



ITS
Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - KI141502
DESAIN DAN ANALISIS ALGORITMA
PENYELESAIAN PERMASALAHAN PENUGASAN
BERSYARAT DENGAN REPRESENTASI *BIPARTITE*
GRAPH

Muhammad Izzuddin
NRP 5112100012

Dosen Pembimbing I
Arya Yudhi Wijaya, S.Kom., M.Kom.
Dosen Pembimbing II
Rully Soelaiman, S.Kom., M.Kom.

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA 2016



ITS
Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - KI141502
DESAIN DAN ANALISIS ALGORITMA
PENYELESAIAN PERMASALAHAN PENUGASAN
BERSYARAT DENGAN REPRESENTASI *BIPARTITE*
GRAPH

Muhammad Izzuddin
NRP 5112100012

Dosen Pembimbing I
Arya Yudhi Wijaya, S.Kom., M.Kom.

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

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA 2016



FINAL PROJECT - KI141502

*DESIGN AND ANALYSIS OF ALGORITHM FOR
SOLVING CONDITIONAL ASSIGNMENT PROBLEM
USING BIPARTITE GRAPH REPRESENTATION*

Muhammad Izzuddin
NRP 5112100012

Supervisor 1
Arya Yudhi Wijaya, S.Kom. , M.Kom.

Supervisor 2
Rully Soelaiman, S.Kom. , M.Kom.

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY
SEPULUH NOPEMBER INSTITUTE OF TECHNOLOGY
SURABAYA 2016

LEMBAR PENGESAHAN

DESAIN DAN ANALISIS ALGORITMA PENYELESAIAN PERMASALAHAN PENUGASAN BERSYARAT DENGAN REPRESENTASI *BIPARTITE GRAPH*

TUGAS AKHIR

Diajukan untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Dasar Terapan & Komputasional
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh:

Muhammad Izzuddin
NRP. 5112 100 012

Disetujui oleh Pembimbing Tugas Akhir:

1. Arya Yudhi Wijaya, S.Kom., M.Kom.
NIP. 19840904 201012 1 002
(Pembimbing I)
2. Rully Soelaiman, S.Kom., M.Kom.
NIP. 19700213 199402 1 001
(Pembimbing II)

SURABAYA
JUNI, 2016

DESAIN DAN ANALISIS ALGORITMA PENYELESAIAN PERMASALAHAN PENUGASAN BERSYARAT DENGAN REPRESENTASI *BIPARTITE GRAPH*

Nama Mahasiswa : Muhammad Izzuddin
NRP : 5112 100 012
Jurusan : Teknik Informatika FTIf – ITS
Dosen Pembimbing I : Arya Yudhi W, S.Kom., M.Kom.
Dosen Pembimbing II : Rully Soelaiman, S.Kom., M.Kom.

ABSTRAK

Masalah penugasan adalah salah satu masalah optimasi untuk menemukan susunan penugasan pada sebuah bipartite graph. Permasalahan dalam tugas akhir ini adalah permasalahan penugasan bersyarat “Yet Another Assignment Problem” yang terdapat pada situs penilaian daring SPOJ dengan nomor soal 6819 dan kode soal ASSIGN5. Dalam permasalahan ini, diilustrasikan terdapat beberapa pekerjaan dan beberapa orang pekerja. Setiap pekerjaan harus dilakukan oleh semua orang yang ada serta setiap orang memiliki waktu yang dibutuhkan tersendiri dalam menyelesaikan pekerjaan tersebut. Setiap orang hanya dapat mengerjakan sebuah pekerjaan dan sebuah pekerjaan hanya dapat dikerjakan oleh satu orang dalam satu waktu. Pekerjaan yang dimaksud juga bersifat independen dalam artian dapat dilakukan kapanpun oleh setiap orang. Semua orang juga dapat berhenti kapanpun untuk melakukan sebuah pekerjaan tersebut.

Tugas akhir ini akan mengimplementasikan metode pencarian maximum-size matching pada sebuah bipartite graph yang mengacu pada permasalahan penugasan bersyarat. Metode pencarian maximum-size matching yang digunakan adalah algoritma Hopcroft-Karp.

Dalam buku ini akan dibahas algoritma Hopcroft-Karp untuk menyelesaikan permasalahan penugasan bersyarat tersebut dengan menggunakan bahasa pemrograman C++. Dari

serangkaian proses penelitian yang telah dilakukan, didapatkan kesimpulan bahwa algoritma yang dirancang sesuai dengan permasalahan ini dipengaruhi secara kuadratik baik oleh jumlah pekerjaan ataupun pekerjanya. Secara umum Algoritma ini memiliki kompleksitas $O((m + n)^2)$ dimana m adalah jumlah pekerjaan dan n adalah jumlah pekerja.

Kata kunci: algoritma Hopcroft-Karp, graf bipartite, masalah penugasan bersyarat, perfect matching, teori graf.

**DESIGN AND ANALYSIS OF ALGORITHM FOR SOLVING
CONDITIONAL ASSIGNMENT PROBLEM USING
BIPARTITE GRAPH REPRESENTATION**

Student Name : Muhammad Izzuddin
NRP : 5112 100 012
Major : Teknik Informatika FTIf – ITS
Supervisor I : Arya Yudhi W, S.Kom., M.Kom.
Supervisor II : Rully Soelaiman, S.Kom., M.Kom.

ABSTRACT

Assignment problem is one of the optimization problem to find a composition of assignment in a bipartite graph. This final project is based to “Yet Another Assignment Problem” on the online judge site SPOJ with problem number 6819 and problem code ASSIGN5. In this problem, illustrated that there are some jobs and also some workers. Each work must be done by everyone there and everyone has their own time taken in completing the work. Each person can only do a job, and a job can only be done by one person at a time. Work is also independent it means that can be done at any time by any person. Everyone also can stop at any time to do the job.

This final project will implement maximum-size matching finding method on a bipartite graph which refers to the conditional assignment problem. The methods in finding maximum-size matching that will be used for this problem is Hopcroft-Karp algorithm.

This book will implement Hopcroft-Karp algorithm to solve the conditional assignment problems using C++ programming language. From a series of processes of research that has been done, it was concluded that the algorithm that designed according the problem is quadratic influenced both by the number of jobs or workers. In genera term, this algorithm has a complexity of

$O((m + n)^2)$ where m is the number of jobs and n is the number of workers.

Key words: *bipartite graph, conditional assignment problem, graph theory, Hopcroft-Karp algorithm, perfect matching.*

DAFTAR ISI

LEMBAR PENGESAHAN.....	v
ABSTRAK	vii
ABSTRACT	ix
KATA PENGANTAR.....	xi
DAFTAR ISI.....	xiii
DAFTAR GAMBAR	xvii
DAFTAR TABEL	xxi
DAFTAR KODE SUMBER	xxiii
1. BAB I PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	2
1.3 Batasan Masalah.....	2
1.4 Tujuan.....	3
1.5 Metodologi	3
2. BAB II DASAR TEORI.....	7
2.1 Definisi Umum	7
2.2 <i>Bipartite Graph</i>	8
2.3 <i>Matching</i>	9
2.4 <i>Bipartite Matching</i>	12
2.5 Algoritma <i>Hopcroft-Karp</i>	15
2.6 Permasalahan <i>The dilemma of Idli</i> pada SPOJ	16
2.7 Penyelesaian Permasalahan <i>The dilemma of Idli</i> pada SPOJ	19
2.8 Permasalahan <i>Yet Another Assignment Problem</i> Pada SPOJ	20

2.9	Penyelesaian Permasalahan <i>Yet Another Assignment Problem</i>	24
3.	BAB III DESAIN	29
3.1	Desain penyelesaian Permasalahan <i>The dilemma of Idli</i>	29
3.1.1	Definisi Umum Sistem	29
3.1.2	Desain Algoritma.....	29
3.1.2.1	Desain Fungsi <i>InitGraph</i>	31
3.1.2.2	Desain Algoritma Hopcroft-Karp.....	31
3.2	Desain penyelesaian Permasalahan <i>Yet Another Assignment Problem</i>	33
3.2.1	Definisi Umum Sistem	33
3.2.2	Desain Algoritma.....	33
3.2.2.1	Desain Fungsi <i>ExpandingMatrix</i>	33
3.2.2.2	Desain Fungsi <i>Solve</i>	34
3.2.2.3	Desain Algoritma Hopcroft-Karp.....	36
3.2.3	Desain Data <i>Generator</i>	37
4.	BAB IV IMPLEMENTASI.....	39
4.1	Lingkungan Implementasi	39
4.2	Implementasi penyelesaian permasalahan <i>The Dilemma of Idli</i>	39
4.2.1	Penggunaan <i>Library</i> , Konstanta dan Variabel Global	39
4.2.2	Implementasi Fungsi <i>Main</i>	42
4.2.3	Implementasi Fungsi <i>InitGraph</i>	44
4.2.4	Implementasi Fungsi <i>BFS</i>	45
4.2.5	Implementasi Fungsi <i>DFS</i>	46

4.2.6	Implementasi Fungsi <i>HopcroftKarp</i>	46
4.3	Implementasi penyelesaian permasalahan <i>Yet Another Assignment Problem</i>	47
4.3.1	Penggunaan <i>Library, Template</i> , Konstanta dan Variabel Global	47
4.3.2	Implementasi Fungsi <i>Main</i>	50
4.3.3	Implementasi Fungsi <i>Solve</i>	51
4.3.4	Implementasi Fungsi <i>ExpandingMatrix</i>	52
4.3.5	Implementasi Fungsi <i>BFS</i>	53
4.3.6	Implementasi Fungsi <i>DFS</i>	54
4.3.7	Implementasi Fungsi <i>HopcroftKarp</i>	54
4.3.8	Implementasi Fungsi <i>Generate</i>	55
5.	BAB V UJI COBA DAN ANALISIS	57
5.1	Lingkungan Uji Coba	57
5.2	Skenario Uji Coba	57
5.2.1	Uji Coba Kebenaran	57
5.2.1.1	Uji Coba Kebenaran Penyelesaian Permasalahan <i>The Dilemma of Idli</i>	57
5.2.1.2	Uji Coba Kebenaran Penyelesaian Permasalahan <i>Yet Another Assignment Problem</i>	63
5.2.2	Uji Coba Kinerja Penyelesaian Permasalahan <i>Yet Another Assignment Problem</i>	76
5.2.2.1	Pengaruh Banyaknya Pekerjaan Terhadap Waktu	77
5.2.2.2	Pengaruh Banyaknya Pekerja Terhadap Waktu ..	78
6.	BAB VI KESIMPULAN DAN SARAN.....	81

6.1	Kesimpulan.....	81
6.2	Saran.....	81
LAMPIRAN A		83
DAFTAR PUSTAKA.....		85
BIODATA PENULIS.....		87

DAFTAR TABEL

Tabel 2.1. Tabel penyusunan tugas yang tidak optimal.	22
Tabel 2.2. Tabel penyusunan tugas secara optimal.	22
Tabel 2.3. Contoh permasalahan Yet Another Assignment Problem.	24
Tabel 4.1. Tabel Daftar Variabel Global permasalahan The Dilemma of Idli Bagian 1	40
Tabel 4.2. Tabel Daftar Variabel Global permasalahan The Dilemma of Idli Bagian 2	41
Tabel 4.3. Tabel Daftar Variabel Global Permasalahan Yet Another Assignment Problem Bagian 1	48
Tabel 4.4. Tabel Daftar Variabel Global Permasalahan Yet Another Assignment Problem Bagian 2	49
Tabel 5.1 Tabel deskripsi penduduk grup A	59
Tabel 5.2 Tabel deskripsi penduduk grup B.....	59
Tabel 5.3. Tabel Hasil Uji Coba pada Situs SPOJ Sebanyak 20 Kali	64
Tabel 5.4. Tabel pencarian nilai α	65
Tabel 5.5. Susunan penugasan tiap satuan waktu untuk kasus uji	75
Tabel 5.6. Tabel pengaruh jumlah pekerja terhadap waktu proses program	77
Tabel 5.7. Tabel pengaruh jumlah pekerja terhadap waktu proses program	79

DAFTAR GAMBAR

Gambar 2.1 Contoh graf bipartite (a) graf bipartite standar (b) complete bipartite graf.....	9
Gambar 2.2 Perbedaan kardinalitas dalam matching (a) graf bipartite (b) katching dengan kardinalitas 2 (c) Maximum-size matching.....	10
Gambar 2.3 Macam-macam alternating path	10
Gambar 2.4. Pembuktian Teorema Hall.....	13
Gambar 2.5 Teknik invertng (a) Bipartite graph awal (b) Matching setelah iterasi pertama (c) Matching setelah iterasi kedua (d) Matching setelah iterasi ketiga.....	14
Gambar 2.6. Deskripsi permasalahan The dilemma of Idli SPOJ...	17
Gambar 2.7. Contoh masukan dan keluaran permasalahan The dilemma of Idli.....	18
Gambar 2.8. Deskripsi permasalahan Yet Another Assignment Problem SPOJ	21
Gambar 2.9. Contoh format masukan dan keluaran.....	23
Gambar 2.10. Proses expanding-matrix (a) Matrix awal (b) Matrix setelah mengalami proses expanding-matrix.....	26
Gambar 3.1. Pseudocode Fungsi main The dilemma of Idli	30
Gambar 3.2. Pseudocode Fungsi InitGraph.....	31
Gambar 3.3. Pseudocode Fungsi BFS	32
Gambar 3.4. Pseudocode Fungsi DFS.....	32
Gambar 3.5. Pseudocode Fungsi HopcroftKarp.....	32
Gambar 3.6. Pseudocode Fungsi main Yet Another Assignment Problem	34
Gambar 3.7. Pseudocode Fungsi ExpandingMatrix	35
Gambar 3.8. Pseudocode Fungsi Solve	35
Gambar 3.9. Pseudocode Fungsi BFS	36
Gambar 3.10. Pseudocode Fungsi DFS.....	37

Gambar 3.11. Pseudocode Fungsi HopcroftKarp	37
Gambar 3.12. Pseudocode fungsi Generate	37
Gambar 5.1. Hasil Uji Coba pada Situs Penilaian Daring SPOJ permasalahan The Dilemma of Idli	57
Gambar 5.2 Format masukan uji kebenaran The Dilemma of Idli	58
Gambar 5.3. Bipartite graph hasil representasi permasalahan.....	60
Gambar 5.4 Proses pencarian maximum bipartite matching (a) proses pencarian pada iterasi pertama (b) proses pencarian pada iterasi kedua.....	61
Gambar 5.5. Tree yang dihasilkan dari proses pencarian iterasi kedua	61
Gambar 5.6. Keluaran sistem penyelesain permasalahan The Dilemma of Idli	62
Gambar 5.7. Hasil Uji Coba pada Situs Penilaian Daring SPOJ permasalahan Yet Another Assignment Problem.....	63
Gambar 5.8. Format masukan uji coba kebenaran.	65
Gambar 5.9. Proses expanding-matrix (a) Matrix awal (b) Matrix setelah mengalami proses expanding-matrix.....	66
Gambar 5.10. Pembentukan graf berdasarkan matrix awal (a) Bipartite graf iterasi ke-1 (b) Maximum matching pada graf iterasi ke-1	67
Gambar 5.11. Hasil matrix setelah proses reduksi nilai pada iterasi ke-1	68
Gambar 5.12. Pembentukan graf berdasarkan matrix iterasi ke-1 (a) Bipartite graf iterasi ke-2 (b) Maximum matching pada graf iterasi ke-2	68
Gambar 5.13. Hasil matrix setelah proses reduksi nilai pada iterasi ke-2	69

Gambar 5.14. Pembentukan graf berdasarkan matrix iterasi ke-2 (a) Bipartite graf iterasi ke-3 (b) Maximum matching pada graf iterasi ke-3	69
Gambar 5.15. Hasil matrix setelah proses reduksi nilai pada iterasi ke-3	70
Gambar 5.16. Pembentukan graf berdasarkan matrix iterasi ke-3 (a) Bipartite graf iterasi ke-4 (b) Maximum matching pada graf iterasi ke-4	70
Gambar 5.17. Hasil matrix setelah proses reduksi nilai pada iterasi ke-4	71
Gambar 5.18. Pembentukan graf berdasarkan matrix iterasi ke-4 (a) Bipartite graf iterasi ke-5 (b) Maximum matching pada graf iterasi ke-5	71
Gambar 5.19. Hasil matrix setelah proses reduksi nilai pada iterasi ke-5	72
Gambar 5.20. Pembentukan graf berdasarkan matrix iterasi ke-5 (a) Bipartite graf iterasi ke-6 (b) Maximum matching pada graf iterasi ke-6	72
Gambar 5.21. Hasil matrix setelah proses reduksi nilai pada iterasi ke-6	73
Gambar 5.22. Pembentukan graf berdasarkan matrix iterasi ke-6 (a) Bipartite graf iterasi ke-7 (b) Maximum matching pada graf iterasi ke-7	73
Gambar 5.23. Hasil matrix setelah proses reduksi nilai pada iterasi ke-7	74
Gambar 5.24. Pembentukan graf berdasarkan matrix iterasi ke-7 (a) Bipartite graf iterasi ke-8 (b) Maximum matching pada graf iterasi ke-8	74
Gambar 5.25. Hasil matrix setelah proses reduksi nilai pada iterasi ke-8	75
Gambar 5.26. Masukan dan Keluaran pada Program	76

Gambar 5.27. Grafik pengaruh jumlah pekerjaan terhadap waktu proses program	78
Gambar 5.28. Grafik pengaruh jumlah pekerja terhadap waktu proses program	79
Gambar A.1. Hasil Uji Coba pada Situs SPOJ Sebanyak 20 Kali.	83

DAFTAR KODE SUMBER

Kode Sumber 4.1. Potongan Kode penggunaan Library dan Konstanta The Dilemma of Idli.....	40
Kode Sumber 4.2. Potongan Kode Penggunaan Variabel Global permasalahan The Dilemma of Idli.....	42
Kode Sumber 4.3. Potongan Kode Implementasi Fungsi Main The Dilemma of Idli Bagian 1	42
Kode Sumber 4.4. Potongan Kode Implementasi Fungsi Main The Dilemma of Idli Bagian 2	43
Kode Sumber 4.5. Potongan Kode Implementasi Fungsi InitGraph	44
Kode Sumber 4.6. Potongan Kode Implementasi Fungsi BFS. ..	45
Kode Sumber 4.7. Potongan Kode Implementasi Fungsi DFS. ..	46
Kode Sumber 4.8. Potongan Kode Implementasi Fungsi HopcroftKarp.	46
Kode Sumber 4.9. Potongan Kode penggunaan Library dan Konstanta Yet Another Assignment Problem	47
Kode Sumber 4.10. Potongan Kode Penggunaan Variabel Global Permasalahan Yet Another Assignment Problem.	49
Kode Sumber 4.11. Penggunaan Template getNum Pada Program Permasalahan Yet Another Assignment Problem.	50
Kode Sumber 4.12. Potongan Kode Implementasi Fungsi Main Permasalahan Yet Another Assignment Problem	51
Kode Sumber 4.13. Potongan Kode Implementasi Fungsi Solve Permasalahan Yet Another Assignment Problem.	51
Kode Sumber 4.14. Potongan Kode Implementasi Fungsi ExpandingMatrix.....	52
Kode Sumber 4.15. Potongan Kode Implementasi Fungsi BFS.	53
Kode Sumber 4.16. Potongan Kode Implementasi Fungsi DFS.	54

Kode Sumber 4.17. Potongan Kode Implementasi Fungsi HopcroftKarp.....	54
Kode Sumber 4.18. Kode Implementasi Generator Data	55

BAB I

PENDAHULUAN

Pada bab ini akan dijelaskan bagian dasar dalam penyusunan tugas akhir ini. Penjelasan diawali dengan latar belakang, rumusan masalah, batasan masalah, tujuan, dan metodologi pengerjaan tugas akhir. Terakhir, dijelaskan mengenai sistematika laporan tugas akhir.

