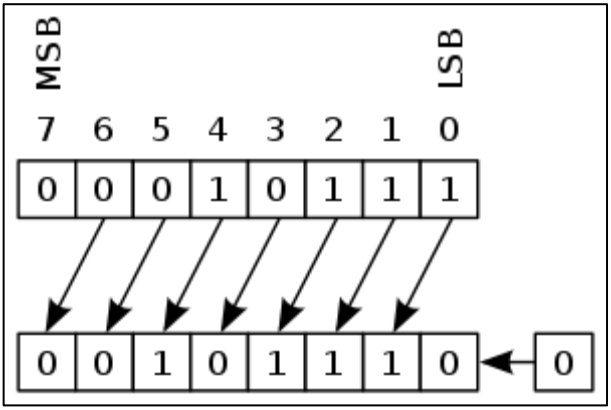
Metode *pruning* yang digunakan untuk menyelesaikan permasalahan ini berpusat pada pengurangan jumlah titik yang

akan dilalui sehingga pada saat pencarian tidak perlu untuk mencari hingga seluruh titik [2]. Pruning akan dilakukan dengan mengecek apakah masih ada kemungkinan titik yang dapat dilalui dengan suatu jarak tertentu.

Sebagai contoh masukan, yaitu: angka 2 dan angka 8, ketika dilakukan operasi AND akan menghasilkan angka 0, hal ini karena perhitungan bitwise antara 1 0 AND 1 0 0 akan menghasilkan 0. Ketika hasil operasi AND adalah 0, maka iterasi tidak dilanjutkan karena tidak mungkin titik selanjutnya memenuhi. Apabila hasil dari operasi tersebut tidak 0, maka iterasi akan dilanjutkan karena masih terdapat kemungkinan untuk menempatkan bangunan dengan suatu jarak tertentu.

2.7 Arithmetic Shift Left

Operasi *Arithmetic Shift* merupakan salah satu operasi dari *Bitwise Operation* yang berfungsi untuk menggeser seluruh bit dari bilangan. *Shift Left* berguna untuk menggeser ke arah kiri seluruh bit dari bilangan dan menambahkan angka nol (0) pada posisi kosong (*Least Significant Bit*) baru yang dihasilkan dari pergeseran bit. Pergerakan tersebut tertera pada Gambar 2.16



Gambar 2.16 Operasi Aritmatika *shift left* [3]

2.8 Pencarian Nilai Maksimum e dan Minimum d

Langkah pertama dalam penyelesaian permasalahan HOUSEBUY adalah melakukan pra komputasi untuk setiap petak yang nantinya akan digunakan untuk operasi AND. Sebagai contoh, akan digunakan masukan sesuai dengan gambar 2.2

1	2	4	8	16
32	64	128	256	512
1024	2048	4096	8192	16384
32768	65536	131072	262144	524288
1048576	2097152	4194304	8388608	16777216

Gambar 2.18 Perhitungan nilai petak

Struktur data graf digunakan untuk memodelkan tata letak kota dengan bangunan sebagai titik yang menyimpan koordinat dari bangunan tersebut dan jumlah langkah untuk mencapai dari satu bangunan ke bangunan lain akan dimodelkan sebagai garis. Pertama, setiap bangunan akan diberi *id* yang berbeda dan mencatat *id* dari bangunan yang merupakan rumah (*house*). Menggunakan contoh diatas, maka id dari setiap bangunan adalah seperti pada Gambar 2.17

Bangunan	Id
house1	0
postoffice	1
school	2
house2	3
house3	4

Gambar 2.17 id bangunan

Bersamaan dengan pencatatan *id*, dilakukan pembentukan graf untuk setiap bangunan dengan *edge* yaitu jarak dari bangunan pertama ke bangunan kedua. Hal ini dilakukan untuk menetapkan bahwa jarak dari bangunan pertama ke bangunan kedua adalah d. Sebelum melakukan pencarian, sistem dibuat

untuk memilah dan mencatat bangunan yang telah memiliki lokasi yang tidak dapat berubah (Gambar 2.19).

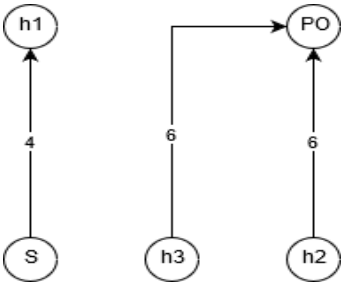
Barulah setelah itu sistem akan menyimpan jarak

	0	1	2	3	4
A	H1				PO
B					
C					
D					

Gambar 2.19 Posisi bangunan awal

maksimum antar bangunan yang kemudian digunakan sebagai nilai untuk menentukan apakah jarak tersebut sesuai dengan ketentuan atau tidak (Gambar 2.20).

Pencarian nilai maksimum e dilakukan dengan melakukan



Gambar 2.20 Graf hubungan bangunan

iterasi pada setiap rumah dan dilakukan pengecekan jarak dengan setiap rumah berikutnya. Pada setiap iterasi, dilakukan juga pengecekan apakah lokasi yang telah ditetapkan tersebut sesuai dengan ketentuan jarak dengan bangunan lain yang berhubungan dengan rumah tersebut.

Pengecekan dilakukan dengan melakukan *pruning* berbasis *bitwise*, yaitu dengan mengeliminasi jarak yang sudah dipastikan tidak mungkin dicapai. *Pruning* dilakukan dengan mengoperasikan nilai dari bangunan a menggunakan operasi

AND dengan kemungkinan jarak yang akan menentukan lokasi bangunan tersebut ditempatkan. Apabila hasil *pruning* sama dengan nol, maka keluar dari iterasi dan lanjut untuk melakukan pencarian jarak pada bangunan berikutnya. *Pruning* akan berhenti ketika jarak d telah tercapai dari hasil operasi AND pada bagian sebelumnya.

Apabila nilai e telah ditemukan, sistem akan mencatat seluruh nilai e terendah dari setiap pasangan rumah dan mencari nilai e tertinggi dari nilai-nilai tersebut, kemudian mencatat seluruh pasangan rumah dengan nilai sama dengan nilai e tertinggi. Hasil dilihat pada Gambar 2.22 dan Gambar 2.21

	0	1	2	3	4
A	H1				PO
B					
C	H2				
D		S			

Gambar 2.22 Formasi letak Pertama

	0	1	2	3	4
A	H1				PO
B					
C	H3				
D		S			
E			H2		

Gambar 2.21 Formasi letak Kedua

Sehingga, kombinasi nilai e dari seluruh formasi adalah: e[1,2] = 2, e[1,3] = 2, e[2,3] = 4. Dari hasil tersebut, nilai maks e yang didapat adalah 4 dengan pasangan rumah yaitu: house2 dan house3.

Pencarian nilai minimum d dilakukan setelah seluruh rumah telah ditempatkan dan memiliki jarak yang satu dengan

yang lainnya. Iterasi akan dilakukan pada seluruh rumah dengan rumah yang lain dan mencari nilai d terendah dari seluruh

	0	1	2	3	4
A	H1				PO
B					
C	H2				
D		S			
E			H3		

	0	1	2	3	4
A	H1				PO
B					
C	H3				
D		S			
E			H2		

Gambar 2.23 Jarak d terjauh dari kedua peta

kombinasi nilai d tertinggi pada setiap rumah (Gambar 2.23).

Dari hasil yang didapat, maka nilai d tertinggi dari setiap formasi adalah: d (L1) = 6, d(L2) = 6. Sehingga nilai d minimum adalah 6.

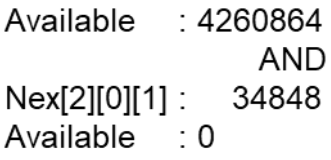
BAB III
ANALISIS DAN PERANCANGAN

Pada bagian ini akan dijelaskan mengenai analisis struktur data yang digunakan untuk menyelesaikan permasalahan pada tugas akhir ini. Pada bagian pertama akan dijelaskan deskripsi umum mengenai sistem yang akan dibuat, kemudian dilanjutkan dengan desain dari struktur data, beserta desain fungsi untuk mendapatkan nilai maksimum e dan minimum d.

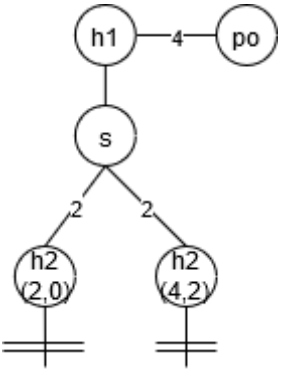
3.1 Analisis Sistem

Sistem yang akan dibuat adalah sistem yang menggunakan metode *Breadth-First Search* (BFS), dimana BFS merupakan salah satu dari algoritma *Complete Search*. Metode ini tepat digunakan untuk menyelesaikan persoalan karena BFS mampu untuk mencari seluruh kemungkinan kombinasi yang ada.

