TUGAS AKHIR – IF184802

DESAIN DAN ANALISIS ALGORITMA UNTUK MENDAPATKAN ALIRAN MAKSIMAL PADA GRAF TIDAK BERARAH DAN BERBOBOT DALAM PENYELESAIAN PERMASALAHAN SPOJ ADAHOSE - ADA AND HOSE

MUHAMMAD PANDU PRAADHA

05111540000128

Dosen Pembimbing

Rully Soelaiman,S.Kom., M.Kom.

TUGAS AKHIR – IF184802

DESAIN DAN ANALISIS ALGORITMA UNTUK MENDAPATKAN ALIRAN MAKSIMAL PADA GRAF TIDAK BERARAH DAN BERBOBOT DALAM PENYELESAIAN PERMASALAHAN SPOJ ADAHOSE - ADA AND HOSE

MUHAMMAD PANDU PRAADHA

05111540000128

Dosen Pembimbing I

Rully Soelaiman, S.Kom., M.Kom.

[*Halaman ini sengaja dikosongkan*]

FINAL PROJECT – IF184802

DESIGN AND ANALYSIS OF ALGORITHMS FOR OBTAINING THE MAXIMUM FLOW ON NON-DIRECTED AND WEIGHTED GRAPHS IN THE SOLUTION TO THE PROBLEM OF SPOJ ADAHOSE - ADA AND HOSE

MUHAMMAD PANDU PRAADHA

05111540000128

Supervisor I

Rully Soelaiman, S.Kom., M.Kom.

[*Halaman ini sengaja dikosongkan*]

LEMBAR PENGESAHAN

DESAIN DAN ANALISIS ALGORITMA UNTUK MENDAPATKAN ALIRAN MAKSIMAL PADA GRAF TIDAK BERARAH DAN BERBOBOT DALAM PENYELESAIAN PERMASALAHAN SPOJ *ADAHOSE - ADA AND HOSE*

TUGAS AKHIR

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

Oleh:
MUHAMMAD PANDU PRAADHA
NRP: 05111540000128

Disetujui oleh Pembimbing Tugas Akhir:

1. Rully Soelaiman, S.Kom., M.Kom. .....
   (NIP. 19700213199402 1 001) (Pembimbing 1)

2. M.M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil.
   .....
   (NIP. 19740209200212 1 001) (Pembimbing 2)

SURABAYA
Juni, 2020

[Halaman ini sengaja dikosongkan]

DESAIN DAN ANALISIS ALGORITMA UNTUK MENDAPATKAN ALIRAN MAKSIMAL PADA GRAF TIDAK BERARAH DAN BERBOBOT DALAM PENYELESAIAN PERMASALAHAN SPOJ ADAHOSE - ADA AND HOSE

Nama Mahasiswa	: Muhammad Pandu Praadha
NRP	: 0511154000128
Department	: Teknik Informatika Fakultas Teknologi Elektro dan Informatika Cerdas - ITS
Dosen Pembimbing 1	: Rully Soelaiman, S.Kom., M.Kom.
Dosen Pembimbing 2	: M.M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil.

Abstrak

Permasalahan dalam buku Tugas Akhir ini diambil dari dunia nyata namun sudah disederhanakan ke dalam bentuk soal yang terdapat pada situs Sphere Online Jugde "ADAHOSE – Ada and Hose".Dalam permasalahan ini, diketahui bahwa Ada, seorang petani, memiliki sebuah ladang dengan selang yang mengelilingi tiap 1×1 bidang ladang.Terdapat pula sebuah sumur dan alat penyiram di atas dan di bawah ladang masing-masing.Selain itu selang yang sangat besar mengarah dari sumur ke atas ladang dan dari bawah ladang ke alat penyiram.Ada ingin menghitung total aliran-per-waktu yang mengalir dari sumur ke alat penyiram.