1.1 Latar Belakang

Kemajuan teknologi informasi saat ini berjalan dengan begitu pesat terutama pada hal proses komputasi yang kini dapat semakin cepat dilakukan. Perkembangan ini tentunya diikuti dan didukung oleh semakin berkembangnya ilmu pengetahuan yang mempelajari tentang teknologi dan penerapannya. Salah satu rumpun yang mempelajari hal tersebut adalah ilmu komputer.

Teori graf adalah salah satu cabang matematika dan ilmu komputer yang penting dan banyak dikembangkan. Seiring dengan berjalannya perkembangan bidang komputasi serta dalam hal ini berkembangnya perangkat lunak maupun perangkat keras komputer, teori graf banyak dijadikan model dalam memecahkan masalah yang ada dalam kehidupan kita sehari-hari. Graf digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut. Banyak persoalan pada dunia nyata yang sebenarnya merupakan representasi visual dari graf. Banyak hal yang dapat digali dari representasi tersebut, diantaranya adalah menentukan jalur terpendek dari satu tempat ke tempat lain, menggambarkan 2 kota yang bertetangga dengan warna yang berbeda pada peta, menentukan tata letak jalur transportasi, pengaturan jaringan komunikasi atau jaringan internet dan masih banyak lagi.

Salah satu topik yang menarik dalam penerapan teori graf ini adalah permasalahan penugasan. Masalah penugasan adalah salah satu masalah optimasi yang merupakan cabang dari ilmu riset operasi. Secara umum masalah penugasan membahas bagaimana

menemukan susunan pemberian tugas pada pekerja agar sebuah pekerjaan tersebut dapat dilakukan dengan seoptimal mungkin. Terdapat beberapa algoritma yang sudah dikembangkan untuk menyelesaikan permasalahan penugasan ini seperti algoritma Hungarian dan Hopcroft-Karp. Dalam kehidupan nyata permasalahan penugasan dapat diterapkan dalam menyelesaikan masalah transportasi atau bahkan dalam bidang kedokteran dan kesehatan sekalipun.

Pada tugas akhir ini, penulis mengangkat topik penugasan bersyarat dengan representasi graf *bipartite* pada permasalahan SPOJ Klasik 6819 Yet Another Assignment Problem (ASSIGN5).

1.2 Rumusan Masalah

Perumusan masalah yang terdapat pada Tugas Akhir ini, antara lain adalah:

1. Bagaimana menyelesaikan permasalahan penugasan bersyarat pada SPOJ Klasik 6819 Yet Another Assignment Problem (ASSIGN5) dengan representasi graf bipartit?
2. Bagaimana mengkonstruksi dan membuktikan kebenaran algoritma Hopcroft-Karp dengan menggunakan permasalahan SPOJ Klasik 19295 The Dilemma of Idli (WPC5G)?
3. Bagaimana hasil efisiensi waktu penyelesaian permasalahan penugasan bersyarat pada SPOJ Klasik 6819 Yet Another Assignment Problem (ASSIGN5) dengan representasi graf bipartit?

1.3 Batasan Masalah

Batasan masalah yang terdapat pada Tugas Akhir ini, yaitu sebagai berikut:

1. Implementasi algoritma menggunakan bahasa pemrograman C++.
2. Dalam setiap *testcase*, jumlah *node* yang merepresentasikan banyak pekerjaan maksimal sebanyak 2000 buah.

3. Dalam setiap *testcase*, jumlah *node* yang merepresentasikan banyak orang maksimal sebanyak 2000 buah.
4. Setiap *node* pekerjaan memiliki *edge* yang terhubung dengan semua *node* orang yang melakukan pekerjaan tersebut.
5. Setiap *edge* yang menghubungkan pekerjaan dengan orang yang melakukan pekerjaan memiliki nilai yang merepresentasikan waktu yang dibutuhkan untuk melakukan pekerjaan tersebut dengan nilai maksimal 10^6 satuan.

1.4 Tujuan

Tujuan dari pembuatan Tugas Akhir ini antara lain sebagai berikut:

1. Melakukan desain serta analisis penyelesaian permasalahan penugasan bersyarat pada SPOJ Klasik 6819 Yet Another Assignment Problem (ASSIGN5) dengan representasi graf bipartit.
2. Menganalisis hasil kinerja algoritma yang dibangun dalam menyelesaikan permasalahan penugasan bersyarat pada SPOJ Klasik 6819 Yet Another Assignment Problem (ASSIGN5) dengan representasi graf bipartit.

1.5 Metodologi

Ada beberapa tahap dalam proses pengerjaan tugas akhir ini, yaitu sebagai berikut:

1. Studi Literatur
Pada tahap ini, akan dilakukan studi mengenai permasalahan penugasan dengan representasi graf *bipartite* serta algoritma penyelesaiannya, yaitu algoritma Hopcroft-Karp. Sumber studi yang digunakan antara lain buku-buku literatur, paper, situs-situs pembelajaran *online*, serta materi perkuliahan yang terkait.

2. Implementasi

Pada tahap ini, akan dilakukan pembangunan dari algoritma yang telah dipelajari menjadi sebuah sistem yang dapat digunakan. Bahasa pemrograman yang digunakan adalah C++ dengan menggunakan IDE (*Integrated Development System*) Dev-C++.

3. Uji coba dan evaluasi

Tahap ini merupakan tahap pengujian aplikasi dengan data masukan yang telah ditentukan untuk menguji kebenaran hasil implementasi algoritma serta menguji waktu eksekusi aplikasi untuk data masukan yang telah ditentukan. Pada tahap ini juga dilakukan optimasi dari hasil implementasi apabila aplikasi masih kurang efisien.

4. Penyusunan buku tugas akhir

Tahap ini merupakan tahap penyusunan laporan berupa buku tugas akhir sebagai dokumentasi pelaksanaan tugas akhir yang mencakup seluruh teori, implementasi serta hasil pengujian yang telah dikerjakan.

Sistematika Laporan

Laporan tugas akhir ini dibagi menjadi 6 bab yang masing-masing menjelaskan bagian-bagian yang berbeda, yaitu:

1. Bab I, Pendahuluan, berisi penjelasan mengenai latar belakang, rumusan masalah, batasan masalah, tujuan penulisan tugas akhir, dan sistematika penulisan laporan.
2. Bab II, Dasar Teori, berisi penjelasan teori-teori yang digunakan sebagai dasar pengerjaan tugas akhir ini..
3. Bab III, Desain, berisi rancangan pembuatan sistem penyelesaian permasalahan pada tugas akhir ini..
4. Bab IV, Implementasi, berisi lingkungan serta hasil penerapan rancangan sistem penyelesaian permasalahan dalam tugas akhir ini dalam bentuk sumber kode beserta penjelasannya.
5. Bab V, Uji Coba dan Analisis, berisi rangkaian uji coba yang dilakukan untuk menguji kebenaran dan efisiensi program.

6. Bab VI, Penutup, berisi kesimpulan pengerjaan tugas akhir dan saran untuk pengembangan tugas akhir.

BAB II DASAR TEORI

Bab ini akan membahas mengenai dasar teori yang menjadi dasar pengerjaan tugas akhir ini.

2.1 Definisi Umum

Pada subbab ini akan di dibahas istilah dan definisi umum yang sering digunakan dalam buku ini. Pembahasan dalam subbab ini dimaksudkan agar pembaca dapat lebih mudah memahami isi dari buku ini yang mana akan banyak dibahas istilah dalam *domain* teori graf.

Sebuah graf $G = (V, E)$ terdiri dari dua buah himpunan V dan E . Setiap elemen V disebut sebagai *vertex* atau simpul sedangkan elemen dari himpunan E disebut sebagai *edge* atau sisi. Setiap *edge* dalam graf G merupakan pasangan dari dua buah *vertex* dengan notasi (u, v) yang menandakan hubungan antara *vertex* u dengan *vertex* v . Kedua *vertex* dalam *edge* (u, v) tersebut dinamakan *endpoint*. Sebuah *edge* dapat menghubungkan *vertex* yang sama. Sebagai contoh, himpunan V dapat berisi $\{a, b, c, d, e\}$ dan himpunan E dapat berisi $\{\{a, b\}, \{b, c\}, \{c, d\}, \{d, e\}, \{e, a\}, \{a, a\}\}$.

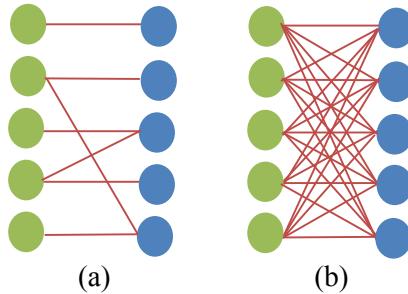
Graf tidak berarah merupakan sebuah graf dimana *edge* (u, v) dianggap sama dengan *edge* (v, u) sedangkan Graf berarah merupakan sebuah graf dimana *edge* (u, v) dianggap tidak sama dengan *edge* (v, u) . Dalam graf berarah, sebuah *edge* (u, v) berarti *vertex* v dapat diakses dari *vertex* u , serta *edge* (v, u) berarti *vertex* u dapat diakses dari *vertex* v . Sebuah *vertex* dikatakan bersinggungan dengan sebuah *edge* apabila *vertex* tersebut merupakan *endpoint* dari *edge* yang dimaksud. Dalam kata lain apabila *vertex* v merupakan *endpoint* dari *edge* e maka *vertex* v dikatakan bersinggungan dengan *edge* e . Derajat dari sebuah *vertex* adalah banyaknya *edge* yang bersinggungan dengan *vertex* tersebut.

Jika *vertex* u serta *vertex* v merupakan endpoint dari sebuah *edge* e , maka *vertex* u adalah tetangga *vertex* v begitu pula sebaliknya, *vertex* v adalah tetangga *vertex* u , sehingga dapat dikatakan *vertex* u dan *vertex* v dihubungkan oleh *edge* e .

Path dalam istilah graf adalah deretan *vertex* dimana tiap *vertex* yang bersebelahan adalah *endpoint* dari sebuah *edge* dan *edge* tersebut tidak boleh muncul lebih dari satu kali. *Cycle* adalah *path* yang diawali dan diakhiri oleh *vertex* yang sama. *Connected component* adalah subgraf $H = (P, Q)$ dimana untuk tiap pasang *vertex* u dan v pada P terdapat setidaknya satu *path* yang berawal di u dan berakhir di v atau berawal di v atau sebaliknya. *Symmetric difference* dari dua buah himpunan *edge* E_1 dan E_2 dengan notasi $E_1 \oplus E_2$ adalah semua *edge* yang berada hanya pada salah satu himpunan E_1 atau E_2 saja.

2.2 *Bipartite Graph*

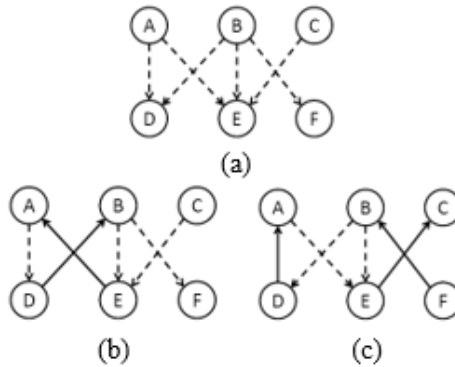
Bipartite graph merupakan graf spesial yang *vertex*-nya dapat dibagi menjadi dua himpunan *vertex* X dan Y dimana pada setiap himpunan *vertex*-nya, tidak ada *vertex* yang saling bertetangga. Contoh dari *bipartite graph* ditunjukkan pada Gambar 2.1(a) Dalam buku ini *bipartite graph* dinotasikan sebagai $G = (X \cup Y, E)$ dimana X dan Y merupakan himpunan *vertex* yang membentuk graf seperti pada penjelasan sebelumnya dan E merupakan himpunan dari *edge*. Dalam kasus khusus, *bipartite graph* dapat membentuk sebuah *complete bipartite graph*. *Complete bipartite graph* adalah sebuah *bipartite graph* dimana tiap *vertex* pada satu himpunan, bersinggungan dengan semua *vertex* pada himpunan yang lainnya. Contoh dari *bipartite graph* ditunjukkan pada Gambar 2.1(b). *Regular bipartite graph* adalah sebuah istilah dalam *bipartite* graf yang digunakan apabila sebuah graf $G = (X \cup Y, E)$ memiliki derajat yang sama untuk *vertex*-nya. *k-regular bipartite graph* berarti setiap *vertex* pada graf tersebut memiliki derajat sebanyak k .



Gambar 2.1 Contoh graf bipartite (a) graf bipartite standar (b) complete bipartite graf

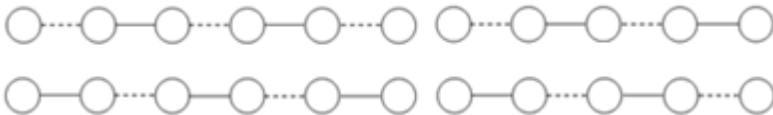
2.3 Matching

Sebuah *matching* pada graf $G = (V, E)$ adalah sebuah himpunan *edge* $M \subseteq E$ dimana tidak ada *edge* yang saling bersentuhan di dalam M . Untuk setiap *matching* M pada graf $G = (V, E)$, kumpulan *edge* $e \in M$ dinamakan *matched edge*, sedangkan kumpulan *edge* $e \in E - M$ dinamakan *unmatched edge*. Sebuah *vertex* dikatakan *matched vertex* apabila *vertex* tersebut bersinggungan dengan *matched edge* pula. Sebaliknya *unmatched vertex* adalah *vertex* yang tidak bersinggungan dengan *matched edge*. Tiap *matched vertex* v memiliki sebuah *mate* atau pasangan yang terletak pada *endpoint* dari *matched edge* yang bersinggungan dengan v . Sebuah *matching* dapat dikatakan sebagai *perfect matching* apabila semua *edge* pada *matching* M dalam graf $G = (V, E)$ mencakup setiap *vertex* dalam graf atau tiap *vertex* dalam V bersinggungan dengan tepat satu *edge* pada M . Kardinalitas dari *matching* M adalah banyaknya *edge* pada M dan biasa dinotasikan dengan $|M|$.



Gambar 2.2 Perbedaan kardinalitas dalam matching (a) graf bipartite (b) matching dengan kardinalitas 2 (c) Maximum-size matching

Maximum-size matching adalah sebuah *matching* M pada graf $G = (V, E)$ yang kardinalitasnya tidak dapat dinaikkan lagi atau memiliki $|M|$ terbesar. Pada gambar 2.2, terdapat beberapa contoh *matching* pada sebuah graf, garis tegas menyatakan *matched edge* sedangkan garis putus-putus menyatakan *unmatched edge*. Gambar 2.2(a) menunjukkan *bipartite graph* tanpa terdapat *matching* didalamnya. Gambar 2.2(b) menunjukkan *matching* dengan kardinalitas 2 dari graf yang sama sedangkan Gambar 2.2(c) menunjukkan *matching* dengan kardinalitas 3 dari graf pada Gambar 2.2(a). *Matching* pada Gambar 2.2(c) tidak dapat dinaikkan lagi kardinalitasnya dikarenakan semua *vertex* pada graf sudah dicakup oleh *matched edge* oleh karena itu *matching* ini disebut *maximum-size matching*.



Gambar 2.3 Macam-macam alternating path

Alternating path dari sebuah *matching* M adalah *path* dimana urutan *edge*-nya terdiri dari *matched edge* dan *unmatched edge* secara bergantian. Beberapa macam *alternating path* yang dapat dibentuk dapat dilihat pada Gambar 2.3. *Augmenting path* dari sebuah *matching* M adalah *alternating path* yang dimulai dari sebuah *unmatched vertex* dan berakhir pada *unmatched vertex* pula.

Teorema 2.1. (Teorema Berge) *Matching* M dalam sebuah graf G disebut sebagai *maximum-size matching* jika dan hanya jika tidak ada *augmenting path* yang dapat dibentuk dari *matching* M .

Pembuktian. Misalkan dalam sebuah graf terdapat *maximum-size matching* M , apabila terdapat *augmenting path* P yang dapat terbentuk dari *matching* M , maka M bukan merupakan *maximum-size matching* dikarenakan pasti terdapat *matching* lain yang terbentuk dari $N = M \oplus P$ dan sudah dipastikan $|N| > |M|$. Hal tersebut bersifat kontradiksi dengan Teorema 2.1. Sebaliknya, apabila *matching* M bukan merupakan *maximum-size matching* maka terdapat *matching* N yang merupakan *maximum-size matching*. Diasumsikan subgraf H adalah graf yang edgenya diperoleh dengan menghilangkan *edge* yang sama dari *union* himpunan *matching* M dan N disimbolkan dengan $M \oplus N = (M \cup N) - (M \cap N)$ atau dalam hal ini adalah *symmetric difference* dari M dan N . Dikarenakan M dan N merupakan sebuah *matching*, maka *vertex* pada subgraf $H = (V, M \oplus N)$ sudah dipastikan memiliki derajat tidak lebih dari 2. Hal ini menunjukkan bahwa *connected component* pada subgraf H dapat berupa *alternating path* atau *alternating cycle* (sebuah *alternating cycle* yang berawal dan berakhir pada *vertex* yang sama). *Path* dan *cycle* yang terbentuk merupakan deretan *edge* yang bergantian pada *matching* M dan N . Dikarenakan *matching* N memiliki *edge* yang lebih banyak dari *matching* M maka hal ini juga berpengaruh memberikan *edge* N yang lebih banyak pada himpunan $M \oplus N$ sehingga setidaknya pasti terdapat satu *path* Q yang berawal dan

berakhir pada *edge* dalam *matching* N . *Path* Q yang terbentuk merupakan augmenting path yang dapat dibentuk dari *matching* M dikarenakan *vertex* pertama dan terakhirnya tidak berada pada *matching* M .

Teorema Berge yang disebutkan pada Teorema 2.1 merupakan teorema yang menjadi dasar algoritma pencarian *maximum-size matching* dengan menggunakan *augmenting path* yang dibentuk dari sebuah *matching* pada sebuah graf.