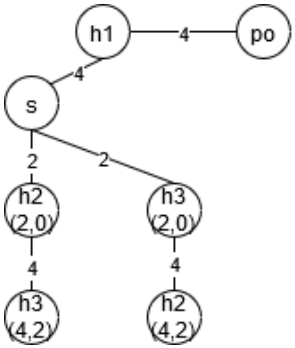
Pada persoalan tersebut terdapat kombinasi *node* yang apabila ditelusuri tidak akan memberikan hasil yang terbaik. Sehingga, ada kemungkinan bahwa *node-node* tersebut dapat dihilangkan/*pruning*. Karakteristik yang dapat dilihat untuk menentukan *node* yang dapat dilakukan *pruning* adalah dengan melihat hasil operasi *bitwise and* pada *node* tersebut, dimana apabila hasil operasi adalah nol (Gambar 3.1), maka *node* tersebut dapat dihilangkan/*pruning*. Cara inilah yang akan mengurangi jumlah *node* yang ditelusuri pada saat melakukan pencarian dengan metode BFS (Gambar 3.2 dan Gambar 3.3).



Gambar 3.1 operasi bitwise and pada node



Gambar 3.3 *tree* dengan *pruning*



Gambar 3.2 keseluruhan *tree* tanpa *pruning*

3.2 Deskripsi Umum Sistem

Pada sub-bab ini akan dijelaskan mengenai gambaran secara umum dari sistem yang akan dibuat. Pertama, sistem akan menerima masukan berupa m dan n yang masing-masing merupakan jumlah blok pada setiap arah (horizontal dan vertikal). Setelah menerima masukan, sistem akan mengolah masukan tersebut dan melakukan komputasi untuk setiap kemungkinan

petak berkesinambungan dengan jaraknya. Sistem kemudian akan menerima masukan dari salah satu diantara dua ketentuan pada setiap barisnya, yaitu:

Nama LOCATION r c

Nama DISTANCE d Nama2

Selanjutnya, sistem akan mencatat dan memberikan *id* pada setiap bangunan. Sehingga setiap bangunan akan dipanggil dengan *id*-nya pada proses-proses selanjutnya.

Pada setiap baris yang dimasukkan, sistem akan melakukan pengecekan apakah baris yang dimasukan merupakan baris yang mengandung informasi lokasi dari bangunan atau baris yang mengandung informasi tentang jarak dari bangunan pertama ke bangunan kedua. Apabila masukan mengandung *string* "LOCATION" maka sistem akan menyimpan dan memberikan label bahwa bangunan tersebut telah memiliki lokasi tetap yang tidak dapat dirubah. Namun, apabila baris masukan mengandung *string* "DISTANCE", maka sistem akan membentuk graf yang menyimpan bangunan pertama, bangunan kedua, beserta jarak antara kedua bangunan tersebut. Masukan kasus akan diakhiri apabila menerima masukan berupa *string* "end" yang menandakan bahwa masukan untuk kasus tersebut sudah berakhir.

Kemudian, sistem akan melakukan pencarian menyeluruh terhadap seluruh bangunan dan melakukan pengecekan apakah lokasi yang didapat telah sesuai dengan ketentuan yang dimasukkan. Pencarian tersebut akan dipercepat dengan bantuan *pruning* yang akan mengeliminasi kemungkinan-kemungkinan yang tidak perlu dilalui. Setelah itu, sistem baru akan mencari nilai maksimum dari e dengan mengurutkan nilai maksimum dari nilai-nilai e yang didapatkan dan nilai minimun dari d dengan mengurutkan nilai-nilai d yang diapatkan dan mengurutkannya.

Kemudian, sistem akan menunggu untuk menerima kasus masukan kedua dan seterusnya hingga menemukan angka 0 0 yang menandakan bahwa tidak akan ada masukan lagi.

Keluaran dari sistem adalah integer e dan d, dimana e merupakan nilai maksimum dari keseluruhan nilai minimum e pada setiap kemungkinan hubungan rumah, dan d merupakan nilai minimum dari keseluruhan nilai maksimum d pada setiap kemungkinan hubungan rumah, dari seluruh kemungkinan kombinasi penempatan bangunan. Setelah itu sistem menuliskan seluruh kombinasi rumah yang memiliki nilai e sama dengan nilai maksimum e. *Pseudocode* dari fungsi *main* dapat dilihat pada Gambar 3.1.

3.3 Desain Fungsi clear

Fungsi clear pada penyelesaian permasalahan HOUSEBUY memiliki peran penting karena memiliki peran sebagai inisiator dan memastikan bahwa setiap elemen yang akan digunakan telah ditetapkan sesuai ketentuan awal. *Pseudocode* dai fungsi *clear* dapat dilihat pada Gambar 3.5.

3.4 Desain Fungsi pra komputasi

Fungsi pra komputasi pada penyelesaian permasalahan HOUSEBUY berfungsi untuk melakukan perhitungan pada setiap petak. Iterasi dilakukan sejumlah setiap petak dan setiap petak akan menyimpan nilai sebanyak jumlah kemungkinan langkah, dimana maksimal langkah = m + n. Pada operasi yang dilakukan, pertama-tama sistem akan membandingkan apakah nilai dari petak tersebut lebih kecil dari maksimal langkah, jika ya, maka nilai petak merupakan hasil operasi nilai petak dengan posisi dari petak tersebut dihitung dari posisi 0,0 yang telah dilakukan operasi *shift left* sebanyak 1 *bit*. *Pseudocode* dari fungsi *precompute* dapat dilihat pada Gambar 3.6

```
main()
WHILE 1 do
   Call: clearAll()
   Tinggi ← INPUT
```

```
        Lebar ← INPUT
        IF tinggi = 0 and lebar = 0 then
           Exit
        Tinggi_lebar ← tinggi * lebar
        Call: Precomp_dist()
        Call: Read_input()
        Call: Create_edge()
        Call: Solve_maxe()
        Call: Solve_mind()
        Call: Print ”mind maxe”
        FOR pr 0,ctk do
           Cetak(pr.fi)
           Print “ “
           Cetak(pr.se)
           Endline
        Endfor
     Endwhile
```

Gambar 3.4 Gambar *pseudocode* fungsi *main*

```
clearAll()
ids SET 0
pos_x  SET -1
pos_y SET -1
house SET NULL
edge SET NULL
avail_building SET NULL
list_house SET NULL
petak SET NULL
fixed_location SET NULL
fixed_distance SET NULL
counter SET 0
return
```

Gambar 3.5 Gambar pseudocode fungsi clearAll

```
Precompute()
Bts ← tg + lb
Tps SET 0
Kd SET 0
For i ← 0,tg do
   For j ← 0,lb do
      For k ← 0,bts do
         Nex[i][j][k] ←{0, 0}
      ENDFOR
      For k ← 0,tg do
         For l ← 0,lb do
            Tps ← abs(i-k) + abs(j-l)
            Kd ← k*lb + l
            Nex[i][j][tps] ← Nex[i][j][tps] or kd
         ENDFOR
      ENDFOR
   ENDFOR
ENDFOR
```

Gambar 3.6 Gambar *pseudocode* fungsi *precompute*

3.5 Desain Fungsi input

Pada fungsi input, sistem akan menunggu untuk menerima masukan. Apabila menerima masukan, sistem pertama-tama akan masuk ke fungsi end. Dimana fungsi end berfungsi untuk mengecek apakah masukan tersebut merupakan string “end”. Apabila ya, maka sistem akan berhenti menunggu masukan dan lanjut ke tahap selanjutnya. *Pseudocode* dari fungsi *end* adalah seperti pada Gambar 3.7.

```
End()
Char_end ← {‘E’, ‘N’, ‘D’}
Len  ← length of input
IF len is not 3 THEN
   Return 0
FOR i ← 0,len do
   IF input[i] is not char_end[i] THEN
      Return 0
   ENDIF
ENDFOR
```

Gambar 3.7 Gambar *pseudocode* fungsi *end*

Dilanjutkan dengan melakukan pemberian *id* pada setiap bangunan yang masuk (*Pseudocode* dapat dilihat pada Gambar 3.8). Pemberian id ini akan dilakukan pada fungsi getid, dimana pada fungsi getid ini selain memberikan id, akan melakukan pengecekan apakah bangunan tersebut merupakan rumah atau bukan dengan fungsi checkhouse yang akan mengecek apakah kelima karakter pertama dari masukan tersebut merupakan karakter h, o, u, s, dan e. Apabila merupakan rumah, maka sistem juga akan memberikan penanda terhadap id dari rumah tersebut yang menandakan bahwa id tersebut adalah rumah dengan *pseudocode* dari fungsi *checkhouse* dapat dilihat pada Gambar 3.9

```
Getid()
Len ← length of input
FOR i ← 0,len do
   Hsh ← (1 * hsh * BASE) + call: conv(input[i])
ENDFOR
it ← ids.find(hsh)
IF it is not ids.end then
   Return it.second
ids[hsh] ← counter
ln[counter] ← len
FOR i ← 0,len do
   name[counter][i] ← input[i]
ENDFOR
IF call: checkhouse is true then
   house ← add counter
   ishouse[counter] ← 1
ELSE
   ishouse[counter] ← 0
ENDIF
```

Gambar 3.8 Gambar *pseudocode* fungsi *getid*

```
Checkhouse()
Char_house ← {‘h’, ‘o’, ‘u’, ‘s’, ‘e’}
Len ← length of input
IF len < 5 then
   Return 0
FOR i ← 0,5 do
   IF input[i] is not char_house[i] then
      Return 0
   ENDIF
ENDFOR
```