Tugas Akhir ini akan diimplementasikan dengan metode pencarian jalan terpendek pada sebuah graf tidak berarah dan berbobot. Penulis memilih algoritma Dijkstra dengan pertimbangan, algoritma Dijkstra merupakan metode pencarian jalan terpendek yang cepat dan menghasilkan solusi yang benar.

Dalam buku ini akan dibahas implementasi algoritma Dijkstra untuk mencari aliran maksimal dari sumur ke alat penyiram tersebut dengan menggunakan bahasa pemrograman

C++. Dari serangkaian percobaan yang telah dilakukan, didapatkan kesimpulan bahwa algoritma yang dirancang adalah sesuai dengan permasalahan ini dan algoritma tersebut dipengaruhi secara kuadrat oleh ukuran ladang.

Kata kunci: aliran maksimal, graf tak berarah, graf berbobot.

DESIGN AND ANALYSIS OF ALGORITHMS FOR OBTAINING THE MAXIMUM FLOW ON NON-DIRECTED AND WEIGHTED GRAPHS IN THE SOLUTION TO THE PROBLEM OF SPOJ ADAHOSE - ADA AND HOSE

Student Name	: Muhammad Pandu Praadha
Registration Number	: 05111540000128
Department	: Informatics Department Faculty of Intelligent Electrical and Informatics Technology - ITS
First Supervisor	: Rully Soelaiman, S.Kom., M.Kom.
Second Supervisor	: M.M. Irfan Subakti, S.Kom. M.Sc.Eng., M.Phil.

Abstract

The problem in this final project has een taken from the real world bit it's has been simplified into the form of questions which can be found on the Sphere Online Judge site "Adahose - Ada and Hose". In this problem, it is known that Ada, the farmer, owns a field with hose wrapped around each 1×1 sub-field. There are also a well and a sprinkle above and below the field respectively. Additionally very big hoses are lead from well to top of the field and from bottom to the sprinkler. Ada wants to calculate the maximum total flow-per-time which goes from well to sprinkler.

This Final Project will implement a searching method for a shortest path on an undirected and weighted graph. The author chose the Dijkstra algorithm with consideration, the Dijkstra algorithm is a method of finding the shortest path that is fast and produces the correct solution.

In this book, the Dijkstra algorithm will be discuss to find the maximum flow from the well to the sprinkler which will be implemented by using the C++ programming language. From the

experiment which have been done, it can be concluded that the algorithm designed is in accordance with this problem and the algorithm is influenced quadraticallyby the size of the field.

Keywords: maximum flow, non-directed graph, weighted graph.

KATA PENGANTAR

Puji syukur kepada Allah yang Maha Kuasa karena berkat rahmat-Nya saya dapat melaksanakan Tugas Akhir yang berjudul:

"DESAIN DAN ANALISIS ALGORITMA UNTUK MENDAPATKAN ALIRAN MAKSIMAL PADA GRAF TIDAK BERARAH DAN BERBOBOT DALAM PENYELESAIAN PERMASALAHAN SPOJ ADAHOSE - ADA AND HOSE"

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

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

5. Teman-teman dari wardug yang selalu memberikan dukungan untuk penyelesaian tugas ini.
6. Mas Atam selaku pemilik wardug yang telah menyediakan tempat dan wifi bagi saya untuk menyelesaikan Tugas Akhir ini.
7. Findryan dan Adit sebagai teman seperjuangan yang telah memberikan saya masukan dan bantuan dalam menyelesaikan penulisan buku dan program Tugas Akhir ini.
8. Seluruh mahasiswa Informatika ITS angkatan 2015 yang telah menjadi teman penulis selama menjalani masa kuliah di Informatika ITS.
9. Serta semua pihak yang yang telah turut membantu penulis dalam menyelesaikan Tugas Akhir ini.