2.4 *Bipartite Matching*

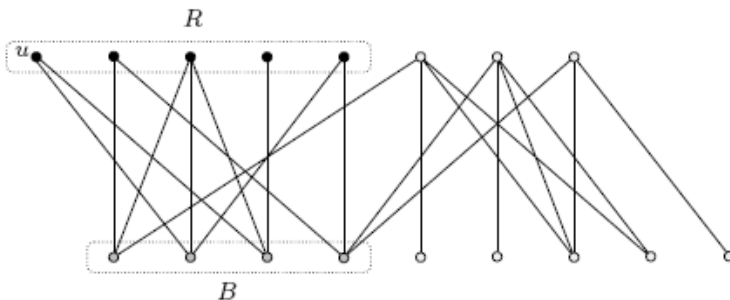
Bipartite matching merupakan kasus *matching* khusus yang melibatkan struktur *bipartite graph* didalamnya. *Bipartite graph* $G = (X \cup Y, E)$ dapat memiliki sebuah *complete matching* dengan kardinalitas $\min\{|X|, |Y|\}$. Salah satu teorema yang dapat digunakan dalam pencarian *complete matching* adalah teorema yang diajukan oleh Philip Hall. Sebelum membahas teorema ini, terlebih dulu disepakati bahwa didefinisikan S sebagai *subset* dari graf $G = (X \cup Y, E)$. Himpunan dari semua *vertex* yang bersinggungan dengan *vertex* pada S di notasikan sebagai $N(S)$.

Teorema 2.2. (Teorema Hall) Sebuah *bipartite graph* $G = (X \cup Y, E)$ memiliki sebuah *matching* yang mencakup semua *vertex* pada X (*complete matching*) jika dan hanya jika $|N(S)| \geq |S|$ untuk semua $S \subseteq X$.

Pembuktian. Misal terdapat graf $G = (X \cup Y, E)$ yang mempunyai *matching* M dan mencakup seluruh *vertex* dalam X atau memiliki *complete matching*, maka untuk sembarang $S \subseteq X$, masing-masing *vertex* dalam S terdapat pada M dan dipasangkan pada $N(S)$ yang berbeda. Dari hal tersebut dapat ditarik kesimpulan bahwa $|N(S)| \geq |S|$. Sebaliknya, jika G tidak memiliki *complete matching* dan N adalah *maximum-size matching* pada graf G maka terdapat *vertex* u pada X yang tidak berada pada *matching* N . Misal Z adalah himpunan *vertex* yang terdapat pada

semua *alternating path* yang dapat dibentuk dimulai dari *vertex* u sesuai dengan *matching* N . Maka menurut Teorema 2.1, u adalah satu-satunya *vertex* di Z yang tidak berada di dalam *matching* N . *Vertex* di dalam Z dibagi menjadi dua himpunan, yaitu $R = X \cap Z$ dan $B = Y \cap Z$ seperti pada Gambar 2.4. Terlihat pada gambar tersebut, *vertex* pada $R - \{u\}$ berada pada N dan berpasangan dengan *vertex* pada B . Oleh sebab itu $|R| - 1 = |B|$ dan $N(R) \supseteq B$. Dalam hal ini B adalah $N(R)$ karena semua *vertex* yang berada pada B terhubung dengan u melalui *alternating path*. Melalui persamaan yang telah didapatkan tersebut, dapat disimpulkan bahwa $|N(R)| = |B| = |R| - 1$ dan terjadi kontradiksi.

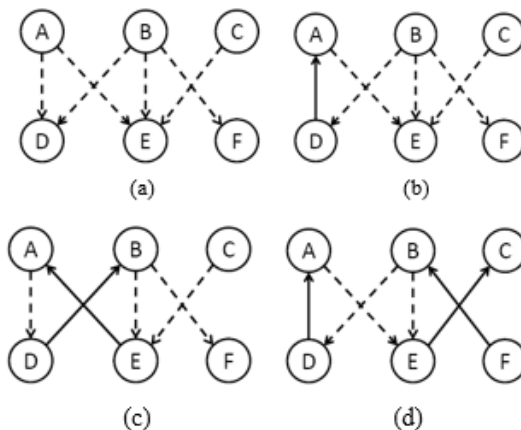
Berdasarkan Teorema 2.1, *maximum-size matching* dapat dibentuk secara iteratif. Dimulai dengan *matching* kosong, $M = \emptyset$, kemudian dilanjutkan dengan mencari *augmenting path* dari *matching* yang sekarang sudah di dapatkan. Pada setiap iterasinya didapatkan sebuah *matching* baru yang memiliki kardinalitas $|M_{i+1}| = |M_i| + 1$ dengan cara menjadikan *unmatched edge* pada *augmenting path* yang telah didapatkan sebagai *matched edge* dan juga sebaliknya. Iterasi ini akan terus dilakukan hingga tidak ada lagi *augmenting path* yang dapat dibentuk dari *matching* yang terbaru. Untuk setiap iterasi yang di lakukan, pencarian *augmenting path* dilakukan dengan menggunakan *breadth-first*



Gambar 2.4. Pembuktian Teorema Hall.

search. Metode pencarian *maximum-size matching* seperti ini dinamakan *inverting* atau pembalikan *augmenting path*. Dalam kasus *complete bipartite* graf $G = (V, E)$, setiap *vertex* dapat memiliki hingga sebanyak n^2 *edge* yang terhubung dengannya dimana $n = |V|$. Melalui *breadth-first search*, *augmenting path* dapat diperoleh dalam kompleksitas waktu $O(n^2)$. Secara keseluruhan algoritma *inverting augmenting path* ini memiliki kompleksitas waktu $O(n^3)$ untuk membentuk *maximum-size matching*.

Pada Gambar 2.5, ditunjukkan *matching* yang ditemukan dalam setiap iterasi *inverting*. Gambar 2.5(a) menunjukkan kondisi awal dari *bipartite graph* yang akan dicari *maximum-size matching*-nya. Pada saat iterasi pertama dilakukan pencarian *augmenting path* yang dimulai dari *vertex* A dan menghasilkan *augmenting path* yang berisi $\{A, D\}$ dan setelah *inverting*, *matching* berisi *edge* $\{A, D\}$ hal ini ditunjukkan oleh Gambar 2.5(b). Pada iterasi kedua



Gambar 2.5 Teknik *inverting* (a) *Bipartite graph* awal (b) *Matching* setelah iterasi pertama (c) *Matching* setelah iterasi kedua (d) *Matching* setelah iterasi ketiga

yang ditunjukkan oleh Gambar 2.5(c), pencarian *augmenting path* dimulai dari *vertex B*, kemudian didapat *augmenting path* $\{\{B, D\}, \{D, A\}, \{A, E\}\}$ dan setelah *inverting* didapat *matching* $\{\{B, D\}, \{A, E\}\}$. Pada Gambar 2.5(d), pencarian pada iterasi ketiga dimulai dari *vertex C* dan menghasilkan *augmenting path* $\{\{C, E\}, \{E, A\}, \{A, D\}, \{D, B\}, \{B, F\}\}$ sehingga dihasilkan *matching* $\{\{C, E\}, \{A, D\}, \{B, F\}\}$.

2.5 Algoritma Hopcroft-Karp

Algoritma *Hopcroft-Karp* adalah algoritma yang memerlukan *bipartite graph* sebagai masukannya dan menghasilkan *maximum-size matching* didalam graf tersebut sebagai keluarannya.

Algoritma ini ditemukan John Hopcroft and Richard Karp pada tahun 1973. Sebagian besar ide dasar dari algoritma ini hampir sama dengan teknik *inverting augmenting path* dalam mencari *maximum-size matching*, hanya saja algoritma ini mencari beberapa *augmenting path* sekaligus dalam sekali iterasi dengan menggunakan himpunan maksimal dari *augmenting path* terpendek.

Secara keseluruhan, algoritma ini berjalan pada sebuah graf $G = (X \cup Y, E)$, dimana $|V|$ adalah jumlah *vertex* yang berada pada $X \cup Y$ dan $|E|$ adalah jumlah *edge* pada graf, dengan kompleksitas $O(|V|\sqrt{|E|})$.

Dimisalkan terdapat sebuah graf $G = (X \cup Y, E)$ dan M adalah *matching* yang di dapat dari graf tersebut. Secara garis besar, algoritma ini berjalan sebagai berikut:

- Sebuah *breadth-first search* digunakan untuk membagi *vertex-vertex* menjadi beberapa *layer*. Sebuah *layer* dalam hal ini diartikan sebagai sebuah *level* dalam *tree*, dan setiap *level* dalam *tree* menunjukkan panjang augmented path yang

dapat di bentuk. *Unmatched vertex* pada X digunakan sebagai *vertex* awal dari pencarian.

- Pada awal pencarian, hanya terdapat *unmatched edge* dikarenakan M masih merupakan himpunan kosong. Setelah itu, pada pencarian selanjutnya, melalui *unmatched vertex* pada X , pencarian path dilakukan dengan men-*reverse unmatched edge dan matched edge* secara bergantian. Pencarian akan berhenti pada sebuah layer- k dimana terdapat satu atau beberapa *unmatched vertex* pada Y tercapai.
- Setelah layer pada graf tersebut terbentuk, melalui *depth-first search*, sebuah himpunan maksimal dari *augmenting path* terpendek yang memiliki panjang k dapat dicari dengan awal *unmatched vertex* pada X dan berakhir pada *unmatched vertex* Y .
- Untuk setiap *augmenting path* yang ditemukan, semua *unmatched vertex* pada tersebut diubah menjadi *matched vertex* begitu pula sebaliknya.

Algoritma ini berakhir apabila pada proses pencarian *breadth-first search* tidak menemukan *augmented path* yang dapat dibentuk.

2.6 Permasalahan *The dilemma of Idli* pada SPOJ

Pada subbab ini akan dijelaskan mengenai permasalahan yang digunakan untuk membuktikan kebenaran dari algoritma Hopcroft-Karp yang dirancang. Contoh permasalahan yang dapat diselesaikan dengan algoritma Hopcroft-Karp adalah permasalahan pada situs penilaian daring SPOJ dengan judul permasalahan *The dilemma of Idli* dan kode soal WPC5G. Deskripsi soal ditunjukkan oleh Gambar 2.6.

Sphere online judge **PROBLEMS** **STATUS** **RANK**

Problems / classical / The dilemma of Idli

WPC5G - The dilemma of Idli

no tags

It is emergency time! There has been a clash between the organizers of the Galactical Wars and the Inter-Galactic Parliament due to some feud. The entire civilization has been split into two groups A and B. The civilians in A love some particular members in the Organization Committee of the Galactical Wars, and hate some members from the Inter-Galactic Parliament. On the other hand, civilians in B love some particular members from the Inter-Galactic Parliament, and hate some in the Organization Committee.

You are Idli, the Inter-Galactic Dean, and it is your job to ensure satisfaction of the civilians. You decide to do so by impeaching some members from the Organization Committee and some from the Parliament. A civilian is satisfied if and only if all the members he loves are not impeached, and all the members he hates are impeached.

Of course, Idli wants to satisfy most number of civilians. Help Idli in devising an algorithm to do so.

Gambar 2.6. Deskripsi permasalahan *The dilemma of Idli* SPOJ

Didalam soal tersebut di deskripsikan terdapat sebuah negeri yang penduduknya terbagi menjadi 2 bagian yaitu penduduk A dan B. Dikarenakan sebuah kepentingan, penduduk A menyukai beberapa orang tertentu dari grup *Committee of the Galactical Wars*, dan membenci beberapa orang tertentu dari grup *Inter-Galactic Parliament*. Sebaliknya, penduduk B menyukai beberapa orang tertentu dari grup *Inter-Galactic Parliament* dan membenci beberapa orang dari grup *Committee of the Galactical Wars*.

Idli merupakan pimpinan tertinggi di negeri tersebut, bagaimanapun juga Idli harus memastikan masyarakatnya puas dengan semua keputusannya. Dia memutuskan untuk mencurigai beberapa orang dari *Committee of the Galactical Wars* serta *Inter-Galactic Parliament* agar masyarakatnya merasa tenang dan puas. Seorang penduduk dapat dikatakan puas jika dan hanya jika semua orang yang dia suka, tidak dicurigai dan semua orang yang dia benci, dicurigai. Idli menyadari bahwa akan sulit untuk

memuaskan semua penduduknya, sehingga ia mencari cara agar jumlah penduduk yang merasa puas maksimal.

Format masukan diawali dengan dua buah bilangan n_1 dan n_2 yang masing-masing mewakili jumlah dari anggota *Committee of the Galactical Wars* yang ada (dilambangkan oleh angka 1 hingga n_1) dan jumlah anggota *Inter-Galactic Parliament* (dilambangkan oleh angka 1 hingga n_2). Selanjutnya terdapat sebuah nilai n yang mendeskripsikan banyaknya penduduk yang ada dan diikuti oleh n baris yang berisi deskripsi penduduk tersebut dengan ketentuan sebagai berikut:

- Karakter pertama berisi huruf 'A' atau 'B' yang menandakan dari kelompok mana penduduk tersebut berasal dan diikuti oleh sebuah spasi.
- Beberapa angka yang dipisahkan oleh spasi dan diakhiri oleh angka -1 yang menunjukkan anggota pada grup yang sama dan disukai oleh penduduk tersebut.
- Beberapa angka yang dipisahkan oleh spasi dan diakhiri oleh angka -1 yang menunjukkan anggota pada grup yang berbeda dan dibenci oleh penduduk tersebut.

```

Input:
5 5
3
A 1 -1 5 3 -1
B 2 3 -1 2 5 -1
B -1 -1

Output:
2

```

Gambar 2.7. Contoh masukan dan keluaran permasalahan *The dilemma of Idli*.

Format keluaran berisi sebuah nilai yang menunjukkan berapa banyak penduduk yang dapat dipuaskan oleh Idli. Contoh dari format masukan dan keluaran dapat dilihat pada Gambar 2.7.

Beberapa batasan yang terdapat pada permasalahan *Yet Another Assignment Problem* adalah sebagai berikut:

1. $1 \leq n1 \leq 1000000$.
2. $1 \leq n2 \leq 1000000$.
3. $1 \leq n \leq 500$.
4. $0 \leq$ banyaknya anggota *CoGW/IGP* yang disukai oleh seorang penduduk ≤ 50 .
5. $0 \leq$ banyaknya anggota *CoGW/IGP* yang dibenci oleh seorang penduduk ≤ 50 .
6. Batas Waktu 1 detik.
7. Batas sumber kode 50000 B.
8. Batas Memori: 1536 MB.

2.7 Penyelesaian Permasalahan *The dilemma of Idli* pada SPOJ

Pada subbab ini akan dijelaskan bagaimana menyelesaikan permasalahan *The dilemma of Idli* dengan penjelasan permasalahan seperti pada subbab sebelumnya.

Pada deskripsi soal tersebut diketahui bahwa seorang penduduk dikatakan puas jika dan hanya jika semua orang yang dia suka, tidak dicurigai dan semua orang yang dia benci, dicurigai oleh Idli. Dengan menegaskan premis tersebut didapatkan bahwa seorang penduduk dikatakan tidak puas jika terdapat salah satu orang yang dia suka dicurigai atau terdapat salah satu orang yang dia benci, tidak dicurigai. Dimisalkan memilih memuaskan seorang penduduk A, sebagai konsekuensinya, semua penduduk B yang menyukai orang yang dibenci oleh A merasa tidak puas, begitu juga sebaliknya.

Dimisalkan terdapat n orang penduduk A dan m orang penduduk B. Permasalahan ini dapat direpresentasikan dengan membuat sebuah *Bipartite Graph* $G = (A \cup B, E)$ dimana A adalah setiap orang dari penduduk A merupakan dan B adalah setiap orang dari penduduk B. Untuk setiap orang yang dibenci penduduk A_i tetapi disukai penduduk B_i ataupun sebaliknya, dibuat sebuah *edge* yang menghubungkan penduduk A_i dan penduduk B_i tersebut. Selanjutnya akan dicari *Maximum Bipartite Matching* dengan menggunakan algoritma Hopcroft-Karp pada *bipartite graph* yang telah terbentuk. *Maximum Bipartite Matching* pada graf tersebut akan mengembalikan nilai t yaitu total *vertex* minimum pada A dan B yang dapat diambil dari hubungan *edge* yang ada atau dengan kata lain banyaknya orang yang tidak dapat dipuaskan. Jumlah total orang yang puas adalah total dari penduduk yang ada dikurangi oleh *Maximum Bipartite Matching* yang ditemukan. Atau dengan kata lain

$$\text{Max Penduduk Puas} = m + n - t.$$

2.8 Permasalahan *Yet Another Assignment Problem* Pada SPOJ

Permasalahan yang diangkat dalam Tugas Akhir ini bersumber dari suatu situs penilaian daring atau *online judge* SPOJ yaitu *Yet Another Assignment Problem* dengan kode soal ASSIGN5, deskripsi soal dapat dilihat pada Gambar 2.8.

Didalam permasalahan ini diceritakan bahwa terdapat suatu permasalahan penugasan namun dengan beberapa kondisi khusus. Permasalahan dalam penugasan tersebut didefinisikan sebagai berikut:

- Terdapat beberapa pekerjaan dan beberapa orang yang akan melakukan pekerjaan tersebut.



PROBLEMS STATUS R/

Problems / classical / Yet Another Assignment Problem

Advertisement blocking software were detected :(Please add this webpage to whitelist.

ASSIGN5 - Yet Another Assignment Problem

no tags

New semester is coming. As the class monitor, Cathy Yin is going to make necessary preparations. She has m jobs to do, n classmates are going to help her. Each job requires some classmates working on it for certain time, say the i -th classmate must work on the j -th job for A_{ij} minutes. As an Oler of great responsibility she wishes to finish all jobs as soon as possible. But a classmate can do only one job at a time, and two classmates can **not** do the same job at the same moment. For example, to decorate the classroom, Alpha must work on it for 3 minutes **plus** Beta works on it for 4 minutes, then one possible assignment will be ABABBAB, taking 7 minutes in total.

Now she is going to make a detailed schedule specifying who is doing what at each moment. Jobs are independent and may be done in any order. Also for each job anyone can do it for arbitrarily long, but **not** longer than the required time A_{ij} . Anyone can be free at any time. Time for certain classmate doing certain work need **not** be consecutive.

As her friend, you are to help her to work out the schedule minimizing the total time needed.

Gambar 2.8. Deskripsi permasalahan Yet Another Assignment Problem SPOJ

- Setiap pekerjaan harus dilakukan oleh semua orang yang ada serta setiap orang memiliki waktu yang dibutuhkan tersendiri dalam menyelesaikan pekerjaan tersebut.
- Setiap orang hanya dapat mengerjakan sebuah pekerjaan dan sebuah pekerjaan hanya dapat dikerjakan oleh satu orang dalam satu satuan waktu.
- Pekerjaan yang dimaksud juga bersifat independen dalam artian dapat dilakukan kapanpun oleh setiap orang.
- Semua orang juga dapat berhenti kapanpun untuk melakukan sebuah pekerjaan tersebut.

Permasalahan tersebut mengharuskan untuk menemukan kombinasi penugasan yang cocok pada setiap satuan waktunya agar semua pekerjaan tersebut dapat diselesaikan dengan optimal.

Tabel 2.1. Tabel penyusunan tugas yang tidak optimal.