Gambar 3.9 Gambar pseudocode fungsi checkhouse

Setelah didapatkan id dari bangunan, maka akan dilakukan pengecekan terhadap karakter pertama dari string kedua yang dimasukkan. Apabila karakter tersebut adalah ‘L’, maka sistem

akan menyimpan nilai y yang merupakan lokasi pada koordinat y dan x yang merupakan lokasi pada koordinat x. Namun, apabila bukan karakter 'L', maka sistem akan mendapatkan id dari bangunan kedua yang dimasukkan dengan fungsi getid dan menyimpan bangunan pertama, bangunan kedua, beserta jaraknya. *Pseudocode* fungsi *readinput* terdapat pada Gambar 3.10

```
Readinput()
While 1 do
   input ← INPUT
IF call: checkend() is true THEN
      EXIT
ENDIF
   id ← call: getid()
   inp ← INPUT
   IF inp[0] is 'L' THEN
      inp ← INPUT
      Y ← inp[0]-'A'
      inp ← INPUT
      x ← inp[0]-'0'
      pos_y[id] ← y
      pos_x[id] ← x
   ELSE
      d ← INPUT
      input ← INPUT
      id_a ← call: getid()
      edge ← pair (d, pair (id_a, id))
   ENDIF
ENDWHILE
total_house SET number of house
```

Gambar 3.10 Gambar *pseudocode* fungsi *readinput*

3.6 Desain Fungsi CreateEdge

Pada bagian ini, pertama-tama akan mendapatkan nilai awal dari fungsi setAvailable (*pseudocode* pada Gambar 3.12). Fungsi setAvailable akan menghasilkan sebuah pasangan jarak

terjauh yang dapat digunakan untuk menempatkan sebuah bangunan pada lokasi tertentu. Setelah itu, dilakukan iterasi sebanyak jumlah bangunan dimana di setiap iterasi dilakukan pengecekan apakah posisi dari bangunan tersebut masih dapat berubah. Apabila ya, maka jarak maksimal bangunan tersebut adalah nilai dari awal yang sebelumnya didapat melalui fungsi setAvailable. Setelah itu bangunan tersebut akan ditambahkan ke daftar bangunan yang masih dapat berubah-ubah lokasinya. Apabila tidak, maka dilakukan pencatatan tentang posisi y dan x dari bangunan tersebut dan memberikan penanda bahwa lokasi tersebut telah diokupasi oleh sebuah bangunan.

Dilanjutkan dengan mengiterasi setiap *edge* yang telah dibentuk pada fungsi readinput sebelumnya. Apabila saat iterasi, bangunan pertama telah memiliki letak yang pasti, maka kemungkinan lokasi bangunan kedua dicari dengan mengoperasikan kemungkinan jarak dengan variabel nex dengan index y dari bangunan pertama, x dari bangunan pertama, dan jarak bangunan pertama dan kedua yang ditentukan pada saat menerima masukan(Gambar 3.11).

```
createEdge()
awal ← call: setAvailable()
FOR i ← 0,counter do
   IF pos_x[i] is -1 then
      available[i] ← awal
      avail_building ADD i
   ELSE
      petak[pos_y[i]][pos_x[i]] ← 1
      fixed_location[i] ← 1
   ENDIF
ENDFOR
total_avail ←  size of avail_building
FOR ed ← 0,edge do
   building1 ← ed
   IF pos_x[building2] is not -1 then
      SWAP building1 WITH building2
```

Gambar 3.11a Gambar pseudocode fungsi createEdge

```
ENDIF
   IF pos_x[building1] is not -1 then
      available[building2] ← available[building2] and
      nex[y][x][d]
   ELSE
adj[bulding1] ADD pair (building2, d)
      adj[building2] ADD pair (building1, d)
   ENDIF
FOR i ← 0,counter do
   sort adj
   erase unique adj
ENDFOR
```

Gambar 3.11b Gambar pseudocode fungsi createEdge

```
setAvailable()
IF tinggi_lebar > bts then
   first of ret ← bts shiftleft 1 – 1
   second of ret ← ((tinggi_lebar – bts) shiftleft 1) – 1
ELSE
   first of ret ← (tinggi_lebar shiftleft 1) – 1
   second of ret ← 0
ENDIF
```

Gambar 3.12 Gambar *pseudocode* fungsi *setAvailable*

3.7 Desain Fungsi maksimum e

Fungsi Maksimum e merupakan fungsi yang bertugas untuk mencari nilai e tertinggi (Gambar 3.13). Diawali dengan melakukan iterasi pada setiap rumah dan membandingkannya dengan rumah yang lain. Apabila kedua rumah telah memiliki lokasi yang tetap, maka jarak antar rumah dapat langsung dihitung. Namun, apabila belum, maka akan melakukan iterasi yang sebanyak maksimum langkah (tinggi + lebar) dimulai dari angka 1. Pertama-tama, rumah pertama akan dipasangkan dengan

rumah kedua dengan jarak bernilai langkah yang sedang diiterasi. Begitu juga sebaliknya untuk rumah kedua dengan rumah pertama. Lalu diberi *flag* yang akan digunakan saat *pruning*.

```
Maxe()
Stp ← 0
Max_langkah ← tinggi + lebar
FOR i ← 0,total_house do
   House1 ← house[i]
   FOR j ← i+1,total_house do
      House2 ← house[j]
IF fixed_location[house1] is true and
      fixed_location[house2] is tue then
         Distance of house1 and house2 = abs(y[house1]-
         y[house2]) + abs(x[house1]-x[house2])
      ENDIF
If distance of house1 and house2 is not null then
         Continue
      ENDIF
      FOR langkah ←1,max_langkah do
         Adj[house1] ADD pair(house2, langkah)
         Adj[house2] ADD pair(house1, langkah)
         Found ← 0
         Call: prune(0)
         Adj[house1] DELETE last item
         Adj[house2] DELETE last item
         IF found is true then
            Distance[house1][house2] ← langkah
            EXIT
         ENDIF
      ENDFOR
   ENDFOR
ENDFOR
Found_maxe ← -1
For i ←0,total_house do
```

Gambar 3.13a Gambar *pseudocode* fungsi *maxe*

```
        House1 ← house[i]
        For j ← i+1,total_house do
            House2 ← house[j]
            If distance[house1][house2] > found_maxe then
                Found_maxe ← distance[house1][house2]
            List_house SET null
            ENDIF
    If Found_maxe is distance[house1][house2] then
                List_house push pair(house1, house2)
            ENDIF
        ENDFOR
    ENDFOR
```

Gambar 3.13b Gambar *pseudocode* fungsi *maxe*

Selanjutnya adalah masuk ke fungsi pruning. Fungsi pruning merupakan fungsi rekursi dengan nilai awal adalah 0 hingga jumlah bangunan yang belum memiliki lokasi pasti. Yang dilakukan pertama adalah mencari bangunan dengan bit yang menyalanya paling banyak menggunakan fungsi msb(). Setelah didapatkan, maka bangunan tersebut selanjutnya akan diiterasi selama masih memiliki kemungkinan jarak yang dapat diubah.

Pada iterasi, dihitung nilai y dan x dari bangunan tersebut dan dicek apakah lokasi y dan x pada peta telah terisi atau belum. Selanjutnya, lokasi y dan x di-*set* sebagai terisi pada peta dan menempatkan posisi y dan x dari bangunan tersebut sebagai y dan x yang telah dihitung.

Dari bangunan yang telah ditentukan, dilakukan iterasi sebanyak bangunan yang memiliki hubungan dengan bangunan tersebut. Apabila bangunan yang berhubungan telah memiliki lokasi pasti, jarak antar bangunan dapat langsung dihitung dan dapat keluar dari iterasi. Apabila belum, maka kemungkinan jarak dihitung kembali dengan operasi AND untuk menentukan jarak yang dapat dicapai.

Apabila sudah ditemukan, maka keluar dari rekursif pruning dan jarak antara rumah pertama dan rumah kedua merupakan nilai dari langkah sedang diiterasi.