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

Surabaya, Juni 2020

DAFTAR ISI

[Halaman ini sengaja dikosongkan]

Daftar Gambar

DaftarTabel

[Halaman ini sengaja dikosongkan]

Daftar Kode Sumber

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar Tugas Akhir (TA) yang meliputi latar belakang, rumusan permasalahan, batasan permasalahan, tujuan pembuatan TA, manfaat Tugas Akhir, metodologi, dan sistematika penulisan buku TA ini.

1.1 Latar Belakang

Perkembangan dunia teknologi dalam beberapa tahun terakhir sangatlah pesat tak terkecuali pada perkembangan ilmu matematika.diantaranya adalah penggunaan *Linear Programming* dan *Non-Linear Programming*. Selain itu terdapat pula penggunaan vektor dan matriks dalam model matematika , yaitu dalam teori graf. Teori graf digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut.

Teori graf merupakan salah satu ilmu yang dibahas dalam matematika yang mempelajari himpunan titik yang dihubungkan oleh himpunan sisi.Sebuah graf terdiri atas dua himpunan yaitu himpunan tak kosong *V* yang elemen–elemennya disebut titik dan himpunan *E* yang elemen–elemennya disebut sisi.Salah satu kajian dalam teori graf adalah jaringan.Jaringan merupakan kumpulan titik dan sisi yang saling berkaitan dengan arah tertentu.Didalam jaringan terdapat beberapa model yang bisa digunakan dalam membantu memecahkan masalah, salah satunya adalah permasalahan penentuan potongan minimal (*minimum cut*).Potongan minimal adalah jumlah potongan minimal yang membagi simpul grafik yang disatukan oleh setidaknya satu sisi menjadi dua himpunan bagian yang terpisah.Potongan minimal juga dapat mencari aliran maksimal pada graf berbobot dengan menjumlahkan berat pada sisi yang dipotong.Pengaplikasian potongan minimal terhadap kehidupan nyata sangatlah beragam salah satunya adalah permasalahan pada SPOJ ADAHOSE - *Ada and Hose* yaitu untuk menentukan jumlah aliran (satuan/waktu) maksimal yang dibutuhkan dari keran air ke alat penyiram.

Topik TA ini akan mengangkat permasalahan yang terdapat pada *Online Judge* SPOJ ADAHOSE - *Ada and Hose* tentang penentuan aliran maksimal pada graf tidak berarah dan berbobot. Pada permasalahan ini, diberikan suatu bilangan bulat N yang memiliki nilai antara 1 sampai 1000. Nilai dari N akan merepresentasikan ukuran kebun. Kemudian pada setiap baris akan mewakili ukuran selang yang terhubung tiap 1 satuan kuadrat kebun.

Permasalahan ini memiliki tantangan untuk merancang algoritma yang dapat menemukan potongan terpendek yang akan digunakan untuk mencari aliran maksimal ke alat penyiram.

Hasil dari TA ini diharapkan dapat menentukan implementasi algoritma yang tepat untuk memecahkan permasalahan di atas secara optimal dan diharapkan dapat memberikan kontribusi pada perkembangan ilmu pengetahuan dan teknologi informasi.

1.2 Rumusan Permasalahan

Rumusan masalah yang diangkat dalam TA ini adalah sebagai berikut:

1. Bagaimana menganalisis dan menentukan algoritma yang tepat untuk menyelesaikan permasalahan SPOJ ADAHOSE - Ada and Hose?
2. Bagaimana mengimplementasikan algoritma yang sudah didesain untuk menyelesaikan permasalahan SPOJ ADAHOSE - Ada and Hose secara efisien?
3. Bagaimana menguji implementasi algoritma yang sudah dirancang untuk mengetahui kinerja dari implementasi yang telah dibuat?