Menit	Alpha	Beta
1	Membersihkan kelas	Mendekor kelas
2	Membersihkan kelas	Istirahat
3	Membersihkan kelas	Mendekor kelas
4	Membersihkan kelas	Mendekor kelas
5	Membersihkan kelas	Mendekor kelas
6	Mendekor kelas	Membersihkan kelas
7	Mendekor kelas	Istirahat
8	Istirahat	Mendekor kelas

Sebagai suatu contoh untuk mendekorasi sebuah kelas, Alpha harus mengerjakan hal tersebut selama 2 menit ditambah Beta harus mengerjakan hal tersebut selama 5 menit dan untuk membersihkan kelas, Alpha harus bekerja selama 5 menit ditambah Beta harus bekerja selama 1 menit. Salah satu penugasan dapat dibuat sebagaimana yang ditunjukkan oleh Tabel 2.1

Dengan penyusunan tugas seperti pada Tabel 2.1 tersebut diperlukan waktu sebanyak 8 menit untuk menyelesaikan semua tugas yang ada. Waktu tersebut bukan merupakan waktu yang paling efisien sehingga terdapat cara penugasan lain yang dapat dilakukan sehingga pekerjaan dapat dilakukan dengan lebih singkat seperti pada tabel 2.2.

Tabel 2.2. Tabel penyusunan tugas secara optimal.

Menit	Alpha	Beta
1	Mendekor kelas	Istirahat
2	Membersihkan kelas	Mendekor kelas
3	Membersihkan kelas	Mendekor kelas
4	Membersihkan kelas	Mendekor kelas
5	Membersihkan kelas	Mendekor kelas
6	Membersihkan kelas	Mendekor kelas
7	Mendekor kelas	Membersihkan kelas

Dengan susunan penugasan Pada tabel 2.2, pekerjaan yang dilakukan dapat menjadi lebih optimal.

Format masukan diawali dengan dua buah bilangan m dan n yang masing-masing mewakili jumlah pekerjaan yang ada dan jumlah teman yang akan melakukan pekerjaan tersebut. Selanjutnya diikuti oleh m baris yang mendeskripsikan lama pengerjaan suatu pekerjaan tiap barisnya. Setiap baris i berisi n buah bilangan dimana bilangan ke j -nya adalah A_{ij} yang berarti teman ke- j harus melakukan pekerjaan i selama A_{ij} menit sesuai dengan deskripsi soal.

Format keluaran diawali bilangan T yang menandakan waktu minimal yang diperlukan untuk melakukan semua pekerjaan tersebut. Baris berikutnya berisi n buah bilangan yang merupakan representasi salah satu penugasan pada menit ke 0 yang mungkin dimana setiap bilangan ke- i mendeskripsikan pekerjaan mana yang dilakukan teman ke- i dan berisi 0 apabila tidak sedang melakukan pekerjaan apapun. Contoh dari format masukan dan keluaran dapat dilihat pada Gambar 2.9.

Example

Input:

2 2

2 5

5 1

Output:

7

1 0

Gambar 2.9. Contoh format masukan dan keluaran.

Beberapa batasan yang terdapat pada permasalahan *Yet Another Assignment Problem* adalah sebagai berikut:

1. $1 \leq m \leq 2000$
2. $1 \leq n \leq 2000$
3. $1 \leq A_{ij} \leq 10^6$
4. Batas Waktu 0.132 - 0.661 detik.
5. Batas sumber kode 50000 B.
6. Batas Memori: 1536 MB.

2.9 Penyelesaian Permasalahan *Yet Another Assignment Problem*.

Pada subbab ini akan dijelaskan bagaimana menyelesaikan permasalahan *Yet Another Assignment Problem* dengan penjelasan permasalahan seperti pada subbab sebelumnya.

Sesuai pada deskripsi masalah, dapat diketahui bahwa setiap orang harus mengerjakan semua pekerjaan yang ada sesuai dengan waktu penyelesaiannya masing-masing dan setiap pekerjaan harus dikerjakan oleh semua orang dengan waktu penyelesaian masing-masing pula. Dimisalkan terdapat 2 buah pekerjaan (m) yang selanjutnya disebut dengan baris dan 3 orang pekerja (n) yang selanjutnya disebut dengan kolom seperti pada Tabel 2.3.

Total waktu yang dibutuhkan bagi seseorang ke- i untuk melakukan semua pekerjaannya adalah penjumlahan setiap nilai pada kolom ke- i tersebut. Sedangkan total waktu yang dibutuhkan

Tabel 2.3. Contoh permasalahan Yet Another Assignment Problem.

	Pekerja 1	Pekerja 2	Pekerja 3
Pekerjaan 1	2	2	4
Pekerjaan 2	1	2	0

agar semua orang dapat menyelesaikan pekerjaan ke- j adalah penjumlahan semua nilai pada baris ke- j tersebut.

Melalui pernyataan tersebut, waktu minimum yang diperlukan untuk melakukan semua pekerjaan yang ada adalah jumlah waktu terbanyak yang dilakukan seseorang untuk menyelesaikan semua pekerjaannya atau waktu terbanyak yang dibutuhkan oleh sebuah pekerjaan agar diselesaikan oleh semua orang. Misal waktu minimum yang dibutuhkan untuk menyelesaikan semua pekerjaan adalah α , maka:

$$\alpha = \text{Max}(\sum_{i=0}^{m-1, n-1} A_{i,j}, \sum_{i=0}^{n-1, m-1} A_{j,i})$$

Pernyataan α sebagai waktu optimal yang diperlukan untuk menyelesaikan semua pekerjaan dapat dibuktikan apabila mempertimbangkan dua kasus berikut:

1. Terdapat baris yang memiliki jumlah yang maksimal. Apabila hal tersebut terjadi, maka baris tersebut harus di proses setidaknya sebanyak α kali. Karena hanya ada satu baris yang dipasangkan dengan satu kolom dalam satu waktu, tidak mungkin mendapatkan waktu penyelesaian kurang dari α . Selanjutnya yang dilakukan adalah menunjukkan bahwa bisa didapatkan solusi optimal dengan tepat α satuan waktu.
2. Terdapat kolom yang memiliki jumlah yang maksimal. Dikarenakan hanya satu orang yang dapat mengerjakan satu pekerjaan pada satu waktu serta pekerjaan dan pekerja tersebut dapat ditukar waktu pengerjaannya maka penyusunan dapat dilakukan dengan menukar saja posisi pekerjaan dan pekerjanya, kemudian mempertimbangkan kasus 1.

Untuk menyusun penyelesaian permasalahan ini dibentuk sebuah *matrix* awal yang berukuran $m * n$. Melalui *matrix* awal tersebut terlebih dahulu dicari nilai α sesuai dengan persamaan yang dijelaskan sebelumnya. Selanjutnya dibentuk sebuah *matrix*

2	2	4
1	2	0

(a)

2	2	4	0	0
1	2	0	0	5
5	0	0	3	0
0	4	0	4	0
0	0	4	1	3

(b)

Gambar 2.10. Proses *expanding-matrix* (a) *Matrix* awal (b) *Matrix* setelah mengalami proses *expanding-matrix*

dengan ukuran $(n + m) * (m + n)$ dan mengisi $m * n$ *matrix* pertama dalam *matrix* baru tersebut sesuai dengan *matrix* awal. Kemudian mengisi semua baris dan kolom yang belum terisi dengan sebuah nilai sedemikian rupa sehingga setiap baris dan kolom memiliki jumlah tepat α . Proses ini kemudian dinamakan dengan nama *expanding-matrix* seperti pada Gambar 2.10. Gambar 2.10(a) menunjukkan keadaan *matrix* awal yang sudah dibentuk berdasarkan permasalahan, kemudian pada Gambar 2.10(b) ditunjukkan hasil *matrix* yang sudah melalui proses *expanding matrix*.

Secara umum algoritma yang digunakan untuk menyelesaikan permasalahan ini berjalan sebagai berikut :

1. Membentuk *bipartite* graf $G = (X \cup Y, E)$ dengan X merupakan representasi dari kolom (pekerja) dan Y merupakan representasi dari baris (pekerjaan) dari *matrix* yang baru. Sebuah *edge* pada E menghubungkan *vertex* pada X dan Y apabila nilai $A_{i,j}$ pada *matrix* tersebut tidak 0.

2. Mencari *maximum matching* yang terdapat pada graf tersebut. Apabila sebuah kolom i memiliki pasangan dengan baris j , artinya pekerja ke- i mengerjakan pekerjaan ke- j dalam suatu waktu tersebut. Apabila $j > m$ maka pekerja tersebut tidak melakukan pekerjaan apapun di satuan waktu tertentu.
3. Setiap baris j dan kolom i yang berpasangan, kurangi nilai $A_{i,j}$ pada matrix tersebut sebanyak 1.
4. Ulangi setiap proses 1 – 3 hingga matrix tersebut menjadi *matrix* dengan semua isinya adalah 0.

Pada setiap proses yang terjadi, sebuah *perfect matching* terbentuk melalui graf tersebut. Dikarenakan *matrix* adalah sebuah *matrix* persegi dengan ukuran $(m + n) * (n + m)$, maka semua baris memiliki pasangan pada kolom, begitu pula dengan kolom yang seluruhnya memiliki pasangan pada baris. Seperti yang dijelaskan sebelumnya, diketahui bahwa jumlah dari semua elemen *matrix* tersebut adalah $(m + n) * \alpha$ dan setiap kali proses jumlah dari *matrix* tersebut berkurang sebanyak $m + n$, maka total proses yang diperlukan untuk menjadikan *matrix* tersebut menjadi *matrix* 0 adalah tepat sebanyak α .

Berdasarkan deskripsi permasalahan, keluaran yang diminta adalah konfigurasi atau urutan penugasan yang mungkin pada menit awal sehingga satu kali proses pencarian *perfect matching* pada graf tersebut sudah mewakili solusi pada permasalahan ini. Dengan menggunakan algoritma Hopcroft-Karp yang dijelaskan pada subbab 2.5, pencarian *perfect matching* dapat diselesaikan dengan kompleksitas $O(|V|\sqrt{|E|})$. Dengan representasi graf diatas, banyaknya *edge* $|E|$ yang mungkin terbentuk adalah maksimal $(m + n)^2$ dengan banyaknya *vertex* $|V|$ pada $X \cup Y$ adalah $2 * (m + n)$, sehingga kompleksitas total dengan representasi graf tersebut adalah $O((m + n)^2)$.

BAB III DESAIN

Pada bab ini akan dijelaskan desain algoritma yang akan digunakan dalam pengerjaan Tugas Akhir ini.

3.1 Desain penyelesaian Permasalahan *The dilemma of Idli*

Pada subbab ini akan dijelaskan desain penyelesaian permasalahan *The dilemma of Idli*.

3.1.1 Definisi Umum Sistem

Sistem akan menerima masukan berupa dua buah bilangan n_1 dan n_2 yang masing-masing mewakili jumlah dari anggota *Committee of the Galactical Wars* yang ada (dilambangkan oleh angka 1 hingga n_1) dan jumlah anggota *Inter-Galactic Parliament* (dilambangkan oleh angka 1 hingga n_2). Selanjutnya sistem menerima masukan sebuah nilai n yang mendeskripsikan banyaknya penduduk yang ada dan diikuti oleh n baris yang berisi deskripsi penduduk tersebut sesuai dengan format yang ditentukan pada subbab 2.6 sebelumnya. Kemudian, sistem akan memodelkan masukan tersebut sebagai sebuah *bipartite graph* dengan memanggil fungsi *InitGraph*. Setelah graf tersebut terbentuk, sistem akan mencari *Maximum Bipartite Matching* m yang terdapat pada graf tersebut lalu mengeluarkan sebuah nilai $n - m$ sebagai hasil keluaran program yang berarti jumlah penduduk maksimal yang dapat dipuaskan oleh Idli. Pseudocode dari fungsi *main* ditunjukkan oleh Gambar 3.1.

3.1.2 Desain Algoritma

Sistem terdiri dari dua fungsi utama, yaitu fungsi *InitGraph* serta fungsi *HopcroftKarp*. Pada subbab ini akan dijelaskan desain algoritma dari kedua fungsi tersebut.

```

main()
1. input n1
2. input n2
3. input n
4. for i ← 1 to n
5.     input tmp
6.     if tmp == 'A'
7.         while true
8.             input tmp2
9.             if tmp2 == -1
10.                break
11.                Push(grupA[counterA][0],tmp2)
12.        while true
13.            input tmp2;
14.            if tmp2 == -1
15.                break
16.                Push(grupA[counterA][1],tmp2)
17.        counterA++
18.    else
19.        while true
20.            input tmp2
21.            if tmp2 == -1
22.                break
23.                Push(personBL[tmp2],counterB)
24.        while true
25.            input tmp2
26.            if tmp2 == -1
27.                break
28.                Push(personBH[tmp2],counterB)
29.        counterB++
30.
31. InitGraph()
32. HopcroftKarp()
33. output n - maximum matching found

```

Gambar 3.1. Pseudocode Fungsi main The dilemma of Idli

InitGraph ()
1. for i ← 0 to counterA - 1
2. for j ← 0 to SizeOf(grupA[i][0]) - 1
3. for k ← 0 to SizeOf(personBH[grupA[i][0][j]]) - 1
4. Push(G[i+1, personBH[grupA[i][0][j]][k] + 501)
5. for j ← 0 to SizeOf(grupA[i][1]) - 1
6. for k ← 0 to SizeOf(personBL[grupA[i][1][j]]) - 1
7. if personBL[grupA[i][1][j]][k]+501 not in G[i+1]
8. Push(G[i+1], personBL[grupA[i][1][j]][k] + 501)

Gambar 3.2. Pseudocode Fungsi InitGraph

3.1.2.1 Desain Fungsi *InitGraph*

Fungsi *InitGraph* digunakan untuk membentuk *Bipartite Graph* $G = (A \cup B, E)$ dimana A adalah setiap orang dari penduduk A merupakan dan B adalah setiap orang dari penduduk B . Untuk setiap orang yang dibenci penduduk A_i tetapi disukai penduduk B_i ataupun sebaliknya, dibuat sebuah *edge* yang menghubungkan penduduk A_i dan penduduk B_i tersebut. *Pseudocode* dari fungsi *InitGraph* ditunjukkan oleh Gambar 3.2.

3.1.2.2 Desain Algoritma Hopcroft-Karp

Algoritma Hopcroft-Karp bertugas untuk mencari *perfect matching* yang mungkin didapat dari graf yang telah tersedia. Fungsi BFS digunakan untuk mencari *augmented path* serta fungsi DFS untuk mencari himpunan maksimal dari *augmenting path* terpendek pada *augmenting path* yang ada kemudian menjadikan *unmatched vertex* pada *path* tersebut sebagai *matched vertex* dan begitu pula sebaliknya. Fungsi HopcroftKarp sendiri akan memanggil fungsi BFS secara terus menerus hingga tidak ditemukan *augmented path* pada graf tersebut. Desain Fungsi HopcroftKarp terdapat pada Gambar 3.5, sedangkan Desain fungsi BFS terdapat pada Gambar 3.3 dan Desain fungsi DFS pada Gambar 3.4.

3.1.2.2.1 Desain Fungsi BFS

BFS()
1. for each u in U
2. if Pair_U[u] == ZERO
3. Dist[u] \leftarrow 0
4. Enqueue(Q,u)
5. else
6. Dist[u] \leftarrow ∞
7. Dist[ZERO] \leftarrow ∞
8. while Empty(Q) == false
9. u = Dequeue(Q)
10. if Dist[u] < Dist[ZERO]
11. for each v in Adj[u]
12. if Dist[Pair_V[v]] == ∞
13. Dist[Pair_V[v]] = Dist[u] + 1
14. Enqueue(Q,Pair_V[v])
15. return Dist[NIL] != ∞

Gambar 3.3. Pseudocode Fungsi BFS

3.1.2.2.2 Desain Fungsi DFS

DFS(u)
1. if u != ZERO
2. for each v in Adj[u]
3. if Dist[Pair_V[v]] == Dist[u] + 1
4. if DFS(Pair_V[v]) == true
5. Pair_V[v] = u
6. Pair_U[u] = v
7. return true
8. Dist[u] = ∞
9. return false
10. return true

Gambar 3.4. Pseudocode Fungsi DFS

3.1.2.2.3 Desain Fungsi HopcroftKarp

HopcroftKarp()
1. while BFS() == true
2. for each u in U
3. if Pair_U[u] == ZERO and DFS(u)
4. matching++
5. return matching

Gambar 3.5. Pseudocode Fungsi HopcroftKarp

3.2 Desain penyelesaian Permasalahan *Yet Another Assignment Problem*

Pada subbab ini akan dijelaskan mengenai desain penyelesaian permasalahan *Yet Another Assignment Problem*.

3.2.1 Definisi Umum Sistem

Sistem akan menerima masukan berupa banyaknya jumlah pekerjaan dan pekerja dalam permasalahan tersebut lengkap dengan waktu yang diperlukan setiap pekerja dalam mengerjakan suatu pekerjaan. Setelah itu sistem akan memodelkan masukan tersebut menjadi sebuah *matrix* dan melakukan perhitungan terhadap jumlah dari setiap baris dan kolom serta jumlah seluruh elemen dalam *matrix*. Sistem akan melakukan proses *expanding-matrix* terlebih dahulu. Berdasarkan *matrix* baru yang telah dibuat, sistem akan mencari sebuah *perfect matching* yang dapat dibentuk dan mengeluarkan waktu minimal yang diperlukan untuk menyelesaikan semua pekerjaan tersebut serta sistem akan mengeluarkan konfigurasi atau susunan penugasan yang mungkin berdasarkan *perfect matching* tersebut. *Pseudocode* fungsi *main* ditunjukkan pada Gambar 3.6.

3.2.2 Desain Algoritma

Sistem terdiri dari dua fungsi utama, yaitu fungsi *ExpandingMatrix* serta fungsi *Solve*. Pada subbab ini akan dijelaskan desain algoritma dari kedua fungsi tersebut.

3.2.2.1 Desain Fungsi *ExpandingMatrix*

Fungsi *ExpandingMatrix* digunakan untuk membentuk *matrix* dengan ukuran $(m + n) * (n + m)$ dimana semua baris dan kolomnya mempunyai jumlah yang sama yaitu α seperti pada penjelasan pada subbab 2.9. *Matrix* ini nantinya digunakan sebagai dasar pembentukan *bipartite graph*. *Pseudocode* dari fungsi *ExpandingMatrix* ditunjukkan oleh Gambar 3.7.

```

main()
1. input number of job
2. input number of student
3. for i = 1 to number of job
4.     for j = 1 to number of student
5.         input coresponding time for student j to
           finish job i
6.             calculate sum of row i based on input
7.             calculate sum of column j based on input
9. ExpandingMatrix()
9. Solve()
10. Output minimum time needed to finish all job
11. for i = 1 to number of student
12.     if match of student i > number of job
13.         output 0
14.     else
15.         output match of student i
16. return

```

Gambar 3.6. Pseudocode Fungsi main Yet Another Assignment Problem

3.2.2.2 Desain Fungsi Solve

Fungsi *Solve* digunakan untuk mencari waktu minimum yang diperlukan untuk menyelesaikan semua pekerjaan yang ada. Secara umum, fungsi ini akan membentuk sebuah *bipartite graph* dari matrix yang telah ada sesuai penjelasan pada subbab 2.9. Setelah *bipartite graph* terbentuk, fungsi ini akan memanggil sebuah fungsi untuk mencari *maximum matching* yang terdapat pada graf tersebut.

Algoritma yang akan digunakan untuk mencari *maximum matching* pada graf tersebut adalah algoritma Hopcroft-Karp yang didalamnya terdapat 3 fungsi utama yaitu fungsi HopcorftCarp, BFS, dan DFS. *Pseudocode* untuk fungsi solve terdapat pada Gambar 3.8.