Setelah seluruh rumah telah ditemukan jaraknya antara yang satu dengan yang lain, maka dilakukan iterasi kembali untuk mendapatkan jarak tertinggi dan membuat daftar pasangan rumah dengan jarak tersebut. *pseudocode* pada Gambar 3.14

```
Prune(current)
If current is total_avail then
   Found ← true
   Return
ENDIF
Maks ← 200
For building ← 0,avail_building do
   If fixed_location[building] is true then
      Continue
   ENDIF
   Jarak ← cbit(building)
If jarak < maks then
      Maks ← jarak
      Curr_building ← building
   ENDIF
If maks is 0 then
   Return
ENDIF
Fixed_location[curr_building] ← 1
For i←0,counter do
   Save[curr_building][i] ← available[i]
ENDFOR
While available[cur_building] is true do
   Kd ← msb(curr_building)
   Y ← kd / lebar
   X ← kd – (y * lebar)
   If petak[y][x] is true then
      Continue
```

```
    ENDIF
    Valid ← 1
```

Gambar 3.14a Gambar *pseudocode* fungsi *prune*

```
Petak[y][x] ← 1
  If stp is true and isHouse[curr_building] is true then
    For curr_house ← 0,house do
      If curr_house is curr_building or
fixed_location[curr_house] is true then
        Continue
 ENDIF
      Distance ← abs(y-pos_y[curr_house])+abs(x-
      pos_x[curr_house])
      If distance > lim then
        Valid ← 0
        Break
      ENDIF
    ENDFOR
  ENDIF
Pos_y[curr_building] ← 1
  Pos_x[curr_building] ← 1
  For mk ← 0,adj[curr_building] do
    If fixed_location[mk] is true then
      Dst ← abs(pos_y[curr_building]-pos_y[mk]) +
      abs(pos_x[curr_building]-pos_x[mk])
      If dst is not mk then
        Valid ← 0
        Break
      ENDIF
      Continue
    ENDIF
    Available[mk] ← save[cur_building][mk] and
    nex[y][x][mk]
    If available[mk] is false then
      Valid ← 0
      Break
```

```
        ENDIF
      ENDFOR
```

Gambar 3.14b Gambar *pseudocode* fungsi *prune*

```
        If valid is true then
          Call: prune(current+1)
        ENDIF
      Petak[y][x] ← 0
        If found is true then
          Break
        ENDIF
      Endwhile
      For i ← 0,counter do
        Available[i] ← save[curr_building][i]
      ENDFOR
Fixed_location[curr_building] ← 0
```

Gambar 3.14c Gambar *pseudocode* fungsi *prune*

```
      Cbit(x)
      a ← bitcount(available[x] first)
      b ← bitcount(available[x] second)
      return a + b
```

Gambar 3.15 Gambar *pseudocode* fungsi *cbit*

```
      Msb()
      If available[x] has second is true
        Tps ← available[x] second and (-available[x] second)
        No ← bitcount(tps - 1)
        Available[x] second ← available[x] second – tps
        Return no + bts
      Else
        Tps ← available[x] first and (-available[x] first)
        No ← bitcount(tps - 1)
        Available[x] first ← available[x] first – tps
```

```
    Return no
ENDIF
```

Gambar 3.16 Gambar *pseudocode* fungsi *msb*

3.8 Desain Fungsi minimum d

Fungsi minimum d merupakan fungsi untuk mencari nilai d terkecil diantara seluruh nilai d maksimum pada setiap formasi lokasi. Dimana d merupakan jarak dari rumah pertama ke rumah kedua. Fungsi ini dijalankan setelah didapatkan nilai untuk maksimum e, dimana pada fungsi tersebut, setiap rumah telah menentukan lokasinya. *Pseudocode* fungsi *mind* terdapat pada Gambar 3.17.

Fungsi minimum d diawali dengan mengiterasi sejumlah rumah dan dibandingkan dengan setiap rumah lainnya. Apabila kedua rumah tersebut telah memiliki lokasi pasti, maka dihitung nilai min d nya. Tidak menutup kemungkinan bahwa masih terdapat nilai d yang lain yang merupakan nilai d minimum. Oleh karena itu, dilakukan iterasi selama nilai mind masih lebih kecil dari jumlah maksimum langkah. Hal ini dilakukan dengan menggunakan kembali fungsi pruning.

Pada fungsi pruning, dicari kembali setiap kemungkinan rumah yang pertama dengan rumah kedua dan apabila ditemukan bahwa jarak kedua rumah tersebut lebih besar, maka akan keluar dari fungsi dan menetapkan bahwa jarak tersebut adalah jarak terbaru.

```
Mind()
Stp ← 1
Found_min ← 0
Max_langkah ← tinggi + lebar
For i ← 0,total_house do
   House1 ← house[i]
   For j ← i+1,total_house do
      House2 ← house[j]
      If fixed_location[house1] and
      fixed_location[house2] is true then
```

```
            Foundmin ← max(found_min, distance(house1,
            house2)
        ENDIF
```

Gambar 3.17a Gambar *pseudocode* fungsi *mind*

```
        ENDFOR
    ENDFOR
    While found_mind <= max_langkah do
        Lim ← found_mind
    Found ← 0
        Call: prune(0)
        If found is true then
            Break
        ENDIF
        Found_mind++
Endwhile
```

Gambar 3.17b Gambar *pseudocode* fungsi *mind*

3.9 Desain Fungsi cetak

Fungsi cetak adalah fungsi yang digunakan untuk mencetak pasangan rumah. Pada fungsi ini, menerima id bangunan sebagai masukan dan mengeluarkan nomor rumah yang sesuai dengan id tersebut. *Pseudocode* untuk fungsi cetak dapat dilihat pada Gambar 3.18.

```
    Cetak(x)
    For i ← 0,ln[x] do
        Print(name[x][i])
    ENDFOR
```

Gambar 3.18 Gambar *pseudocode* fungsi *cetak*

BAB IV
IMPLEMENTASI

Pada bab ini akan dijelaskan mengenai implementasi dari penerapan struktur data sebagai penyelesaian permasalahan HOUSEBUY. Pertama akan dijelaskan implementasi dari fungsi utama, dan dilanjutkan dengan implementasi dari struktur data beserta fungsi yang digunakan.

4.1 Lingkungan Implementasi

Lingkungan implementasi dan pengembangan yang digunakan untuk menyelesaikan tugas akhir ini adalah sebagai berikut.

1. Perangkat Keras
   a. Prosesor
   b. *Random Access Memory*
2. Perangkat Lunak
   a. *Operating System* Windows 10 64-bit
   b. Bahasa Pemrograman C++
   c. *Integrated Development Environment* Orwell Dev-C++

4.2 Implementasi Fungsi main

Fungsi main adalah implementasi Algoritma yang dirancang pada subab 3.1. Implementasi fungsi main dapat dilihat pada Kode Sumber 4.1.

```
int main()
{
   while (1)
   {
      clearAll();
      scanf("%d %d", &tinggi, &lebar);
      if (tinggi == 0 && lebar == 0)
```

Kode Sumber 4.1a Implementasi fungsi *main*

```
        break;
            tinggi_lebar = tinggi * lebar;
            precompute();
            readInput();
            createEdge();
            maxe();
            mind();
            cout << found_mind << " ";
            cout << found_maxe << "\n";
            for (auto print_house : list_house)
            {
               cetak(print_house.fi);
               cout << " ";
               cetak(print_house.se);
               endln;
            }
         }
}
```

Kode Sumber 4.1b Implementasi fungsi *main*

Untuk mencetak hasil, maka diperlukan sebuah fungsi untuk mengambil id dari bangunan dan menerjemahkan id tersebut ke bangunan yang memiliki id tersebut lalu dicetak untuk mengeluarkan keluaran. Hal ini diimplementasikan melalui fungsi cetak. Fungsi cetak berada pada Kode Sumber 4.2.

```
        void cetak(int x)
        {
           for (int i = 0; i < ln[x]; i++)
              printf("%c", name[x][i]);
           return;
        }
```

Kode Sumber 4.2 Implementasi fungsi *cetak*

4.3 Implementasi Fungsi clear

Fungsi clear berfungsi untuk melakukan clear pada variabel beserta struktur data yang akan digunakan. Hal ini diperlukan

agar setiap masukan dapat dijalankan dengan dimulai dari titik awal yaitu nol atau kosong, sehingga tidak berhubungan dengan masukan sebelumnya. Funsi *clear* berada pada Kode Sumber 4.3.

```
void clearAll()
{
   ids.clear();
   setmin(pos_x);
   setmin(pos_y);
   house.clear();
   edge.clear();
   avail_building.clear();
   list_house.clear();
   setnul(petak);
   setnul(fixed_location);
   setnul(fixed_distance);
   counter = 0;
   return;
}
```

Kode Sumber 4.3 Implementasi fungsi *clear*

4.4 Implementasi fungsi pra komputasi

Fungsi pra komputasi merupakan fungsi yang bertujuan untuk menentukan nilai awal pada setiap petak dengan jumlah langkah yang telah ditentukan. Berikut implementasi dari fungsi tersebut. Fungsi *precompute* terdapat pada Kode Sumber 4.4

```
void precompute()
{
   int batas = tinggi + lebar, langkah, kd;
   for (int i = 0; i < tinggi; i++)
   {
      for (int j = 0; j < lebar; j++)
      {
         for (int k = 0; k <= batas; k++)
         {
            nex[i][j][k] = {0LL, 0LL};
```

Kode Sumber 4.4a Implementasi fungsi *precompute*

```
        }
                for (int k = 0; k < tinggi; k++)
                {
                   for (int l = 0; l < lebar; l++)
                   {
                      langkah = abs(i - k) + abs(j - l);
                      kd = k * lebar + l;
                      nex[i][j][langkah] = nex[i][j][langkah] | kd;
                   }
                }
             }
          }
          return;
}
```

Kode Sumber 4.4b Implementasi fungsi *precompute*

4.5 Implementasi Fungsi readinput

Fungsi readinput merupakan fungsi yang bertugas untuk menerima masukan dan mengolah masukan tersebut sesuai kebutuhan yang pada bagian selanjutnya akan diproses lebih lanjut. Fungi *readinput* dapat dilihat di Kode Sumber 4.5.

```
        void readInput()
        {
           while (1)
           {
              scanf("%s", input);
              if (end())
                 break;
              int id = getid();
              scanf("%s", input);
              if (input[0] == 'L')
              {
                 int y, x;
                 scanf("%s", input);
```