1.3 Batasan Permasalahan

Permasalahan yang dibahas pada TA ini memiliki beberapa batasan, sebagai berikut:

1. Implementasi algoritma menggunakan bahasa pemrograman C++.
2. Kebutuhan memori hasil implementasi mengacu pada hasil keluaran dari server situs SPOJ untuk permasalahan ADAHOSE – *Ada and Hose*.
3. Algoritma diimplementasikan untuk menemukan jumlah potongan minimal pada graf tidak berarah dan berbobot

Adapun batasan dalam SPOJ untuk Tugas Akhir ini adalah sebagai berikut :

1. Batas jumlah kota adalah *N* dengan rentang 1 sampai 1000.
2. *Test case* yang digunakan adalah *test case* yang terdapat pada permasalahan SPOJ ADAHOSE - *Ada and Hose*.
3. Batas maksimal waktu untuk mendapatkan potongan minimal adalah 2 detik.
4. Batas maksimal ukuran file *source code* yang dihasilkan adalah 50000B.
5. Batas maksimal memori RAM yang digunakan untuk pemrosesan *dataset* adalah 1536MB.

1.4 Tujuan Pembuatan Tugas Akhir

Tujuan dari TA ini adalah sebagai berikut:

1. Melakukan analisis dan mendesain algoritma untuk menyelesaikan permasalahan SPOJ ADAHOSE - Ada *and Hose* dengan pembuktian (*proofing*) yang jelas.
2. Mengimplementasikan algoritma yang sudah didesain untuk menyelesaikan permasalahan SPOJ ADAHOSE - *Ada and Hose* secara efisien.
3. Menguji implementasi dari algoritma yang telah didesain untuk mengetahui kinerja algoritma yang telah dibuat untuk permasalahan SPOJ ADAHOSE - *Ada and Hose*.

1.5 Manfaat Tugas Akhir

TA ini diharapkan dapat membantu untuk memahami algoritma dalam menentukan jumlah potongan minimal pada graf

tidak berarah dan berbobot serta dapat mengimplementasikan solusi dari permasalahan SPOJ ADAHOSE – *Ada and Hose*sehingga dapat memberikan kontribusi pada pengembangan ilmu pengetahuan dan teknologi informasi.

1.6 Metodologi

Langkah-langkah yang ditempuh dalam pengerjaan TA ini yaitu:

1.6.1 Penyusunan Proposal TA

Tahap awal untuk memulai pengerjaan TA adalah penyusunan proposal TA.Pada proposal ini, penulis mengajukan gagasan untuk menyelesaikan permasalahan dalam menentukan jumlah potongan minimal pada graf tidak berarah dan berbobot pada studi kasus SPOJ ADAHOSE - *Ada and Hose*.

1.6.2 Studi Literatur

Pada tahap ini dilakukan pencarian informasi dan studi literatur yang relevan untuk dijadikan referensi dalam melakukan pengerjaan TA.Informasi didapatkan dari materi-materi yang berhubungan dengan graf dan algoritma-algoritma dalam penentuan jumlah potongan minimal. Informasi tersebut didapatkan dari buku, internet, dan materi kuliah yang berhubungan dengan metode yang akan digunakan.

1.6.3 Implementasi Perangkat Lunak

Tahapan ini merupakan tahapan untuk membangun algoritma yang akan digunakan. Pada tahap ini dilakukan implementasi dari rancangan struktur data yang akan dimodelkan sesuai dengan permasalahan. Implementasi ini dilakukan dengan menggunakan bahasa pemrograman C++.

1.6.4 Pengujian dan Evaluasi

Pada tahap ini dilakukan ujicoba kebenaran dan kinerja solusi yang telah diimplementasikan dengan melakukan pengiriman sumber kode sistem ke situs penilaian *Sphere Online*

Judge (SPOJ) pada permasalahan yang terkait dan melihat hasil umpan balik.Pengujian dilakukan dengan membandingkan kompleksitas hasil ujicoba dengan kompleksitas hasil analisis.