ExpandingMatrix()	
1.	maxSum \leftarrow 0
2.	for i \leftarrow 0 to M
3.	maxSum \leftarrow MAX(sumRow[i],maxSum)
4.	for i \leftarrow 0 to N
5.	maxSum \leftarrow MAX(sumCol[i],maxSum)
6.	i \leftarrow 0
7.	for j \leftarrow N to M + N
8.	matrix[i,j] \leftarrow maxSum - sumRow[i])
9.	sumRow[i] \leftarrow sumRow[i] + matrix[i,j]
10.	sumCol[j] \leftarrow sumCol[j] + matrix[i,j]
11.	i \leftarrow i + 1
12.	j \leftarrow 0
13.	for i \leftarrow M to M + N
14.	matrix[i,j] \leftarrow maxSum - sumCol[j])
15.	sumRow[i] \leftarrow sumRow[i] + matrix[i,j]
16.	sumCol[j] \leftarrow sumCol[j] + matrix[i,j]
17.	j \leftarrow j + 1
18.	for i \leftarrow M to M + N
19.	for j \leftarrow N to M+N
20.	matrix[i,j] \leftarrow maxSum - MAX(sumRow[i],sumCol[j])
21.	sumRow[i] \leftarrow sumRow[i] + matrix[i,j]
22.	sumCol[j] \leftarrow sumCol[j] + matrix[i,j]

Gambar 3.7. Pseudocode Fungsi ExpandingMatrix

Solve()	
1.	step \leftarrow maxSum
2.	for i \leftarrow 0 to M + N
3.	for j \leftarrow 0 to M + N
4.	if matrix[x,j] > 0
5.	push(G[j+1] , (i+1) + (M+N))
6.	HopcroftKarp()
7.	return step

Gambar 3.8. Pseudocode Fungsi Solve

3.2.2.3 Desain Algoritma Hopcroft-Karp

Algoritma Hopcroft-Karp bertugas untuk mencari *perfect matching* yang mungkin didapat dari graf yang telah tersedia. Fungsi BFS digunakan untuk mencari *augmented path* serta fungsi DFS untuk mencari himpunan maksimal dari *augmenting path* terpendek pada augmenting path yang ada kemudian menjadikan *unmatched vertex* pada path tersebut sebagai *matched vertex* dan begitu pula sebaliknya. Fungsi HopcroftKarp sendiri akan memanggil fungsi BFS secara terus menerus hingga tidak ditemukan *augmented path* pada graf tersebut. Desain Fungsi HopcroftKarp terdapat pada Gambar 3.11, sedangkan Desain fungsi BFS terdapat pada Gambar 3.9 dan Desain fungsi DFS pada Gambar 3.10.

3.2.2.3.1 Desain Fungsi BFS

BFS()	
1.	for each u in U
2.	if $\text{Pair_U}[u] == \text{ZERO}$
3.	$\text{Dist}[u] \leftarrow 0$
4.	$\text{Enqueue}(Q, u)$
5.	else
6.	$\text{Dist}[u] \leftarrow \infty$
7.	$\text{Dist}[\text{ZERO}] \leftarrow \infty$
8.	while $\text{Empty}(Q) == \text{false}$
9.	$u = \text{Dequeue}(Q)$
10.	if $\text{Dist}[u] < \text{Dist}[\text{ZERO}]$
11.	for each v in $\text{Adj}[u]$
12.	if $\text{Dist}[\text{Pair_V}[v]] == \infty$
13.	$\text{Dist}[\text{Pair_V}[v]] = \text{Dist}[u] + 1$
14.	$\text{Enqueue}(Q, \text{Pair_V}[v])$
15.	return $\text{Dist}[\text{NIL}] != \infty$

Gambar 3.9. Pseudocode Fungsi BFS

3.2.2.3.2 Desain Fungsi DFS

DFS(u)
1. if u != ZERO
2. for each v in Adj[u]
3. if Dist[Pair_V[v]] == Dist[u] + 1
4. if DFS(Pair_V[v]) == true
5. Pair_V[v] = u
6. Pair_U[u] = v
7. return true
8. Dist[u] = ∞
9. return false
10. return true

Gambar 3.10. Pseudocode Fungsi DFS

3.2.2.3.3 Desain Fungsi HopcroftKarp

HopcroftKarp()
1. while BFS() == true
2. for each u in U
3. if Pair_U[u] == ZERO
4. DFS(u)
5. return

Gambar 3.11. Pseudocode Fungsi HopcroftKarp

3.2.3 Desain Data Generator

Data generator digunakan untuk menghasilkan format masukan sesuai dengan deskripsi permasalahan seperti yang telah dijelaskan pada subbab 2.8. *Pseudocode* dari data generator ditunjukkan pada Gambar 3.12.

Generate()
1. input m
2. input n
3. output m
4. output n
5. for i ← 0 to m
6. for i ← 0 to n
7. A _{ij} ← random % 100
8. output A _{ij}

Gambar 3.12. Pseudocode fungsi Generate

BAB IV IMPLEMENTASI

Pada bab ini akan dibahas tentang implementasi yang dilakukan berdasarkan apa yang telah dirancang pada bab sebelumnya.

4.1 Lingkungan Implementasi

Lingkungan implementasi yang digunakan adalah sebagai berikut :

1. Perangkat Keras:
Processor: Intel(R) Core(TM) i3-2130 @ 3.30 GHz.
Random Access Memory (RAM): 6.00 GB.
2. Perangkat Lunak:
Operating System: Windows 8 Professional 64 bit.
IDE : Orwell Bloodshed Dev-C++ 5.9.2.
Compiler: g++ (TDM-GCC 4.8.1 32-bit).
Bahasa Pemrograman: C++.

4.2 Implementasi penyelesaian permasalahan *The Dilemma of Idli*

Pada subbab ini akan dijelaskan mengenai implementasi dari penyelesaian permasalahan *The Dilemma of Idli* sesuai dengan desain yang telah dibuat pada subbab 3.1.

4.2.1 Penggunaan *Library*, Konstanta dan Variabel Global

Pada subbab ini akan dibahas penggunaan *library*, *template*, konstanta dan variabel global yang digunakan dalam sistem. Berikut adalah potongan kode yang menyatakan penggunaan *library* dan konstanta. Pada Kode sumber 4.1 terdapat lima *library* yang dipanggil antara lain: *cstdio*, *vector*, *queue*, *algorithm* serta *cstring*. Didefinisikan konstanta *MAX* yang bernilai 1001 sebagai jumlah maksimal dari banyaknya pekerjaan maupun pekerja.

```

1.  #include <cstdio>
2.  #include <vector>
3.  #include <queue>
4.  #include <algorithm>
5.  #include <cstring>
6.  using namespace std;
7.  #define MAX 1001
8.  #define ZERO 0
9.  #define INF (1<<28)

```

Kode Sumber 4.1. Potongan Kode penggunaan Library dan Konstanta The Dilemma of Idli

Konstanta ZERO memiliki nilai 0 sebagai *vertex dummy* yang berguna untuk melakukan proses *augmenting path*. Serta ditetapkan konstanta INF dengan besar 2^{28} sebagai representasi nilai *infinity*.

Pada tabel berikut dijelaskan mengenai variabel-variabel global yang akan digunakan dalam implementasi program.

Tabel 4.1. Tabel Daftar Variabel Global permasalahan The Dilemma of Idli Bagian 1

No	Nama Variabel	Tipe	Penjelasan
1	G	vector<int> []	Digunakan untuk menyimpan <i>bipartite graph</i> yang akan dibuat.
2	counterA	int	Digunakan untuk menghitung penduduk yang berada pada grup A
3	counterB	int	Digunakan untuk menghitung penduduk yang berada pada grup B

Tabel 4.2. Tabel Daftar Variabel Global permasalahan The Dilemma of Idli Bagian 2

No	Nama Variabel	Tipe	Penjelasan
4	match	int []	Digunakan untuk menyimpan <i>vertex</i> yang berpasangan dengan <i>indexnya</i> .
5	dist	int []	Digunakan untuk menyimpan jarak suatu node ke node lain pada saat proses pencarian <i>augmenting path</i> .
6	grupA	Vector <int>[][]	Digunakan untuk menyimpan data penduduk A beserta data anggota mana saja yang dia suka pada grup A dan anggota mana saja yang dia benci pada grup B
7	personBL	Vector <int>[]	Digunakan untuk menyimpan data semua orang pada grup B disertai dengan penduduk B mana saja yang menyukainya
8	PersonBH	Vector <int>[]	Digunakan untuk menyimpan data semua orang pada grup A disertai dengan penduduk B mana saja yang membencinya

Potongan kode yang dijelaskan oleh Tabel 4.1 dan Tabel 4.2 ditunjukkan oleh Kode Sumber 4.2.

```

1.   vector <int> G[2*MAX], grupA[MAX][2],
      personBL[1000005], personBH[1000005];
2.   int match[MAX], dist[MAX], counterA=0, counterB=0;

```

Kode Sumber 4.2. Potongan Kode Penggunaan Variabel Global permasalahan The Dilemma of Idli.

4.2.2 Implementasi Fungsi Main

Fungsi Main diimplementasikan sesuai pseudocode pada subbab 3.1.1.

```

1.   int main(){
2.       int n, n1, n2;
3.       scanf("%d %d",&n1, &n2);
4.       scanf("%d",&n);
5.
6.       for(int test=0; test < n; test++){
7.           char tmp;
8.           scanf(" %c",&tmp);
9.           if (tmp == 'A'){
10.              while(1){
11.                  int tmp2;
12.                  scanf("%d",&tmp2);
13.                  if(tmp2==-1)
14.                      break;
15.                  grupA[counterA][0].push_back(tmp2
16.              );
17.              }
18.              while(1){
19.                  int tmp2;
20.                  scanf("%d",&tmp2);
21.                  if(tmp2==-1)
22.                      break;
23.                  grupA[counterA][1].push_back(tmp2
24.              );
25.              }
26.          }
27.      }

```

Kode Sumber 4.3. Potongan Kode Implementasi Fungsi Main The Dilemma of Idli Bagian 1

```

24.         counterA++;
25.     }
26.     else{
27.         while(1){
28.             int tmp2;
29.             scanf("%d",&tmp2);
30.             if(tmp2== -1)
31.                 break;
32.             personBL[tmp2].push_back(counterB
33. );
34.         }
35.         while(1){
36.             int tmp2;
37.             scanf("%d",&tmp2);
38.             if(tmp2== -1)
39.                 break;
40.             personBH[tmp2].push_back(counterB
41. );
42.         }
43.     }
44.     InitGraph();
45.     printf("%d\n",n-HopcroftKarp());
46. }

```

Kode Sumber 4.4. Potongan Kode Implementasi Fungsi Main The Dilemma of Idli Bagian 2

Fungsi *main* yang diimplementasikan seperti pada Kode Sumber 4.3 digunakan untuk membaca masukan dari sistem (baris kode ke 6 hingga 43). Untuk penduduk grup A, setiap orang yang dia suka akan disimpan pada variabel `grupA[counterA][0]` dan untuk setiap orang yang dia benci akan disimpan pada variabel `grupA[counterA][1]`. Untuk penduduk grup B, pada variabel `personBL[i]`, akan disimpan penduduk grup B mana saja yang suka dengan orang ke $-i$ tersebut. Pada variabel `personBH[i]`, akan disimpan penduduk grup B mana saja yang benci dengan orang ke $-i$ tersebut.

4.2.3 Implementasi Fungsi *InitGraph*

Fungsi *InitGraph* diimplementasikan pada Kode Sumber 4.5 sesuai pseudocode pada subbab 3.1.2.1. Fungsi ini digunakan untuk membentuk sebuah *bipartite graph* dari deskripsi dan masukan permasalahan yang telah dijelaskan sebelumnya.

```

1.  void InitGraph(){
2.      for (int i=0; i<counterA; i++){
3.          for(int j=0; j<grupA[i][0].size(); j++){
4.              for(int k=0; k <
personBH[grupA[i][0][j]].size(); k++){
5.                  G[i+1].push_back(
personBH[grupA[i][0][j]][k] + 501);
6.
7.              }
8.          }
9.          for(int j=0; j<grupA[i][1].size(); j++){
10.             for(int k=0; k <
personBL[grupA[i][1][j]].size(); k++){
11.                 if(!(find(G[i+1].begin(),
G[i+1].end(), personBL[grupA[i][1][j]][k] + 501)
!= G[i+1].end()))
12.                     G[i+1].push_back(
personBL[grupA[i][1][j]][k] + 501);
13.             }
14.         }
15.     }
16. }

```

Kode Sumber 4.5. Potongan Kode Implementasi Fungsi *InitGraph*

4.2.4 Implementasi Fungsi *BFS*

Fungsi *BFS* diimplementasikan pada Kode Sumber 4.6 sesuai *pseudocode* pada subbab 3.1.2.2.1

```

1.  bool BFS() {
2.      int i, u, v, len;
3.      queue< int > Q;
4.      for(i=1; i<=MAX; i++) {
5.          if(match[i]==ZERO) {
6.              dist[i] = 0;
7.              Q.push(i);
8.          }
9.          else dist[i] = INF;
10.     }
11.     dist[ZERO] = INF;
12.     while(!Q.empty()) {
13.         u = Q.front(); Q.pop();
14.         if(u!=ZERO) {
15.             len = G[u].size();
16.             for(i=0; i<len; i++) {
17.                 v = G[u][i];
18.                 if(dist[match[v]]==INF) {
19.                     dist[match[v]] = dist[u]
20. + 1;
21.                     Q.push(match[v]);
22.                 }
23.             }
24.         }
25.     return (dist[ZERO]!=INF);
26. }

```

Kode Sumber 4.6. Potongan Kode Implementasi Fungsi *BFS*.

4.2.5 Implementasi Fungsi *DFS*

Fungsi DFS diimplementasikan pada Kode Sumber 4.7 sesuai *pseudocode* pada subbab 3.1.2.2.2

```

1.  bool DFS(int u) {
2.      int i, v, len;
3.      if(u!=ZERO) {
4.          len = G[u].size();
5.          for(i=0; i<len; i++) {
6.              v = G[u][i];
7.              if(dist[match[v]]==dist[u]+1) {
8.                  if(DFS(match[v])) {
9.                      match[v] = u;
10.                     match[u] = v;
11.                     return true;
12.                 }
13.             }
14.         }
15.         dist[u] = INF;
16.         return false;
17.     }
18.     return true;
19. }

```

Kode Sumber 4.7. Potongan Kode Implementasi Fungsi *DFS*.

4.2.6 Implementasi Fungsi *HopcroftKarp*

Fungsi Hopcroft-Karp diimplementasikan pada Kode Sumber 4.8 sesuai *pseudocode* pada subbab 3.1.2.2.3

```

1.  int hopcroft_karp() {
2.      int matching = 0;
3.      while(bfs())
4.          for(int i=1; i<=MAX; i++)
5.              if(match[i]==ZERO && dfs(i))
6.                  matching++;
7.      return matching;
8.  }

```

Kode Sumber 4.8. Potongan Kode Implementasi Fungsi *HopcroftKarp*.

4.3 Implementasi penyelesaian permasalahan *Yet Another Assignment Problem*

Pada subbab ini akan dijelaskan mengenai implementasi dari penyelesaian permasalahan *The Dilemma of Idli* sesuai dengan desain yang telah dibuat pada subbab 3.2.

4.3.1 Penggunaan *Library*, *Template*, *Konstanta* dan *Variabel Global*

Pada subbab ini akan dibahas penggunaan *library*, *template*, konstanta dan variabel global yang digunakan dalam sistem. Berikut adalah potongan kode yang menyatakan penggunaan *library* dan konstanta:

```

1.  #include <cstdio>
2.  #include <vector>
3.  #include <queue>
4.  #include <algorithm>
5.  #include <cstring>
6.  using namespace std;
7.  #define MAX 4001
8.  #define ZERO 0
9.  #define INF (1<<28)

```

Kode Sumber 4.9. Potongan Kode penggunaan Library dan Konstanta Yet Another Assignment Problem

Pada Kode sumber 4.9 terdapat lima *library* yang dipanggil antara lain: *cstdio*, *vector*, *queue*, *algorithm* serta *cstring*. Didefinisikan konstanta *MAX* yang bernilai 4001 sebagai jumlah maksimal dari banyaknya pekerjaan maupun pekerja. Konstanta *ZERO* memiliki nilai 0 sebagai *vertex dummy* yang berguna untuk melakukan proses *augmenting path*. Serta ditetapkan konstanta *INF* dengan besar 2^{28} sebagai representasi nilai *infinity*.

Pada tabel berikut dijelaskan mengenai variabel-variabel global yang akan digunakan dalam implementasi program.

Tabel 4.3. Tabel Daftar Variabel Global Permasalahan Yet Another Assignment Problem Bagian 1

No	Nama Variabel	Tipe	Penjelasan
1	G	vector<int> []	Digunakan untuk menyimpan <i>bipartite graph</i> yang akan dibuat.
2	M	int	Jumlah pekerjaan yang ada
3	N	int	Jumlah pekerja yang ada
4	match	int []	Digunakan untuk menyimpan <i>vertex</i> yang berpasangan dengan indexnya.
5	dist	int []	Digunakan untuk menyimpan jarak suatu node ke node lain pada saat proses pencarian <i>augmenting path</i> .
6	matrix	int [] []	Merupakan tempat menyimpan nilai $A_{i,j}$ sebagai waktu penyelesain pekerjaan i oleh pekerja j . Dan digunakan dalam proses <i>ExpandingMatrix</i>

Tabel 4.4. Tabel Daftar Variabel Global Permasalahan Yet Another Assignment Problem Bagian 2

No	Nama Variabel	Tipe	Penjelasan
7	sumRow	long long []	Digunakan untuk menyimpan jumlah setiap baris
8	sumCol	long long []	Digunakan untuk menyimpan jumlah setiap baris
9	maxSum	long long	Digunakan untuk menyimpan nilai tertinggi diantara sumRow dan sum Col

Potongan kode yang dijelaskan oleh Tabel 4.3 dan Tabel 4.4 ditunjukkan oleh Kode Sumber 4.10.

```

1.   vector< int > G[2*MAX];
2.   int M, N, match[2*MAX], dist[2*MAX],
     matrix[MAX][MAX];
3.   long long sumRow[2*MAX], sumCol[2*MAX], maxSum;

```

Kode Sumber 4.10. Potongan Kode Penggunaan Variabel Global Permasalahan Yet Another Assignment Problem.

Sistem ini menggunakan *template* yang ditunjukkan pada Kode Sumber 4.11. Kode Sumber 4.11 menampilkan *template* dari *getNum*. *Template* ini akan digunakan oleh program untuk mempercepat pembacaan masukan sesuai dengan yang dijelaskan pada deskripsi permasalahan.

```

1.     template <typename T>
2.     T getNum(){
3.         T res=0;
4.         char c;
5.         while (1){
6.             c=getchar_unlocked();
7.             if (c==' ' || c=='\n') continue;
8.             else break;
9.         }
10.        res=c-'0';
11.        while (1)    {
12.            c=getchar_unlocked();
13.            if (c>='0' && c<='9')
14.                res=10*res+c-'0';
15.            else break;
16.        }
17.        return res;
18.    }

```

Kode Sumber 4.11. Penggunaan Template getNum Pada Program Permasalahan Yet Another Assignment Problem.