Kode Sumber 4.5a Implementasi fungsi *readInput*

```
                y = input[0] - 'A';
                scanf("%s", input);
                x = input[0] - '0';
                pos_y[id] = y;
                pos_x[id] = x;
            }
            else
            {
                int id_a, d;
                scanf("%d", &d);
                scanf("%s", input);
                id_a = getid();
                edge.psb(mp(d, mp(id_a, id)));
            }
        }
        for (int i = 0; i < counter; i++)
        {
            adj[i].clear();
        }
        total_house = house.sz;
        return;
}
```

Kode Sumber 4.5b Implementasi fungsi *readInput*

Pada fungsi ini, diperlukan juga fungsi end yang bertujuan untuk melakukan pengecekan apakah masukan merupakan kata ‘end’, sehingga sistem dapat menentukan untuk menghentikan masukan. Fungsi *end* disajikan pada Kode Sumber 4.6.

```
char char_end[] = {'E', 'N', 'D'};
bool end()
{
    int len = strlen(input);
    if (len != 3)
        return 0;
    for (int i = 0; i < len; i++)
```

Kode Sumber 4.6a Implementasi fungsi *end*

```
        {
            if (input[i] != char_end[i])
                return 0;
        }
        return 1;
}
```

Kode Sumber 4.6b Implementasi fungsi *end*

Tidak hanya fungsi end yang digunakan pada readinput, tetapi juga terdapat fungsi getid (Kode Sumber 4.7) yang bertujuan untuk memberikan id pada setiap bangunan yang dimasukkan kedalam sistem.

```
int getid()
{
    int len = strlen(input);
    LL hsh = 0LL;
    for (int i = 0; i < len; i++)
        hsh = (1LL * hsh * BASE) + (LL)conv(input[i]);
    it = ids.find(hsh);
    if (it != ids.end())
    {
        return it->se;
    }
    ids[hsh] = counter;
    ln[counter] = len;
    for (int i = 0; i < len; i++)
        name[counter][i] = input[i];
    if (checkHouse())
    {
        house.psb(counter);
        isHouse[counter] = 1;
    }
```

Kode Sumber 4.7a Implementasi fungsi *getid*

```
        else
            isHouse[counter] = 0;
        return counter++;
}
```

Kode Sumber 4.7b Implementasi fungsi *getid*

Pada fungsi getid, dalam proses pemberian id pada setiap bangunan, sistem menggunakan metode hash yang akan mengubah bangunan tersebut agar dapat dibedakan yang satu dengan yang lain dan barulah diberikan id apabila belum memiliki id(Kode Sumber 4.8). Selain itu, diperlukan juga sebuah fungsi untuk mengecek apakah bangunan yang dimasukan merupakan sebuah rumah atau bangunan lain(Kode Sumber 4.9).

```
int conv(char x)
{
    if ('a' <= x && x <= 'z')
        return (int)(x - 'a') + 1;
    return (int)(x - '0') + 27;
}
```

Kode Sumber 4.8 Implementasi fungsi *conv*

```
char char_house[] = {'h', 'o', 'u', 's', 'e'};
bool checkHouse()
{
    int len = strlen(input);
    if (len < 5)
        return 0;
    for (int i = 0; i < 5; i++)
    {
        if (input[i] != char_house[i])
            return 0;
    }
    return 1;
}
```

Kode Sumber 4.9 Implementasi fungsi *checkhouse*

4.6 Implementasi Fungsi createEdge

Fungsi createEdge betujuan untuk mempesiapkan kondisi awal yang kemudian akan digunakan untuk diproses untuk mendapatkan hasil yang diinginkan. Kondisi awal tersebut adalah menentukan bangunan yang belum memiliki lokasi pasti dan menentukan jarak beserta capaian terjauh setiap bangunan. Fungsi *createEdge* dapat dilihat di Kode Sumber 4.10.

```
void createEdge()
{
   awal = setAvailable();
   for (int i = 0; i < counter; i++)
   {
      if (pos_x[i] == -1)
      {
         available[i] = awal;
         avail_building.psb(i);
      }
      else
      {
         petak[pos_y[i]][pos_x[i]] = 1;
         fixed_location[i] = 1;
      }
   }
   total_avail = avail_building.sz;
   for (auto ed : edge)
   {
      int building1 = ed.se.fi;
      int building2 = ed.se.se;
      if (pos_x[building2] != -1)
         swap(building1, building2);
      if (pos_x[building1] != -1 && pos_x[building2] !=
-1)
      {
```

Kode Sumber 4.10a Implementasi fungsi *createEdge*

```
                int langkah = abs(pos_x[building1] –
        pos_x[building2]) + abs(pos_y[building1] –
        pos_y[building2]);
            }
            else if (pos_x[building1] != -1)
            {
                available[building2] = available[building2] &
        nex[pos_y[building1]][pos_x[building1]][ed.first];
            }
            else
            {
                adj[building1].psb(mp(building2, ed.fi));
                adj[building2].psb(mp(building1, ed.fi));
                fixed_distance[building1][building2] =
        fixed_distance[building2][building1] = ed.fi;
            }
        }
        for (int i = 0; i < counter; i++)
        {
            sort(all(adj[i]));
            adj[i].erase(unique(all(adj[i])), adj[i].end());
        }
        return;
}
```

Kode Sumber 4.10b Implementasi fungsi *createEdge*

Dibutuhkan juga sebuah fungsi yang akan menghitung jangkauan terjauh yang dapat dicapai oleh sebuah umah dengan rumah lain. Fungsi tersebut dibei nama setAvailable (Kode Sumber 4.11).

```
        pll setAvailable()
        {
            pll ret;
            if (tinggi_lebar > bts)
            { ret.first = (1LL << bts) - 1LL;
```

Kode Sumber 4.11a Implementasi fungsi *setAvailable*

```
            ret.second = (1LL << (tinggi_lebar - bts)) - 1LL;
        }
        else
        {
            ret.first = (1LL << tinggi_lebar) - 1LL;
            ret.second = 0LL;
        }
        return ret;
}
```

Kode Sumber 4.11b Implementasi fungsi *setAvailable*

4.7 Implementasi Fungsi maxe

Fungsi maxe merupakan fungsi yang akan mencari jarak dengan nilai tertinggi dari keseluruhan nilai terendah dari setiap pasang rumah. Fungsi ini disajikan pada Kode Sumber 4.12.

```
    void maxe()
    {
        stp = 0;
        int max_langkah = tinggi + lebar, house1, house2;
        for (int i = 0; i < total_house; i++)
        {
            house1 = house[i];
            for (int j = i + 1; j < total_house; j++)
            {
                house2 = house[j];
                if (fixed_location[house1] &&
    fixed_location[house2])
                {
                    fixed_distance[house1][house2] =
    abs(pos_y[house1] - pos_y[house2]) +
    abs(pos_x[house1] - pos_x[house2]);
                }
                if (fixed_distance[house1][house2])
                    continue;
```

Kode Sumber 4.12a Implementasi fungsi *maxe*

```
            for (int langkah = 1; langkah <= max_langkah;
langkah++)
            {
               adj[house1].psb({house2, langkah});
               adj[house2].psb({house1, langkah});
               found = 0;
               prune(0);
               adj[house1].ppb();
               adj[house2].ppb();

               if (found)
               {
                  fixed_distance[house1][house2] = langkah;
                  break;
               }
            }
         }
      }
      found_maxe = -1;
      for (int i = 0; i < total_house; i++)
      {
         house1 = house[i];
         for (int j = i + 1; j < total_house; j++)
         {
            house2 = house[j];
            if (fixed_distance[house1][house2] > found_maxe)
            {
               found_maxe = fixed_distance[house1][house2];
               list_house.clear();
            }
            if (fixed_distance[house1][house2] ==
found_maxe)
            {
```

Kode Sumber 4.12b Implementasi fungsi *maxe*

```
                    list_house.psb({house1, house2});
                }
            }
        }
        return;
}
```

Kode Sumber 4.12c Implementasi fungsi *maxe*

Untuk mencari seluruh kombinasi dari rumah, fungsi maxe membutuhkan sebuah fungsi yang menerapkan metode pruning. Fungsi prune sendiri merupakan fungsi rekursif yang bertujuan untuk mencari jarak setiap rumah yang sesuai dengan ketentuan yang dimasukkan. Sehingga, dibuatlah fungsi prune (Kode Sumber 4.13) untuk menjawab kebutuhan tersebut.

```
void prune(int current)
{
    if (current == total_avail)
    {
        found = 1;
        return;
    }
    int curr_building, dst;
    maks = 200;
    for (auto building : avail_building)
    {
        if (fixed_location[building])
            continue;
        jarak = cbit(building);
        if (jarak < maks)
        {
            maks = jarak;
            curr_building = building;
        }
    }
```

Kode Sumber 4.13a Implementasi fungsi *prune*

```
    if (maks == 0)
        return;
    fixed_location[curr_building] = 1;

    for (int i = 0; i < counter; i++)
    {
        save[curr_building][i] = available[i];
    }

    while (available[curr_building].fi ||
available[curr_building].se)
    {
        int kd = msb(curr_building);
        int y = kd / lebar;
        int x = kd - (y * lebar);
        if (petak[y][x])
        {
            continue;
        }
        bool valid = 1;

        if (stp && isHouse[curr_building])
        {
            for (auto curr_house : house)
            {
                if (curr_house == curr_building ||
!fixed_location[curr_house])
                {
                    continue;
                }
                dst = abs(y - pos_y[curr_house]) + abs(x –
pos_x[curr_house]);
                if (dst > lim)
```

Kode Sumber 4.13b Implementasi fungsi *prune*

```
                {
                    valid = 0;
                    break;
                }
            }
        }
        if (!valid)
            continue;

        petak[y][x] = 1;
        pos_y[curr_building] = y;
        pos_x[curr_building] = x;

        for (auto mk : adj[curr_building])
        {
            if (fixed_location[mk.fi])
            {
                dst = abs(pos_y[curr_building] - pos_y[mk.fi]) +
abs(pos_x[curr_building] - pos_x[mk.fi]);
                if (dst != mk.se)
                {
                    valid = 0;
                    break;
                }
                continue;
            }
            available[mk.fi] = save[curr_building][mk.fi] &
nex[y][x][mk.se];
            if (!available[mk.fi].fi && !available[mk.fi].se)
            {
                valid = 0;
                break;
            }
        }
```

Kode Sumber 4.13c Implementasi fungsi *prune*

```
            if (valid)
            {
               prune(current + 1);
            }
            petak[y][x] = 0;
            if (found)
            {
               break;
            }
         }
         for (int i = 0; i < counter; i++)
         {
            available[i] = save[curr_building][i];
         }
         fixed_location[curr_building] = 0;
         return;
}
```

Kode Sumber 4.13d Implementasi fungsi *prune*

Dalam implementasinya, fungsi prune membutuhkan fungsi penunjang seperti fungsi cbit (Kode Sumber 4.14) yang bertujuan untuk menentukan bangunan dengan bit yang aktif (1) terbanyak dan fungsi msb (Kode Sumber 4.15) dengan tujuan untuk menentukan nilai y dan x dari sebuah bangunan.

```
int cbit(int x)
{
   return __builtin_popcountll(available[x].fi) +
__builtin_popcountll(available[x].se);
}
```

Kode Sumber 4.14 Implementasi fungsi *cbit*

```
int msb(int x)
{
   LL tps;
   int no;
```

Kode Sumber 4.15a Implementasi fungsi *msb*

```
        if (available[x].se)
        {
            tps = available[x].se & (-available[x].se);
            no = __builtin_popcountll(tps - 1LL);
            available[x].se -= tps;
            return no + bts;
        }
        else
        {
            tps = available[x].fi & (-available[x].fi);
            no = __builtin_popcountll(tps - 1LL);
            available[x].fi -= tps;
            return no;
        }
}
```

Kode Sumber 4.15b Implementasi fungsi *msb*

4.8 Implementasi Fungsi mind

Fungsi mind digunakan untuk mencari nilai d terkecil dari seluruh nilai d maksimum yang didapatkan dari kombinasi pasangan rumah. Untuk mencari nilai tersebut, fungsi ini menggunakan hasil dari pencarian yang telah dilakukan pada fungsi maxe dan apabila masih ada kemungkinan, maka mencari kembali dengan menggunakan fungsi prune. Keseluruhan fungsi *mind* dapat dilihat di Kode Sumber 4.16.

```
    void mind()
    {
        stp = 1;
        found_mind = 0;
        int max_langkah = tinggi + lebar, house1, house2;
        for (int i = 0; i < total_house; i++)
        {
            house1 = house[i];
            for (int j = i + 1; j < total_house; j++)
```

Kode Sumber 4.16a Implementasi fungsi *mind*

```
        {
         house2 = house[j];
                  if (fixed_location[house1] &&
        fixed_location[house2])
                  {
                     found_mind = max(found_mind,
        abs(pos_y[house1] - pos_y[house2]) + abs(pos_x[house1] -
        pos_x[house2]));
                  }
               }
            }
            while (found_mind <= max_langkah)
            {
               lim = found_mind;
               found = 0;
               prune(0);
               if (found)
                  break;
               found_mind++;
            }
            return;
}
```

Kode Sumber 4.16b Implementasi fungsi *mind*

[*Halaman ini sengaja dikosongkan*]

BAB V
UJICOBA DAN EVALUASI

Pada bab ini, dijelaskan mengenai uji coba dan evaluasi dari implementasi yang telah dilakukan pada tugas akhir ini. Uji coba dilakukan dengan skenario tertentu beserta evaluasi dan analisis dari metode yang digunakan untuk menyelesaikan permasalahan.

5.1 Lingkungan Uji Coba

Untuk pengujian, dilakukan pada situs penilai *online* SPOJ dan evaluasi kebenaran dilakukan melalui komputer personal milik penulis. Lingkungan uji coba dari situs SPOJ dengan *cluster cube* memiliki spesifikasi sebagai berikut:

1. Perangkat Keras
    a. Processor Intel Xeon E3-1220 v5 (5 CPUs)
    b. *Random Access Memory* 1536MB
2. Perangkat Lunak
    a. Compiler GCC 8.3

Untuk lingkungan uji coba yang digunakan untuk melakukan pengujian kinerja yaitu milik pribadi penulis memiliki spesifikasi sebagai berikut:

1. Perangkat Keras
    a. ProcessorIntel®Core™ i5-M560 CPU @ 2.67GHz (4 CPUs), ~2.7GHz
    b. *Random Access Memory* 4096MB
2. Perangkat Lunak
    a. *Operating System* Windows 10 Pro 64-bit
    b. Bahasa Pemrograman C++
    c. Integrated Development Environment Orwell Dev-C++ 5.11

5.2 Skenario Uji Coba

Pada bagian ini akan dijelaskan mengenai skenario uji coba yang digunakan untuk menguji implementasi dari solusi yang diusulkan dalam penyelesaian permasalahan HOUSEBUY.

Uji coba kebenaran dilakukan dengan melihat umpan balik yang diberikan oleh SPOJ setelah sumber kode dikirimkan. SPOJ akan menguji kebenaran dari sumber kode beserta solusi yang dikeluarkan. Batasan dari uji coba sendiri adalah batasan yang digunakan dalam permasalahan HOUSEBUY seperti yang telah dideskripsikan pada sub-bab 2.1, yaitu:

1. m sebagai banyaknya blok horizontal berupa bilangan *integer* 2 hingga 10.
2. n sebagai banyaknya blok vertikal berupa bilangan *integer* 2 hingga 10.
3. Deskripsi data dalam format dan akan diakhiri dengan bilangan 0 0:
    a. Nama LOCATION r c
    b. Nama DISTANCE d Nama2

Uji coba dilakukan dengan mengirimkan kode sumber program hasil implementasi solusi ke situs penilaian SPOJ sebanyak 10 kali yang kemudian dianalisa peforma dari umpan balik yang diberikan.

5.3 Uji Coba Kebenaran

Untuk skenario yang digunakan sebagai materi uji coba adalah masukan seperti pada Gambar 2.2. Dari masukan tersebut, maka dapat dilihat perubahan pada tabel. Dengan setiap iterasi maka ditemukan bahwa untuk menemukan jarak house1 dan house2 dengan jarak adalah 2.

Gambar 5.1 merupakan gambar dari lokasi awal sebelum memulai iterasi

	0	1	2	3	4
A	H1				PO
B					
C					
D					
E					

Gambar 5.1 Penempatan peta awal

Untuk mencari solusi, yang pertama dilakukan oleh sistem adalah mencoba seluruh kombinasi rumah, dimulai dengan *house1* dan *house2*. Dapat dilihat pada Gambar 5.1 bahwa house1 telah memiliki lokasi, yaitu A0, dan house2 belum memiliki lokasi. Oleh karena itu, dilakukan iterasi pencarian jarak antara house1 dan house2 yang dimulai dari 1.

Pencarian jarak yang dilakukan adalah dengan mengiterasi seluruh bangunan yang belum memiliki lokasi, dalam hal ini adalah school sebagai bangunan pertama yang belum memiliki lokasi. Setelah dilakukan perhitungan mengenai lokasi dari school yang memungkinkan dan sesuai dengan masukan yang diberikan, didapatkan bahwa lokasi school adalah D1 dan lokasi ini tetap dan tidak dapat berubah. Dapat dilihat pada Gambar 5.2.

h1				Po
	s			

Gambar 5.2 lokasi school

Dilanjutkan dengan bangunan selanjutnya, yaitu house2. Pada house2, dicoba dengan kemungkinan lokasi pertama, yaitu C0 (Gambar 5.4). Lokasi C0 adalah lokasi yang *valid*, tetapi masih tidak sama nilainya dengan iterasi jarak sekarang (1), sehingga diperlukan untuk mencari lokasi lain. Lokasi selanjutnya adalah D1, namun D1 telah ditempati oleh school sehingga dilanjutkan dengan lokasi selanjutnya. Lokasi ketiga adalah E2 (Gambar 5.3). Lokasi ini juga *valid*, tetapi masih tidak sama nilainya dengan iterasi jarak (1). Karena seluruh kemungkinan house1 dan house2 dengan iterasi jarak 1 tidak dapat ditemukan, maka nilai dari iterasi jarak ditambah 1 menjadi 2.

h1				Po
h2				
	s			

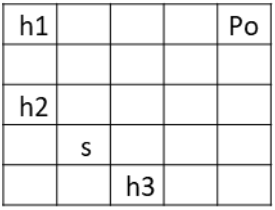
Gambar 5.4 house2 C0

h1				Po
	s			
		h2		

Gambar 5.3 house2 E2

Untuk iterasi jarak = 2, proses yang dilakukan sama ketika iterasi jarak =1, yaitu hingga school menempati lokasi D1. Perbedaan terjadi ketika house2 diletakkan pada C0. Pada iterasi=2, jarak house1 dengan house2 (C0) adalah 2. Hal ini membuktikan bahwa lokasi ini *valid* dan dapat diteruskan untuk melakukan pengecekan terhadap bangunan yang lain. Dilanjutkan

dengan bangunan house3. Karena lokasi yang dapat ditempati oleh house3 adalah E2 dan ketika dilakukan pengecekan ditemukan bahwa lokasi tersebut *valid*, maka ditemukan bahwa lokasi yang dapat digunakan dengan seluruh bangunan telah ditempatkan untuk house2 adalah C0. Sehingga jarak house1 dan house2 adalah 2 seperti pada Gambar 5.5.