4.3.2 Implementasi Fungsi Main

Fungsi Main diimplementasikan sesuai pseudocode pada subbab 3.2.1. Fungsi *main* yang diimplementasikan seperti pada Kode Sumber 4.12 digunakan untuk membaca masukan (baris kode ke-2, 3, dan 6), menghitung total nilai dari setiap baris dan kolom *matrix* yang menjadi inputan.

Setelah fungsi *ExpandingMatrix* dan *Solve* dijalankan (baris kode ke-11 dan 12), fungsi main akan mencetak nilai kembalian dari fungsi *Solve* (baris kode ke-12) sebagai keluaran waktu minimum yang diperlukan untuk menyelesaikan semua pekerjaan. Setelah itu, fungsi main akan mencetak susunan penugasan pada menit pertama menuju ke keluaran (baris kode ke-14 hingga 16).

```

1.     int main() {
2.         M=getNum<int>();
3.         N=getNum<int>();
4.         for(int i=0;i<M;i++){
5.             for(int j=0;j<N;j++){
6.                 matrix[i][j] =getNum<int>();
7.                 sumRow[i]+=matrix[i][j];
8.                 sumCol[j]+=matrix[i][j];
9.             }
10.        }
11.        ExpandingMatrix();
12.        printf("%lld\n",Solve());
13.        for(int i = 1;i<=N;i++){
14.            printf("%d ",match[i]-(M+N) <= M ? match[i]-
(M+N) : 0);
15.        }
16.        printf("\n");
17.        return 0;
18.    }

```

Kode Sumber 4.12. Potongan Kode Implementasi Fungsi Main Permasalahan Yet Another Assignment Problem

4.3.3 Implementasi Fungsi Solve

Fungsi *Solve* diimplementasikan pada Kode Sumber 4.13 sesuai *pseudocode* pada subbab 3.2.2.2.

```

1.     long long Solve(){
2.         long long step = maxSum;
3.         for(int i=0;i<M+N;i++){
4.             for(int j=0;j<M+N;j++){
5.                 if(matrix[i][j]>0){
6.                     G[j+1].push_back(i+1+M+N);
7.                 }
8.             }
9.         }
10.        HopcroftKarp();
11.        return step;
12.    }

```

Kode Sumber 4.13. Potongan Kode Implementasi Fungsi Solve Permasalahan Yet Another Assignment Problem.

4.3.4 Implementasi Fungsi *ExpandingMatrix*

Fungsi *ExpandingMatrix* diimplementasikan pada Kode Sumber 4.14 sesuai pseudocode pada subbab 3.2.2.1.

```

1.  void ExpandingMatrix(){
2.      maxSum = 0;
3.      for(int i=0;i<M;i++){
4.          maxSum = max(sumRow[i],maxSum);
5.      }
6.      for(int i=0;i<N;i++){
7.          maxSum = max(sumCol[i],maxSum);
8.      }
9.      for(int i=0,j=N;j<M+N;i++,j++){
10.         matrix[i][j] = maxSum - sumRow[i];
11.         sumRow[i]+= matrix[i][j];
12.         sumCol[j]+= matrix[i][j];
13.     }
14.     for(int i=M,j=0;i<M+N;i++,j++){
15.         matrix[i][j] = maxSum - sumCol[j];
16.         sumRow[i]+= matrix[i][j];
17.         sumCol[j]+= matrix[i][j];
18.     }
19.     for(int i=M;i<M+N;i++){
20.         for(int j=N; j<M+N; j++){
21.             matrix[i][j] = maxSum -
max(sumRow[i],sumCol[j]);
22.             sumRow[i]+= matrix[i][j];
23.             sumCol[j]+= matrix[i][j];
24.         }
25.     }
26. }

```

Kode Sumber 4.14. Potongan Kode Implementasi Fungsi *ExpandingMatrix*.

4.3.5 Implementasi Fungsi *BFS*

Fungsi *BFS* diimplementasikan pada Kode Sumber 4.15 sesuai *pseudocode* pada subbab 3.2.2.3.1

```

1.  bool BFS() {
2.      int i, u, v, len;
3.      queue< int > Q;
4.      for(i=1; i<=M+N; i++) {
5.          if(match[i]==ZERO) {
6.              dist[i] = 0;
7.              Q.push(i);
8.          }
9.          else dist[i] = INF;
10.     }
11.     dist[ZERO] = INF;
12.     while(!Q.empty()) {
13.         u = Q.front(); Q.pop();
14.         if(u!=ZERO) {
15.             len = G[u].size();
16.             for(i=0; i<len; i++) {
17.                 v = G[u][i];
18.                 if(dist[match[v]]==INF) {
19.                     dist[match[v]] = dist[u] +
20.                     1;
21.                     Q.push(match[v]);
22.                 }
23.             }
24.         }
25.     return (dist[ZERO]!=INF);
26. }

```

Kode Sumber 4.15. Potongan Kode Implementasi Fungsi *BFS*.

4.3.6 Implementasi Fungsi *DFS*

Fungsi *DFS* diimplementasikan pada Kode Sumber 4.16 sesuai *pseudocode* pada subbab 3.2.2.3.2

```

1.  bool DFS(int u) {
2.      int i, v, len;
3.      if(u!=ZERO) {
4.          len = G[u].size();
5.          for(i=0; i<len; i++) {
6.              v = G[u][i];
7.              if(dist[match[v]]==dist[u]+1) {
8.                  if(DFS(match[v])) {
9.                      match[v] = u;
10.                     match[u] = v;
11.                     return true;
12.                 }
13.             }
14.         }
15.         dist[u] = INF;
16.         return false;
17.     }
18.     return true;
19. }

```

Kode Sumber 4.16. Potongan Kode Implementasi Fungsi *DFS*.

4.3.7 Implementasi Fungsi *HopcroftKarp*

Fungsi *Hopcroft-Karp* diimplementasikan pada Kode Sumber 4.17 sesuai *pseudocode* pada subbab 3.2.2.3.3

```

1.  void HopcroftKarp() {
2.      while(BFS())
3.          for(int i=1; i<=M+N; i++)
4.              if(match[i]==ZERO)
5.                  DFS(i);
6.  }

```

Kode Sumber 4.17. Potongan Kode Implementasi Fungsi *HopcroftKarp*.

4.3.8 Implementasi Fungsi *Generate*

Fungsi *Generate* diimplementasikan pada Kode Sumber 4.18 sesuai pseudocode pada subbab 3.2.3.

```
1.  #include<cstdio>
2.  #include<stdlib>
3.  #include<algorithm>
4.  int main(){
5.      int m, n;
6.      scanf("%d %d", &m, &n);
7.      printf("%d %d\n",m,n);
8.      for(int i=0;i<m;i++){
9.          for(int j=0;j<n;j++){
10.             int Aij = rand() % 1000;
11.             printf("%d ",Aij);
12.          }
13.          printf("\n");
14.      }
15. }
```

Kode Sumber 4.18. Kode Implementasi Generator Data

BAB V

UJI COBA DAN ANALISIS

Pada bab ini dijelaskan tentang uji coba dan evaluasi dari hasil implementasi yang dilakukan pada tugas akhir ini.

5.1 Lingkungan Uji Coba

Bahasa yang digunakan pada uji coba ini adalah bahasa pemrograman C++ dengan bantuan *Integrated Development Environment* (IDE) Orwell Bloodshed Dev-C++ 5.9.2 serta Compiler g++ (TDM-GCC 4.8.1 32-bit). Sistem operasi yang digunakan adalah Windows 8 Professional 64 bit. Perangkat keras yang digunakan adalah prosesor Intel(R) Core(TM) i3-2130 @ 3.30 GHz dengan *Random Access Memory* (RAM) sebesar 6.00 GB.

5.2 Skenario Uji Coba

Pada subbab ini akan dijelaskan uji coba yang dilakukan.

5.2.1 Uji Coba Kebenaran

5.2.1.1 Uji Coba Kebenaran Penyelesaian Permasalahan *The Dilemma of Idli*

Uji coba kebenaran dilakukan dengan mengirimkan kode sumber program kedalam situs penilaian daring SPOJ Hasil uji coba dengan waktu terbaik pada situs penilaian daring SPOJ ditunjukkan pada gambar 5.1.

17068371	2016-06-09 08:36:42	The dilemma of Idli	accepted	0.31	26M	C++ 4.3.2
----------	---------------------	---------------------	----------	------	-----	--------------

Gambar 5.1. Hasil Uji Coba pada Situs Penilaian Daring SPOJ permasalahan *The Dilemma of Idli*

Dari hasil uji coba kebenaran yang dilakukan pada situs penilaian daring SPOJ seperti pada Gambar 5.1, dapat dilihat bahwa kode sumber mendapat keluaran *Accepted*. Waktu yang tercepat yang dibutuhkan program adalah 0.31 detik dan memori yang dibutuhkan adalah 26 MB.

Selanjutnya akan dibandingkan hasil uji coba kasus sederhana dengan keluaran sistem. Uji coba kasus sederhana akan diselesaikan sesuai dengan langkah penyelesaian pada subbab 2.7.

Kasus sederhana yang digunakan adalah sebuah kasus dimana terdapat masing-masing 10 orang dari *Committee of the Galactical Wars* maupun *Inter-Galactic Parliament* serta terdapat 8 orang penduduk yang terbagi menjadi dan 4 orang kelompok A, dan 4 orang kelompok B. Format masukan dari masalah yang akan diselesaikan adalah pada Gambar 5.3.

```

1. 10 10
2. 8
3. A 1 5 -1 2 3 -1
4. B 1 7 8 -1 5 7 -1
5. A 2 3 7 -1 1 3 -1
6. A 3 4 10 -1 4 6 8 -1
7. B 2 7 -1 1 -1
8. B 5 6 8 -1 4 10 -1
9. A 6 8 9 -1 3 4 9 -1
10. B 1 3 5 -1 2 6 8 -1

```

Gambar 5.2 Format masukan uji kebenaran *The Dilemma of Idli*

Dimisalkan memilih memuaskan seorang penduduk A, sebagai konsekuensinya, semua penduduk B yang menyukai orang yang dibenci oleh A merasa tidak puas, begitu juga sebaliknya. Hal tersebut menunjukkan bahwa himpunan orang yang disukai pada sebuah kelompok berlawanan dengan himpunan orang yang dibenci pada kelompok yang lain. Berdasarkan hal ini, data masukan dapat direpresentasikan menjadi beberapa. Hasil representasi data masukan ditunjukkan oleh tabel 5.1 dan tabel 5.2.

Tabel 5.1 Tabel deskripsi penduduk grup A

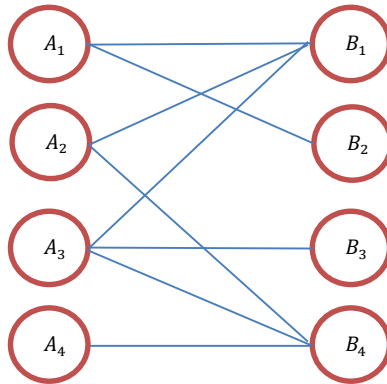
Penduduk Grup A	Orang yang disuka	Orang yang dibenci
1	1, 5	2, 3
2	2, 3, 7	1, 3
3	3, 4, 10	4, 6, 8
4	6, 8, 9	3, 4, 9

Tabel 5.2 Tabel deskripsi penduduk grup B

Penduduk Grup B	Orang yang disuka	Orang yang dibenci
1	1, 7, 8	5, 7
2	2, 7	1
3	5, 6, 8	4, 10
4	1, 3, 5	2, 6, 8, 10

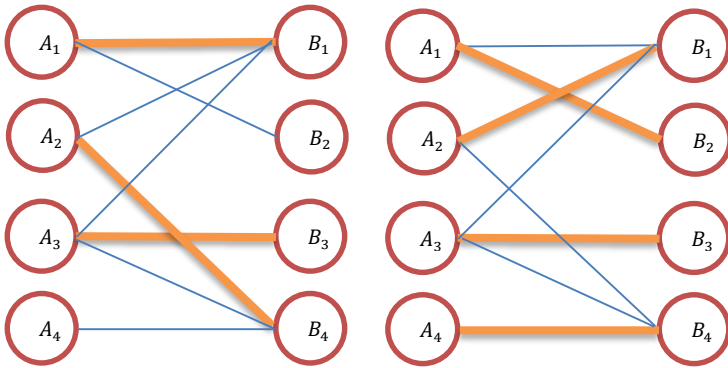
Berdasarkan tabel 5.1 dan tabel 5.2, Sebuah *bipartite graph* $G = (A \cup B, E)$ dengan A merupakan penduduk pada grup A dan B adalah penduduk pada grup B, serta E merupakan representasi dari hubungan setiap orang yang dibenci penduduk A_i tetapi disukai penduduk B_i ataupun sebaliknya. *Bipartite graph* yang dihasilkan ditunjukkan oleh gambar 5.2.

Selanjutnya akan dicari *Maximum Bipartite Matching* dengan menggunakan algoritma Hopcroft-Karp pada *bipartite graph* yang telah terbentuk. *Maximum Bipartite Matching* pada graf tersebut akan mengembalikan nilai t yaitu total *vertex* minimum pada A dan B yang dapat diambil dari hubungan *edge* yang ada atau dengan kata lain banyaknya orang yang tidak dapat dipuaskan. Langkah-langkah pencarian *Maximum Bipartite Matching* pada graf ditunjukkan oleh gambar 5.3.

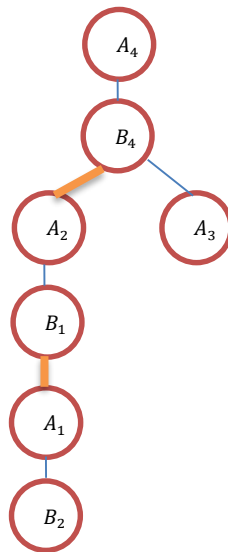


Gambar 5.3. Bipartite graph hasil representasi permasalahan.

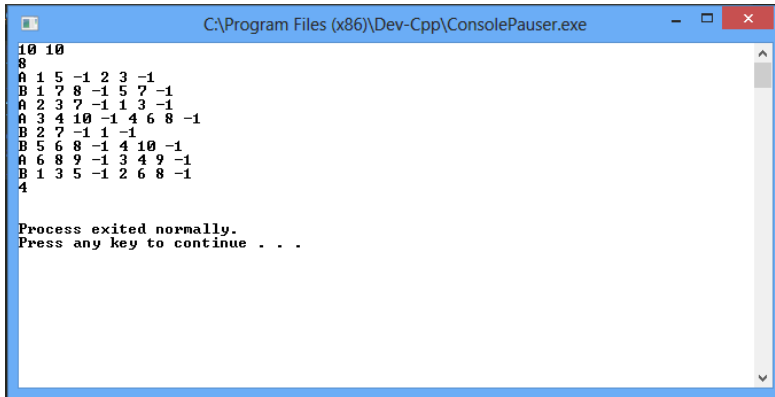
Gambar 5.4 (a) menunjukkan proses iterasi pertama dari pencarian *maximum bipartite matching* pada graf, dikarenakan pada awal iterasi hanya terdapat *unmatched edge*, maka *matching* yang dihasilkan merupakan *edge* dari hubungan setiap *unmatched node A* dan *unmatched node B* pertama yang dapat dijangkau dari proses *depth-first search* yang dilakukan. Pada iterasi kedua, proses pencarian *matching* dilakukan dengan node A_4 sebagai awal pencarian. Melalui proses *breadth-first search* yang dilakukan, didapat sebuah *tree* yang ditunjukkan oleh gambar 5.5. Selanjutnya akan dilakukan *depth-first search* pada *tree* tersebut hingga menemukan *unmatched node* pada B . Proses ini akan menghasilkan sebuah *augmenting path* $\{(A_4, B_4), (B_4, A_2), (A_2, B_1), (B_1, A_1), (A_1, B_2)\}$. Berdasarkan proses *augmenting path* tersebut, didapat *matching* baru $m = \{(A_4, B_4), (A_2, B_1), (A_1, B_2)\}$ yang nantinya akan digabungkan dengan *matching* sebelumnya sehingga didapatkan *matching* seperti yang ditunjukkan oleh gambar 5.4 (b). Dikarenakan sudah tidak *augmenting path* lagi yang dapat ditemukan, maka proses pencarian *maximum bipartite matching* diakhiri dengan nilai kembalian 4 sebagai jumlah *matching* yang ditemukan



Gambar 5.4 Proses pencarian maximum bipartite matching (a) proses pencarian pada iterasi pertama (b) proses pencarian pada iterasi kedua



Gambar 5.5. Tree yang dihasilkan dari proses pencarian iterasi kedua



```

10 10
8
A 1 5 -1 2 3 -1
B 1 7 8 -1 5 7 -1
A 2 3 7 -1 1 3 -1
A 3 4 10 -1 4 6 8 -1
B 2 7 -1 1 -1
B 5 6 8 -1 4 10 -1
A 6 8 9 -1 3 4 9 -1
B 1 3 5 -1 2 6 8 -1
4

Process exited normally.
Press any key to continue . . .

```

Gambar 5.6. Keluaran sistem penyelesaian permasalahan *The Dilemma of Idli*

Jumlah total orang yang puas adalah total dari penduduk yang ada dikurangi oleh *Maximum Bipartite Matching* yang ditemukan. Dengan kata lain jumlah penduduk yang puas adalah $8 - 4 = 4$.

Kemudian, sistem penyelesaian dijalankan dan diberi masukan pada Gambar 5.2. Keluaran sistem telah sesuai dengan permasalahan, tercetak 4 sebagai jumlah maksimal orang yang dapat dipusakan oleh Idli. Keluaran sitem ditunjukkan oleh gambar 5.6.

Berdasarkan hasil uji coba tersebut, Algoritma Hopcroft-Karp yang didesain dan dijalankan pada penyelesaian permasalahan tersebut, dapat menyelesaikan permasalahan pencarian *Maximum Matching* pada sebuah *bipartite graph*. Selanjutnya, desain algoritma Hopcroft-Karp yang telah dibuat ini akan digunakan untuk menyelesaikan permasalahan *Yet Another Assignment Problem*.

5.2.1.2 Uji Coba Kebenaran Penyelesaian Permasalahan *Yet Another Assignment Problem*

Uji coba kebenaran dilakukan dengan mengirimkan kode sumber program kedalam situs penilaian daring SPOJ dan melakukan perbandingan hasil uji coba kasus sederhana dengan langkah-langkah yang ada pada Subbab 2.9 dengan keluaran sistem. Permasalahan yang diselesaikan adalah *Yet Another Assignment Problem* dengan kode soal ASSIGN5 seperti yang dijelaskan pada Subbab 2.9. Hasil uji coba dengan waktu terbaik pada situs penilaian daring SPOJ ditunjukkan pada gambar 5.7.

17014913	2016-05-30 09:29:40	Yet Another Assignment Problem	accepted	0.08	64M	C++ 4.3.2
----------	------------------------	-----------------------------------	----------	------	-----	--------------

Gambar 5.7. Hasil Uji Coba pada Situs Penilaian Daring SPOJ permasalahan *Yet Another Assignment Problem*

Dari hasil uji coba kebenaran yang dilakukan pada situs penilaian daring SPOJ seperti pada Gambar 5.7, dapat dilihat bahwa kode sumber mendapat keluaran *Accepted*. Waktu yang tercepat yang dibutuhkan program adalah 0.08 detik dan memori yang dibutuhkan adalah 64 MB.

Selain itu, dilakukan pengujian sebanyak 20 kali pada situs penilaian daring SPOJ untuk melihat variasi waktu dan memori yang dibutuhkan program. Hasil uji coba sebanyak 20 kali dapat dilihat pada Tabel 5.3.

Dari hasil uji coba sebanyak 20 kali, seluruh kode sumber program mendapat keluaran *Accepted* dengan waktu minimum sebesar 0.08 detik, waktu maksimum sebesar 0.12 detik dan waktu rata-rata sebesar 0.103 detik. Memori yang dibutuhkan program konstan sebesar 64 MB.

Tabel 5.3. Tabel Hasil Uji Coba pada Situs SPOJ Sebanyak 20 Kali

ID	RESULT	TIME	MEM
17014913	accepted	0.08	64M
17014912	accepted	0.10	64M
17014910	accepted	0.10	64M
17014909	accepted	0.09	64M
17014906	accepted	0.11	64M
17014903	accepted	0.11	64M
17014901	accepted	0.09	64M
17014899	accepted	0.10	64M
17014898	accepted	0.11	64M
17014896	accepted	0.12	64M
17014895	accepted	0.11	64M
17014893	accepted	0.09	64M
17014891	accepted	0.11	64M
17014890	accepted	0.10	64M
17014889	accepted	0.11	64M
17014886	accepted	0.12	64M
17014883	accepted	0.11	64M
17014882	accepted	0.10	64M
17014879	accepted	0.10	64M
17011446	accepted	0.10	64M

Selanjutnya akan dibandingkan hasil uji coba kasus sederhana dengan keluaran sistem. Uji coba kasus sederhana akan diselesaikan sesuai dengan langkah penyelesaian pada subbab 2.9. Dalam hal ini akan diselesaikan masalah penugasan bersyarat dengan skala problem yang kecil. Kasus sederhana yang digunakan adalah sebuah kasus dimana terdapat 3 orang pekerja dan 2 buah pekerjaan. Format masukan dari masalah yang akan diselesaikan adalah pada Gambar 5.8.

11. 2 3
12. 2 2 4
13. 1 2 0

Gambar 5.8. Format masukan uji coba kebenaran.

Dari data masukan tersebut diketahui bahwa nilai $m = 2$ dan nilai $n = 3$, pertama-tama akan dicari nilai α yang merupakan waktu minimum yang diperlukan untuk melakukan semua pekerjaan yang ada. Nilai α dapat dihitung dari jumlah waktu terbanyak yang dilakukan seseorang untuk menyelesaikan semua pekerjaannya atau waktu terbanyak yang dibutuhkan oleh sebuah pekerjaan agar diselesaikan oleh semua orang.

Tabel 5.4. Tabel pencarian nilai α .

	Pekerja 1	Pekerja 2	Pekerja 3	$\sum_{i=0, j=0}^{n-1, m-1} A_{j,i}$
Pekerjaan 1	2	2	4	8
Pekerjaan 2	1	2	0	3
$\sum_{i=0, j=0}^{m-1, n-1} A_{i,j}$	3	4	4	

2	2	4
1	2	0

(a)

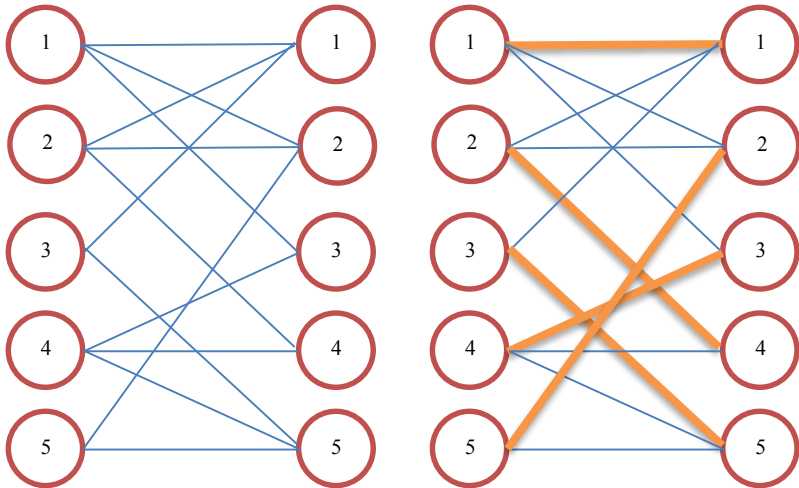
2	2	4	0	0
1	2	0	0	5
5	0	0	3	0
0	4	0	4	0
0	0	4	1	3

(b)

Gambar 5.9. Proses *expanding-matrix* (a) *Matrix awal* (b) *Matrix setelah mengalami proses expanding-matrix*

Berdasarkan Tabel 5.4, untuk kasus yang diujikan, didapatkan nilai $\alpha = 8$. Setelah didapatkan nilai α , dilakukan proses *expanding matrix* dengan terlebih dahulu membuat sebuah *matrix* yang berukuran $(m + n) * (n + m)$ dan mengisi *matrix* tersebut dengan aturan yang dijelaskan pada Subbab 2.9. Proses *expanding matrix* ditunjukkan oleh Gambar 5.9.

Sesuai dengan Gambar 5.9(b), akan dibuat sebuah *bipartite graph* $G = (X \cup Y, E)$ dimana dengan X merupakan representasi dari kolom (pekerja) dan Y merupakan representasi dari baris (pekerjaan). Sebuah *edge* pada E menghubungkan *vertex* pada X dan Y apabila nilai $A_{i,j}$ pada *matrix* tersebut tidak 0. Setelah graf terbentuk, akan dicari *Maximum Bipartite Matching* dengan menggunakan algoritma Hopcroft-Karp. Pembentukan graf dan langkah hasil algoritma Hopcroft-Karp pada graf tersebut ditunjukkan oleh Gambar 5.10. Berdasarkan gambar tersebut, didapatkan *matching* $m = \{(1,1), (2,4), (3,5), (4,3), (5,2)\}$.



Gambar 5.10. Pembentukan graf berdasarkan matrix awal (a) Bipartite graf iterasi ke-1 (b) Maximum matching pada graf iterasi ke-1

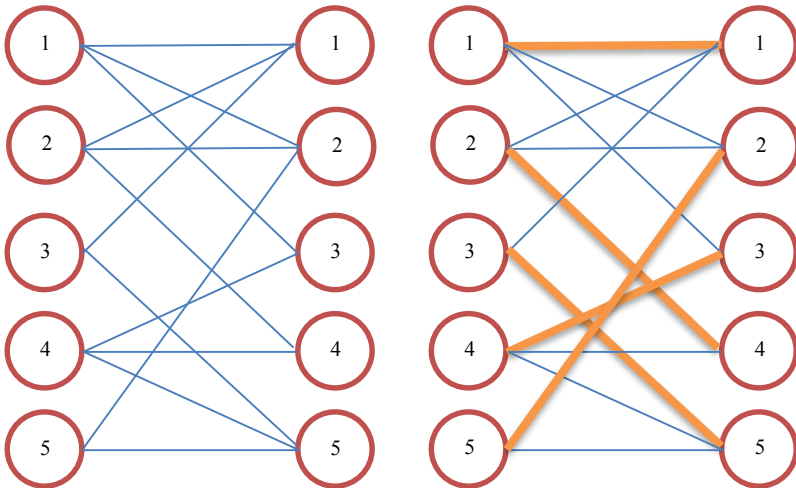
Dari *matching* tersebut diketahui bahwa pekerja 1 mengerjakan pekerjaan 1. Sedangkan pekerja 2 dipasangkan dengan pekerjaan 4 dan pekerja 3 dipasangkan dengan pekerjaan 5, hal ini berarti kedua pekerja tersebut tidak mengerjakan pekerjaan apapun pada satuan waktu tersebut. Hal tersebut dikarenakan nilai baris $j > m$. Sesuai dengan deskripsi keluaran program, hasil tersebut juga dapat direpresentasikan sebagai “1 0 0”.

Selanjutnya dilakukan pengurangan nilai sebanyak 1 pada nilai $A_{i,j}$ matrix yang ditunjukkan oleh Gambar 5.9(b) untuk setiap baris j dan kolom i yang berpasangan pada *matching* m . Selanjutnya, proses pengurangan terhadap nilai matrix ini akan disebut sebagai reduksi *matrix*. Hasil *matrix* setelah dilakukan proses tersebut ditunjukkan oleh Gambar 5.11

1	2	4	0	0
1	2	0	0	4
5	0	0	2	0
0	3	0	4	0
0	0	3	1	3

Gambar 5.11. Hasil matrix setelah proses reduksi nilai pada iterasi ke-1

Bagian yang berwarna kuning pada matrix yang baru adalah posisi nilai *matrix* yang berubah sesuai dengan hasil *matching* yang sebelumnya. Dengan *matrix* yang baru tersebut, langkah-langkah membuat *bipartite graph* yang baru sampai dengan mencari *Maximum bipartite matching* pada graf tersebut akan diulangi hingga *matrix* yang dihasilkan berupa *matrix* kosong. Proses lengkap dari bagian ini ditunjukkan oleh Gambar 5.12 hingga Gambar 5.24.

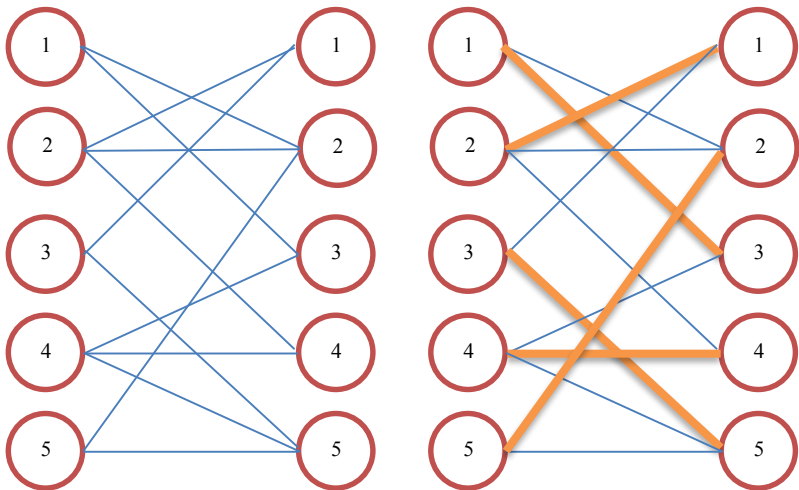


Gambar 5.12. Pembentukan graf berdasarkan matrix iterasi ke-1 (a) Bipartite graf iterasi ke-2 (b) Maximum matching pada graf iterasi ke-2

0	2	4	0	0
1	2	0	0	3
5	0	0	1	0
0	2	0	4	0
0	0	2	1	3

Gambar 5.13. Hasil matrix setelah proses reduksi nilai pada iterasi ke-2

Berdasarkan Gambar 5.12, pada proses iterasi ke-2, *matching* yang didapatkan yaitu $m = \{(1,1), (2,4), (3,5), (4,3), (5,2)\}$. Melalui hasil tersebut diketahui bahwa pekerja 1 mengerjakan pekerjaan 1, sedangkan pekerja 2 dan 3 tidak melakukan pekerjaan apapun pada satuan waktu tersebut. Hasil *matrix* yang baru setelah proses iterasi 2 ditunjukkan oleh Gambar 5.13.

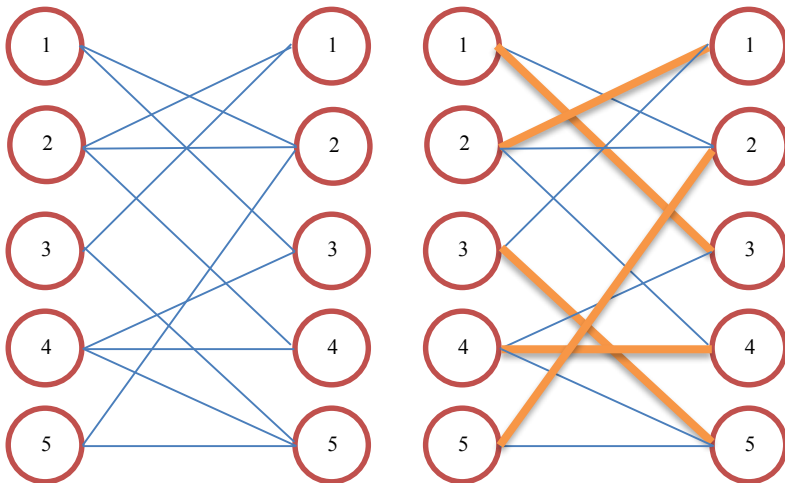


Gambar 5.14. Pembentukan graf berdasarkan matrix iterasi ke-2 (a) Bipartite graf iterasi ke-3 (b) Maximum matching pada graf iterasi ke-3

0	1	4	0	0
1	2	0	0	2
4	0	0	1	0
0	2	0	3	0
0	0	1	1	3

Gambar 5.15. Hasil matrix setelah proses reduksi nilai pada iterasi ke-3

Berdasarkan Gambar 5.14, pada proses iterasi ke-3, *matching* yang didapatkan yaitu $m = \{(1,3), (2,1), (3,5), (4,4), (5,2)\}$. Hasil *matching* tersebut menunjukkan bahwa pekerja 2 mengerjakan pekerjaan 1 sedangkan pekerja 1 dan 3 tidak melakukan pekerjaan apapun pada satuan waktu tersebut. Hasil *matrix* yang baru setelah proses iterasi 3 ditunjukkan oleh Gambar 5.15.

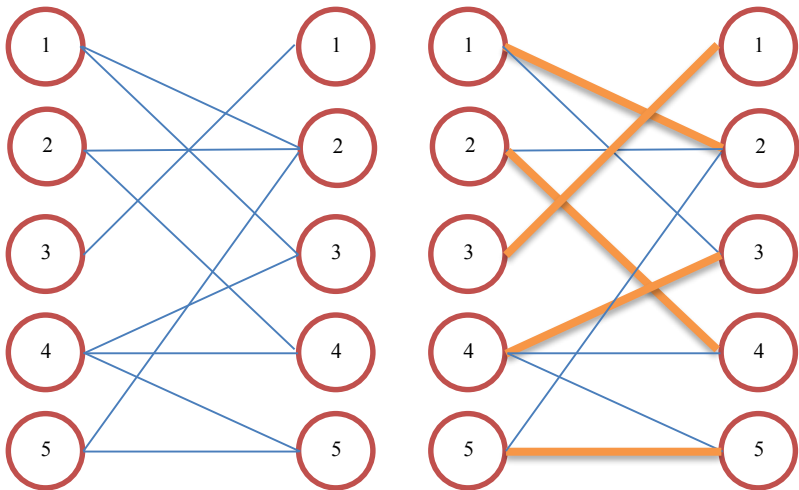


Gambar 5.16. Pembentukan graf berdasarkan matrix iterasi ke-3 (a) Bipartite graf iterasi ke-4 (b) Maximum matching pada graf iterasi ke-4

0	0	4	0	0
1	2	0	0	1
3	0	0	1	0
0	2	0	2	0
0	0	0	1	3

Gambar 5.17. Hasil matrix setelah proses reduksi nilai pada iterasi ke-4

Berdasarkan Gambar 5.16, pada proses iterasi ke-4, *matching* yang didapatkan yaitu $m = \{(1,3), (2,1), (3,5), (4,4), (5,2)\}$. Melalui hasil tersebut diketahui bahwa pekerja 2 mengerjakan pekerjaan 1, sedangkan pekerja 1 dan 3 tidak melakukan pekerjaan apapun pada satuan waktu tersebut. Hasil *matrix* yang baru setelah proses iterasi 4 ditunjukkan oleh Gambar 5.17.

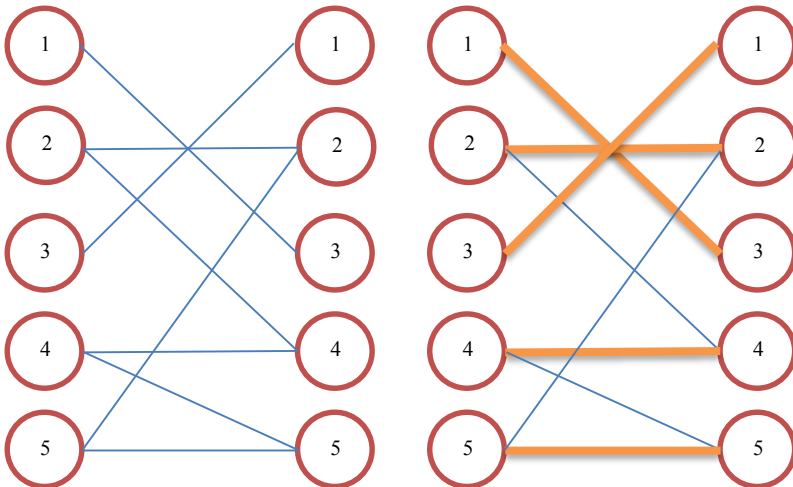


Gambar 5.18. Pembentukan graf berdasarkan matrix iterasi ke-4 (a) Bipartite graf iterasi ke-5 (b) Maximum matching pada graf iterasi ke-5

0	0	3	0	0
0	2	0	0	1
3	0	0	0	0
0	1	0	2	0
0	0	0	1	2

Gambar 5.19. Hasil matrix setelah proses reduksi nilai pada iterasi ke-5

Berdasarkan Gambar 5.18, pada proses iterasi ke-5, *matching* yang didapatkan yaitu $m = \{(1,2), (2,4), (3,1), (4,3), (5,5)\}$. Hasil *matching* tersebut menunjukkan bahwa pekerja 1 mengerjakan pekerjaan 2 sedangkan pekerja 3 mengerjakan pekerjaan 1. Pekerja 3 tidak melakukan pekerjaan apapun pada satuan waktu tersebut. Hasil *matrix* yang baru setelah proses iterasi 5 ditunjukkan oleh Gambar 5.19.