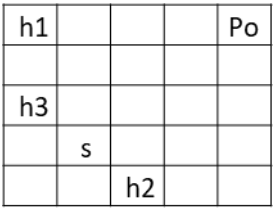


Gambar 5.5 kombinasi house1 dan house2

Karena house1 dan house2 telah ditemukan, maka dilanjutkan dengan kombinasi house1 dengan house3. Proses pencarian lokasi dari house3 sama seperti proses ketika proses pencarian house1 dan house2, yaitu dengan memulai iterasi jarak =1.

Untuk bangunan pertama, yaitu school akan menempati lokasi D1 sama seperti proses sebelumnya (Gambar 5.2). Dilanjutkan dengan menempatkan lokasi dari house2 terlebih dahulu pada C0 (Gambar 5.4). Pada kali ini, tidak dilakukan pengecekan jarak karena jarak yang dicari adalah jarak house1 dan house3. Setelah itu barulah dilanjutkan untuk house3. Karena C0 telah ditempati oleh house2, maka pada percobaan pertama house3 menempati E2 (Gambar 5.5). Dalam hal ini jarak house1 dengan house3 adalah 6 dan nilai ini tidak sama dengan iterasi jarak yaitu 1, sehingga belum memenuhi syarat. Dilanjutkan dengan kemungkinan selanjutnya, yaitu menempatkan house3 pada C0 dan memindah house2 pada E2 (Gambar 5.6). Namun jarak house1 dan house3 adalah 2 dan nilai iterasi adalah 1, sehingga belum ditemukan jarak yang sesuai.

Dilanjutkan dengan iterasi jarak = 2. Proses yang dilakukan sama seperti ketika iterasi jarak = 1. Namun, ketika house3 ditempatkan pada C0 dan house2 pada E2, jarak antara house1 dengan house3 adalah 2 dan nilai ini sama dengan nilai iterasi jarak, yaitu 2 (Gambar 5.6). Dari proses tersebut, lokasi untuk house3 adalah C0 dan jarak untuk house1 dengan house3 adalah 2.



Gambar 5.6 kombinasi house3 dan house1

Setelah house1 dan house3 ditemukan, maka dilanjutkan dengan kombinasi house2 dan house3. Pada kombinasi ini, kedua rumah belum memiliki lokasi. Proses yang dilakukan sama dengan nilai iterasi jarak adalah 1 dan dimulai dengan meletakkan school pada D2 (Gambar 5.2). Selanjutnya adalah meletakkan house2 pada lokasi yang tersedia yaitu C0. Karena kombinasi yang digunakan adalah house2 dan house3, maka ketika house2 telah ditempatkan, house3 juga harus ditempatkan. Dalam hal ini house3 menempati lokasi E2 (seperti pada Gambar 5.5). Karena kedua rumah pada awalnya belum memiliki lokasi yang pasti, maka dilakukan *pruning* untuk memotong kemungkinan yang dapat diiterasi.

Pruning dilakukan dengan mengoperasikan nilai dari house2 dengan nilai kemungkinan lokasi yang dapat dituju (NEX[y][x][jarak]). Dalam hal ini nilai dari house2 adalah 4260864 dan nilai dari NEX[2][0][1] adalah 34848. Hasil operasi AND dari kedua nilai tersebut adalah 0 (Gambar 5.8). Nilai 0 menandakan bahwa untuk house2 dengan lokasi C0, tidak ada house3 yang dapat dicapai dengan jarak 1. Karena tidak ada kemungkinan lagi, maka dicoba house2 dengan lokasi yang lain,

yaitu E2. Nilai dari house2 adalah 4260864 dan nilai untuk NEX[4][2][1] adalah 10616832. Ketika dilakukan operasi AND, maka menghasilkan nilai 0 (Gambar 5.7). Karena lokasi ini juga menghasilkan nilai 0. Maka menandakan bahwa tidak perlu dilanjutkan ke iterasi selanjutnya, karena tidak ada lagi lokasi yang dapat dicapai dengan jarak 1 oleh house2 ke house3.

```
Available    : 4260864
                    AND
Nex[2][0][1] :    34848
Available    : 0
```

Gambar 5.8 house2 AND NEX[2][0][1]

```
Available    : 4260864
                    AND
Nex[4][2][1] : 10616832
Available    : 0
```

Gambar 5.7 house2 AND NEX[4][2][1]

Dilanjutkan dengan iterasi jarak = 2. Pada proses ini akan sama hingga school (Gambar 5.2). Ketika menempatkan house2 pada C0 dan house3 pada E2, dilakukan perhitungan nilai dengan nilai house2 adalah 4260864 dan NEX[2][0][2] adalah 1118273. Operasi AND dari kedua bilangan tersebut adalah 65536 (Gambar 5.10). Karena mendapatkan hasil yang bukan nol, maka dilanjutkan dengan menempatkan house2 pada E2 dan house3 pada C0 kemudian dihitung nilainya (Gambar 5.9). Karena nilai yang didapat juga bukan nol, maka lokasi tersebut juga dapat digunakan. Kedua lokasi yang tidak menghasilkan nol tersebut *valid*, akan tetapi jarak kedua rumah tersebut adalah 4 dimana iterasi jarak adalah 2, sehingga tidak dapat dikatakan bahwa jarak house2 dan house3 telah ditemukan.

```
Available    : 4260864
                    AND
Nex[2][0][2] :  1118273
Available    : 65536
```

Gambar 5.10 house2 AND NEX[2][0][2]

```
Available    : 4260864
                    AND
Nex[4][2][2] : 18157568
Available    : 65536
```

Gambar 5.9 house2 AND NEX[4][2][2]

Karena belum ditemukan, maka iterasi jarak ditambah satu menjadi 3. Proses untuk iterasi jarak=3 sama seperti ketika iterasi jarak=1. Perbedaannya adalah nilai NEX, yaitu NEX[2][0][3] (Gambar 5.11) dan NEX[4][2][3] (Gambar 5.12). Namun hasil perhitungan keduanya menghasilkan nol yang menandakan tidak dapat dilanjutkan pencariannya.

```
Available    : 4260864
                    AND
Nex[2][0][3] : 2236546
Available    : 0
```

Gambar 5.11 house2 AND NEX[2][0][3]

```
Available    : 4260864
                    AND
Nex[4][2][3] :   567424
Available    : 0
```

Gambar 5.12 house2 AND NEX[4][2][3]

Setelah iterasi jarak = 3 tidak dapat dilanjutkan, maka iterasi jarak menjadi 4. Proses yang dilakukan sama hingga school (Gambar 5.2). Namun, berbeda ketika house2 diletakkan pada C0 dan house3 diletakkan pada E2 (Gambar 5.5). Hal ini karena hasil perhitungan memberikan hasil bukan nol (Gambar 5.13) dan jarak antara house2 dan house3 adalah 4, dimana jarak tersebut sama dengan iterasi jarak sekarang, yaitu 4. Karena nilai tersebut telah sama, maka iterasi dapat dihentikan dan didapatkan bahwa jarak antara house2 dan house3 adalah 4.

Available : 4260864
AND
Nex[2][0][4] : 4473092
Available : 4194304

Gambar 5.13 house2 AND NEX[2][0][4]

Dari pencarian e diatas, didapatkan kombinasi sebagai berikut:

e(h1,h2) = 2
e(h1,h3) = 2
e(h2,h3) = 4

Dilihat dari kombinasi diatas, nilai maksimum e adalah 4 dengan pasangan house2 dan house3

Untuk mendapatkan nilai minimum d, maka dengan menggunakan langkah yang telah dilakukan sebelumnya. Dapat direpresentasikan pada Gambar 5.14

	0	1	2	3	4
A	H1				PO
B					
C	4				
D		2			
E			3		

	0	1	2	3	4
A	H1				PO
B					
C	3				
D		2			
E			4		

Gambar 5.14 Jarak d pada setiap petak

Sehingga didapatkan kombinasi untuk d tertinggi adalah
d(h1,h2)=6
d(h1,h3)=6

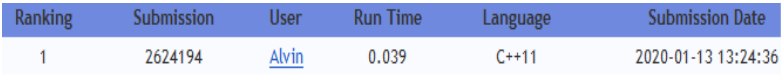
Dari kedua kombinasi diatas, nilai d terendah adalah 6

Uji coba kebenaran dilakukan dengan mengirimkan kode sumber hasil implementasi ke situs penilaian *online* SPOJ. Permasalahan yang ingin diselesaikan oleh penulis adalah *House Buying Optimizations* seperti yang telah dijelaskan pada bab 2. Setelah mengirimkan kode sumber, maka akan menerima umpan balik berupa hasil pengujian dari situs SPOJ.

Dari hasil uji coba kebenaran yang dilakukan ke situs SPOJ. Didapatkan umpan balik berupa status *Accepted* yang berarti solusi yang diajukan memberi jawaban yang benar. Waktu yang diperlukan program adalah 3.92 detik dan memori yang dibutuhkan program adalah 4.5MB.

Selain dari uji coba yang dilakukan pada situs SPOJ, penulis juga menemukan persoalan yang sama dengan format keluaran yang sedikit berbeda pada situs ICPC [4]. Dilakukan juga pengujian kebenaran dengan memasukkan kode sumber yang telah dimodifikasi sesuai dengan bentuk keluaran yang diminta dan dihasilkan umpan balik yaitu *Accepted*.

Gambar 5.15 Hasil uji coba kebenaran pada SPOJ



Gambar 5.16 Hasil uji coba kebenaran pada ICPC

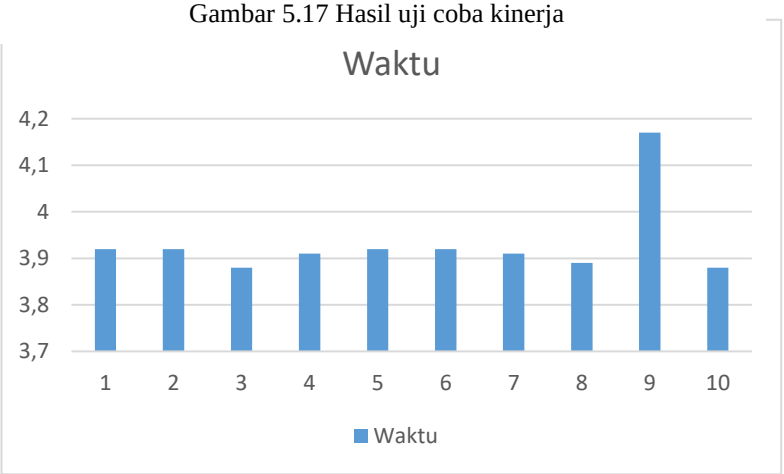
5.4 Uji Coba Kinerja

Kode sumber yang sama dikirimkan ke SPOJ hingga 10 kali untuk melihat perbedaan waktu dan memori yang dibutuhkan untuk menyelesaikan permasalahan. Hasil uji ditunjukkan pada

Gambar 5.17, sementara untuk sebaran waktu hasil uji ditunjukkan pada Gambar 5.18.

No	Verdict	Waktu (s)	Memori (M)	Bahasa
1	Accepted	3,92	4,5	C++
2	Accepted	3,92	4,5	C++
3	Accepted	3,88	4,4	C++
4	Accepted	3,91	4,4	C++
5	Accepted	3,92	4,4	C++
6	Accepted	3,92	4,6	C++
7	Accepted	3,91	4,6	C++
8	Accepted	3,89	4,7	C++
9	Accepted	4,17	4,6	C++
10	Accepted	3,88	4,7	C++

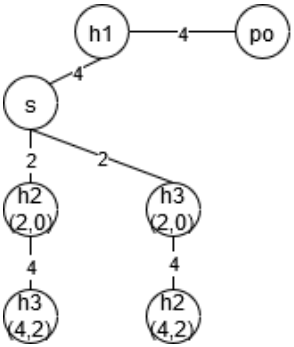
Gambar 5.17 Hasil uji coba kinerja



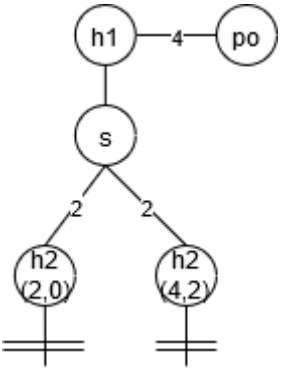
Gambar 5.18 Sebaran waktu uji coba

5.5 Analisis dan Kesimpulan Umum

Pada uji coba kinerja *pruning* yang dilakukan, didapatkan hasil bahwa *pruning* dapat memotong jumlah *node* yang akan diiterasi seperti pada Gambar 5.19 menjadi pada



Gambar 5.20 keseluruhan *tree* tanpa *pruning*



Gambar 5.19 *tree* dengan *pruning*

Kesimpulan secara umum yang didapatkan dari hasil percobaan diatas adalah metode *Pruning* berbasis *Bitwise* mampu mengurangi jumlah iterasi sehingga mempercepat waktu pencarian solusi yang didapat. Hal ini dibuktikan dengan

percobaan yang dilakukan pada situs *SPOJ* seperti pada Gambar 5.17 dan Gambar 5.18 yang memberikan *Verdict Accepted* dengan waktu kurang dari 8 detik. Dari kedua gambar tersebut, dapat dilihat bahwa rata-rata waktu yang dibutuhkan adalah 3.93 detik dan rata-rata memori yang dibutuhkan adalah 4.54MB.

[*Halaman ini sengaja dikosongkan*]

BAB VI
KESIMPULAN

Pada bab ini dijelaskan mengenai kesimpulan dari hasil uji coba yang telah dilakukan beserta saran untuk pengembangan lebih lanjut yang dapat dilakukan terhadap tugas akhir ini di masa yang akan datang.

6.1 Kesimpulan

Dari hasil uji coba yang telah dilakukan terhadap implementasi solusi untuk pemasalahan *House Buying Optimizations*, yang telah disarikan pada subbab 5.5, dapat diambil kesimpulan sebagai berikut:

1. *Pruning* berbasis *Bitwise* terbukti dapat mengurangi jumlah *node* yang akan ditelusuri pada pencarian menggunakan *Complete Search*.
2. Metode *Complete Search* dengan *Pruning* berbasis *Bitwise* mampu menyelesaikan permasalahan *HOUSEBUY* dengan rata-rata waktu 3,93 detik.
3. Kebenaran dari solusi yang diusulkan dapat dibuktikan benar dengan menerima umpan balik yang diberikan oleh situs *SPOJ*, yaitu berupa *Verdict Accepted*.

Implementasi dari metode tersebut mampu menyelesaikan permasalahan *House Buying Optimizations* dengan waktu rata-rata 3.93 detik dan memori yang dibutuhkan rata-rata 4.54MB. Dimana solusi tersebut mampu melewati batas yang ditetapkan dalam permasalahan.

6.2 Saran

Penyelesaian permasalahan *House Buying Optimizations* yang telah dibuat menggunakan metode *complete search* dengan *pruning* berbasis *bitwise* masih memiliki ruang untuk dikembangkan. Pengembangan implementasi bisa dilakukan dengan mengganti metode *pruning* maupun *bitwise* yang

mendukung operasi yang dibutuhkan, atau dengan algorittma lain yang lebih cepat.

DAFTAR PUSTAKA

[C. Xiaohong, "HOUSEBUY - House Buying Optimizations," [Online]. Available: https://www.spoj.com/problems/HOUSEBUY/.

[Minati De, Subhas C. Nandy dan Sasanka Roy, "Prune-and-Search with Limited Work-space," *Journal of Computer and System Sciences,* vol. 81, no. 2, pp. 398-414, March 2015.

["Arithmetic shift," [Online]. Available: https://en.wikipedia.org/wiki/Arithmetic_shift.

[ICPC, "2108 - Houses Divided," ICPC, [Online]. Available: https://icpcarchive.ecs.baylor.edu/index.php?option=com_onlinejudge&Itemid=8&page=problem_stats&problemid=109&category=4. [Diakses 13 01 2020].

[*Halaman ini sengaja dikosongkan*]

BIODATA PENULIS

Alvin Tanuwijaya, lahir di Surabaya pada tanggal 28 Juli 1998. Penulis merupakan anak kedua dari 2 bersaudara. Penulis telah menempuh pendidikan di SD Petra 10 Surabaya, SMP Petra 1 Surabaya, dan SMA Petra 1 Surabaya. Penulis melanjutkan pendidikan pada tingkat sarjana di Depatemen Infomatika, Institut Teknologi Sepuluh Nopember (ITS).

Selama masa studi S1 di ITS, penulis mengikuti beberapa organisasi dan kepanitiaan seperti PMK dan Schematics. Penulis memiliki ketertarikan dalam bidang merancang algoritma dan menyelesaikan permasalahan secara terstruktur. Dalam meningkatkan kemampuan, penulis memiliki pengalaman magang di PT Picodio. Apabila ingin berdiskusi lebih lanjut, dapat menghubungi penulis melalui email: alvin.tanuwijaya16@gmail.com.