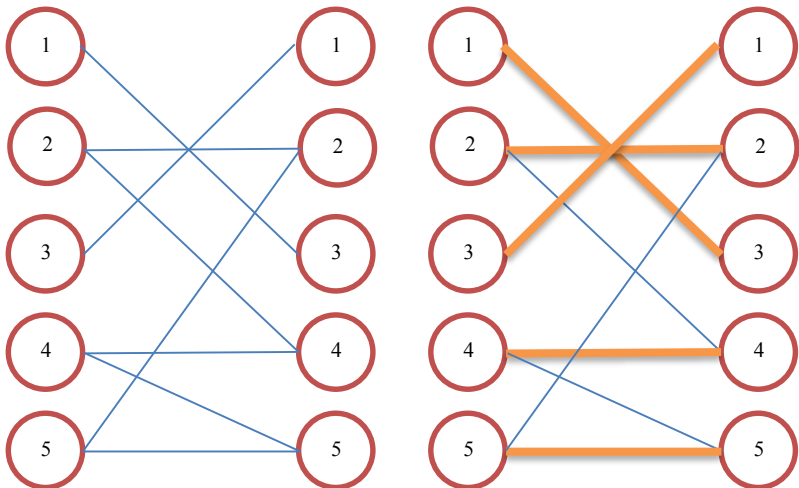


Gambar 5.20. Pembentukan graf berdasarkan matrix iterasi ke-5 (a) Bipartite graf iterasi ke-6 (b) Maximum matching pada graf iterasi ke-6

0	0	2	0	0
0	1	0	0	1
2	0	0	0	0
0	1	0	1	0
0	0	0	1	1

Gambar 5.21. Hasil matrix setelah proses reduksi nilai pada iterasi ke-6

Berdasarkan Gambar 5.20, pada proses iterasi ke-6, *matching* yang didapatkan yaitu $m = \{(1,3), (2,2), (3,1), (4,4), (5,5)\}$. Hasil *matching* tersebut menunjukkan bahwa pekerja 2 mengerjakan pekerjaan 2 sedangkan pekerja 3 melakukan pekerjaan 1 dan pekerja 3 tidak melakukan pekerjaan apapun pada satuan waktu tersebut. Hasil *matrix* yang baru setelah proses iterasi 6 ditunjukkan oleh Gambar 5.21.

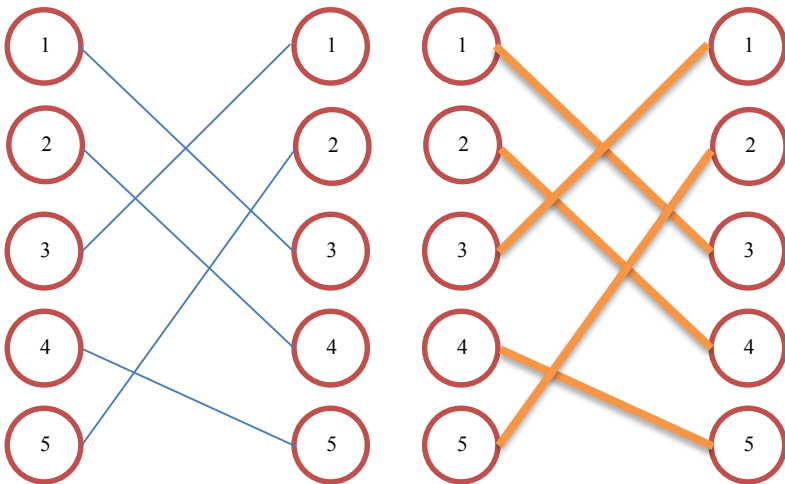


Gambar 5.22. Pembentukan graf berdasarkan matrix iterasi ke-6 (a) Bipartite graf iterasi ke-7 (b) Maximum matching pada graf iterasi ke-7

0	0	1	0	0
0	0	0	0	1
1	0	0	0	0
0	1	0	0	0
0	0	0	1	0

Gambar 5.23. Hasil matrix setelah proses reduksi nilai pada iterasi ke-7

Berdasarkan Gambar 5.22, pada proses iterasi ke-7, *matching* yang didapatkan yaitu $m = \{(1,3), (2,2), (3,1), (4,4), (5,5)\}$. *Matching* tersebut menunjukkan hasil yang sama dengan iterasi ke-6, sehingga dapat diketahui bahwa pekerja 2 mengerjakan pekerjaan 2 sedangkan pekerja 3 melakukan pekerjaan 1 dan pekerja 3 tidak melakukan pekerjaan apapun pada satuan waktu tersebut. Hasil *matrix* yang baru setelah proses iterasi 7 ditunjukkan oleh Gambar 5.23.



Gambar 5.24. Pembentukan graf berdasarkan matrix iterasi ke-7 (a) Bipartite graf iterasi ke-8 (b) Maximum matching pada graf iterasi ke-8

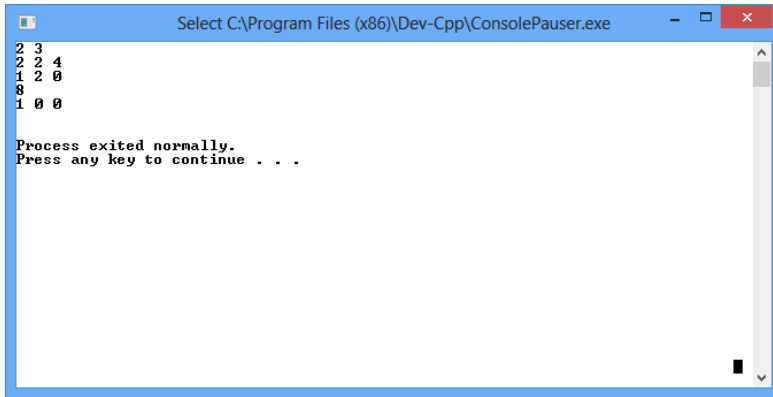
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

Gambar 5.25. Hasil matrix setelah proses reduksi nilai pada iterasi ke-8

Berdasarkan Gambar 5.24, pada proses iterasi ke-8, *matching* yang didapatkan yaitu $m = \{(1,3), (2,4), (3,1), (4,5), (5,2)\}$. Hasil *matching* tersebut menunjukkan bahwa pekerja 2 mengerjakan pekerjaan 2 sedangkan pekerja 3 melakukan pekerjaan 1 dan pekerja 3 tidak melakukan pekerjaan apapun pada satuan waktu tersebut. Hasil *matrix* yang baru setelah proses iterasi 8 ditunjukkan oleh Gambar 5.25. *Matrix* yang dihasilkan setelah iterasi ini adalah *matrix* kosong, sehingga kasus uji coba tersebut selesai pada iterasi ke 8. Susunan lengkap dari penugasan tiap satuan waktunya berdasarkan proses diatas ditunjukkan oleh Tabel 5.5. Berdasarkan tabel tersebut, waktu yang dibutuhkan untuk menyelesaikan semua pekerjaan tersebut adalah 8 menit. Hal ini membuktikan bahwa nilai $\alpha = 8$ adalah waktu minimal yang diperlukan agar semua pekerjaan tersebut dapat diselesaikan.

Tabel 5.5. Susunan penugasan tiap satuan waktu untuk kasus uji

Menit	Pekerja 1	Pekerja 2	Pekerja 3
1	1	0	0
2	1	0	0
3	0	1	0
4	0	1	0
5	2	0	1
6	0	2	1
7	0	2	1
8	0	0	1



```

Select C:\Program Files (x86)\Dev-Cpp\ConsolePauser.exe
2 3
2 2 4
1 2 0
8
1 0 0

Process exited normally.
Press any key to continue . . .

```

Gambar 5.26. Masukan dan Keluaran pada Program.

Didapatkan hasil penugasan pada menit pertama yang dilakukan secara manual adalah 1, 0, 0. Hal ini berarti pekerja 1 mengerjakan pekerjaan 1 dan pekerja lainnya tidak mengerjakan pekerjaan apapun.

Kemudian, sistem penyelesaian dijalankan dan diberi masukan pada Gambar 5.3. Keluaran sistem telah sesuai dengan permasalahan, tercetak 8 untuk waktu minimum yang diperlukan menyelesaikan semua pekerjaan dan “1 0 0” sebagai susunan penugasan pada menit pertama, seperti terlihat pada Gambar 5.26. Hal ini menunjukkan bahwa sistem penyelesaian yang telah dibuat, tepat mengeluarkan solusi dari permasalahan *Yet Another Assignment Problem*”.

5.2.2 Uji Coba Kinerja Penyelesaian Permasalahan *Yet Another Assignment Problem*

Kompleksitas waktu pencarian *perfect matching* dengan menggunakan algoritma Hopcroft-Karp adalah $O(|V|\sqrt{|E|})$. Dengan representasi graf diatas, banyaknya *edge* $|E|$ yang mungkin terbentuk adalah maksimal $(m + n)^2$ dengan banyaknya *vertex* $|V|$ pada $X \cup Y$ adalah $2 * (m + n)$, sehingga kompleksitas

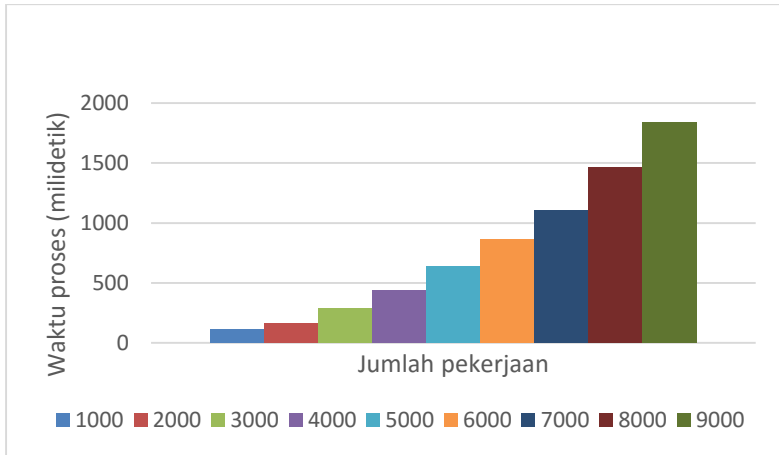
total dengan representasi graf tersebut adalah $O((m + n)^2)$. Dalam bab ini akan dilakukan dua macam uji coba. Pertama, banyaknya pekerja dibuat tetap namun banyaknya pekerjaan dibuat bervariasi. Kedua, banyaknya pekerja dibuat bervariasi namun banyaknya pekerjaan dibuat tetap. Hal tersebut akan menunjukkan bagaimana pengaruh dari data yang dibuat bervariasi terhadap waktu eksekusi program.

5.2.2.1 Pengaruh Banyaknya Pekerjaan Terhadap Waktu

Pada uji coba ini banyaknya pekerjaan dibuat bervariasi antara 1000 hingga 9000. Jumlah pekerja ditetapkan sebanyak 1000. Dicatat waktu eksekusi program dimana tiap data dalam satuan milidetik agar pengaruh banyaknya pekerjaan terhadap waktu eksekusi dapat diamati. Hasil uji coba dari percobaan tersaji pada Tabel 5.6 dan digambarkan dalam grafik seperti yang terlihat pada Gambar 5.21.

Tabel 5.6. Tabel pengaruh jumlah pekerja terhadap waktu proses program

Percobaan	Jumlah pekerjaan	Waktu (milidetik)
1	1000	115
2	2000	163
3	3000	291
4	4000	441
5	5000	635
6	6000	859
7	7000	1102
8	8000	1463
9	9000	1843



Gambar 5.27. Grafik pengaruh jumlah pekerja terhadap waktu proses program

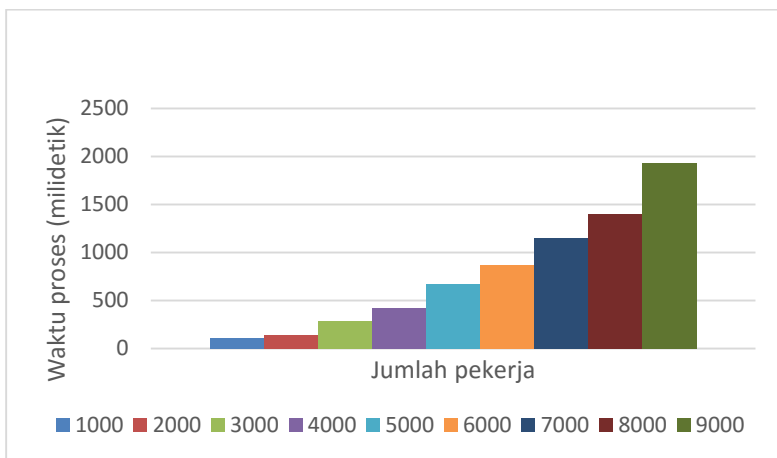
Seperti yang terlihat pada Gambar 5.27, grafik cenderung mendekati kurva kuadratik. Hal ini sesuai dengan kompleksitas algoritma penyelesaian yang dipengaruhi secara kuadratik oleh jumlah pekerja.

5.2.2.2 Pengaruh Banyaknya Pekerja Terhadap Waktu

Pada uji coba ini banyaknya pekerja dibuat bervariasi antara 1000 hingga 9000. Jumlah pekerja ditetapkan sebanyak 1000. Dicatat waktu eksekusi program dimana tiap data dalam satuan milidetik agar pengaruh banyaknya pekerja terhadap waktu eksekusi dapat diamati. Hasil uji coba dari percobaan tersaji pada Tabel 5.7 dan digambarkan dalam grafik seperti yang terlihat pada Gambar 5.28.

Tabel 5.7. Tabel pengaruh jumlah pekerja terhadap waktu proses program

Percobaan	Jumlah pekerja	Waktu (milidetik)
1	1000	105
2	2000	142
3	3000	281
4	4000	420
5	5000	668
6	6000	870
7	7000	1150
8	8000	1402
9	9000	1932



Gambar 5.28. Grafik pengaruh jumlah pekerja terhadap waktu proses program

Seperti yang terlihat pada Gambar 5.28, grafik cenderung mendekati kurva kuadratik pula. Hal ini sesuai dengan kompleksitas algoritma penyelesaian yang dipengaruhi secara kuadratik oleh jumlah pekerja yang ada.

LAMPIRAN A

17014895	2018-05-30 09:28:54	Yet Another Assignment Problem	accepted	0.21	64M	C++ 4.3.2
17014893	2018-05-30 09:26:37	Yet Another Assignment Problem	accepted	0.09	64M	C++ 4.3.2
17014891	2018-05-30 09:26:13	Yet Another Assignment Problem	accepted	0.11	64M	C++ 4.3.2
17014890	2018-05-30 09:26:10	Yet Another Assignment Problem	accepted	0.00	64M	C++ 4.3.2
17014889	2018-05-30 09:24:00	Yet Another Assignment Problem	accepted	0.21	64M	C++ 4.3.2
17014886	2018-05-30 09:23:41	Yet Another Assignment Problem	accepted	0.12	64M	C++ 4.3.2
17014883	2018-05-30 09:22:21	Yet Another Assignment Problem	accepted	0.11	64M	C++ 4.3.2
17014882	2018-05-30 09:22:11	Yet Another Assignment Problem	accepted	0.00	64M	C++ 4.3.2
17014879	2018-05-30 09:21:02	Yet Another Assignment Problem	accepted	0.20	64M	C++ 4.3.2
17011446	2019-05-23 15:01:35	Yet Another Assignment Problem	accepted	0.10	64M	C++ 4.3.2
17014913	2018-05-30 09:20:00	Yet Another Assignment Problem	accepted	0.08	64M	C++ 4.3.2
17014912	2018-05-30 09:20:33	Yet Another Assignment Problem	accepted	0.10	64M	C++ 4.3.2
17014910	2018-05-30 09:20:22	Yet Another Assignment Problem	accepted	0.10	64M	C++ 4.3.2
17014909	2018-05-30 09:20:09	Yet Another Assignment Problem	accepted	0.09	64M	C++ 4.3.2
17014906	2018-05-30 09:20:19	Yet Another Assignment Problem	accepted	0.21	64M	C++ 4.3.2
17014903	2018-05-30 09:21:48	Yet Another Assignment Problem	accepted	0.11	64M	C++ 4.3.2
17014901	2018-05-30 09:21:30	Yet Another Assignment Problem	accepted	0.09	64M	C++ 4.3.2
17014899	2018-05-30 09:21:24	Yet Another Assignment Problem	accepted	0.10	64M	C++ 4.3.2
17014898	2018-05-30 09:21:14	Yet Another Assignment Problem	accepted	0.21	64M	C++ 4.3.2
17014896	2018-05-30 09:21:03	Yet Another Assignment Problem	accepted	0.12	64M	C++ 4.3.2

Gambar A.1. Hasil Uji Coba pada Situs SPOJ Sebanyak 20 Kali.

BAB VI

KESIMPULAN DAN SARAN

Pada bab ini dijelaskan mengenai kesimpulan dari hasil uji coba yang telah dilakukan dan saran mengenai hal-hal yang masih bisa dikembangkan dari tugas akhir ini.

6.1 Kesimpulan

Dari hasil uji coba yang telah dilakukan terhadap implementasi penyelesaian permasalahan *Yet Another Assigment Problem* dapat diambil kesimpulan sebagai berikut:

1. Implementasi algoritma Hopcroft-Karp pada permasalahan *The Dilemma of Idli*, menghasilkan solusi yang benar dan dapat digunakan untuk mencari *maximum bipartite matching* pada suatu graf.
2. Implementasi algoritma Hopcroft-Karp dengan representasi *Bipartite Graph* dapat menyelesaikan permasalahan *Yet Another Assigment Problem* dengan benar.
3. Kompleksitas waktu yang dibutuhkan untuk seluruh proses sistem adalah $O((m + n)^2)$ pada m buah pekerjaan dan n orang pekerja sehingga algoritma ini dipengaruhi secara kuadratik baik oleh jumlah pekerja maupun jumlah pekerjaan.

6.2 Saran

Saran yang diberikan dalam pengembangan algoritma penyelesaian masalah penugasan adalah agar pada penelitian selanjutnya algoritma yang digunakan memiliki kompleksitas waktu yang lebih kecil dari $O((m + n)^2)$.

DAFTAR PUSTAKA

- [1] D. Jungnickel, Graphs, Networks and Algorithms, 4th Edition penyunt., Berlin: Springer, 2012.
- [2] H. A. Taha, Operations Research: An Introduction, 8th Edition penyunt., Upper Saddle River: Prentice Hall, 2007.
- [3] J. Bondy dan U. Murty, Graph Theory, 3rd penyunt., Berlin: Springer, 2008.
- [4] SPOJ, “The dilemma of Idli,” March 2014. [Online]. Available: <http://www.spoj.com/problems/WPC5G/>.
- [5] SPOJ, “Yet Another Assignment Problem,” June 2010. [Online]. Available: <http://www.spoj.com/problems/ASSIGN5/>.
- [6] S. Halim dan F. Halim, “Competitive Programming 3,” Singapore, Lulu Publisher, 2013.

BIODATA PENULIS



Muhammad Izzuddin, lahir pada tanggal 21 April 1994 di Sidoarjo, Jawa Timur. Penulis telah menempuh pendidikan formal mulai dari TK Muslimat VIII Islamiyah (1998-2000), SD Negeri Geluran III (2000-2006), SMP Negeri 1 Taman (2006-2009), SMA Negeri 1 Sidoarjo (2009-2012), dan terakhir sebagai mahasiswa di Jurusan Teknik Informatika ITS dengan bidang minat Dasar Terapan dan Komputasional. Memiliki beberapa hobi antara lain yaitu *travelling* dan bermain bola *volly*. Selama menempuh pendidikan di kampus penulis juga aktif dalam organisasi kemahasiswaan, antara lain Staf Departemen Kewirausahaan dan Minat Bakat Himpunan Mahasiswa Teknik Computer-Informatika pada tahun ke-2 serta Kepala Departemen Kewirausahaan dan Minat Bakat Himpunan Mahasiswa Teknik Computer-Informatika pada tahun ke-3. Semasa kuliah, penulis pernah mengikuti lomba Mandiri Hackathon E-Cash 2016 dan berhasil meraih juara 2 Nasional pada perlombaan tersebut. Selain itu penulis juga menjabat sebagai asisten mata kuliah dasar pemrograman, struktur data, pemrograman web serta riset operasional.