1.6.5 Penyusunan Buku TA

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

1.7 Sistematika Penulisan

Buku TA ini merupakan laporan secara lengkap mengenai TA yang telah dikerjakan baik dari sisi teori, rancangan, maupun implementasi sehingga memudahkan bagi pembaca dan juga pihak yang ingin mengembangkan lebih lanjut. Sistematika penulisan buku TA secara garis besar sebagai berikut:

Bab I Pendahuluan

Bab ini berisi penjelasan latar belakang, rumusan masalah, batasan masalah dan tujuan pembuatan TA.Selain itu, metodologi pengerjaan dan sistematika penulisan laporan TA juga menjadi bahan pembahasan.

Bab II Dasar Teori

Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang dan teori-teori yang digunakan untuk mendukung pembuatan TA ini.

Bab III Analisis dan Perancangan

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

Bab IV Implementasi

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

Bab V Uji Coba dan Evaluasi

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

Bab VI Kesimpulan dan Saran

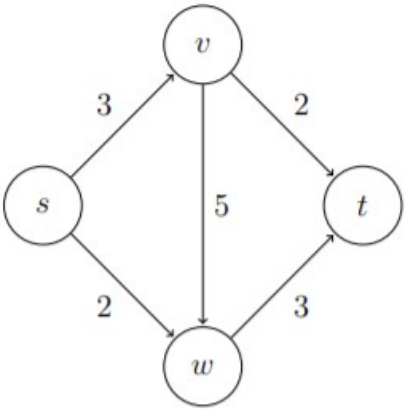
Bab ini merupakan bab terakhir yang menjelaskan kesimpulan dari hasil ujicoba yang dilakukan dan saran untuk pengembangan perangkat lunak selanjutnya.

BAB II
DASAR TEORI

Pada bab ini akan dijelaskan mengenai dasar teori yang menjadi dasar pengerjaan TA ini.

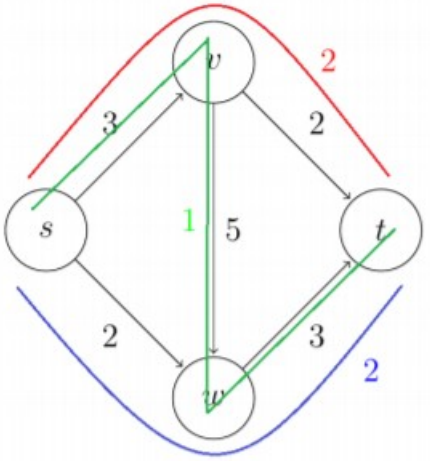
2.1 Deskripsi Umum Permasalahan

Permasalahan yang diangkat para TA ini adalah permasalahan pencarian jumlah aliran maksimal atau jumlah aliran maksimal yang dapat melewati sebuah jaringan dari titik sumber ke titik tujuan. Permasalahan ini akan merepresentasikan simpul pada graf sebagai persimpangan yang menghubungkan beberapa selang dan sisi yang menjadi penghubung antar simpul merepresentasikan selang. Dari graf tersebut akan dicari aliran maksimal sebagai representasi dari jumlah aliran maksimal yang dapat dilewati selang-selang yang terhubung dari sumber ke tujuan sehingga didapatkan jumlah aliran maksimal yang keluar dari sumber. Sebagai contoh, persoalan sederhana dengan simulasi penyelesaian untuk mendapatkan aliran maksimal, dijelaskan seperti berikut ini:



Gambar 2.1 Contoh Permasalahan Aliran Maksimal

Untuk mencari aliran maksimal pada Gambar 2.1, maka perlu dicari jalan yang menghubungkan sumber(*s*) dan tujuan(*t*) lalu mencari aliran maksimal yang memungkinkannya dapat mengalir melewati jalan tersebut. Jalan yang didapatkan adalah: *s*-*v*-*t* dengan aliran maksimal 2, *s*-*v*-*w*-*t* dengan aliran maksimal 1, dan *s*-*w*-*t* dengan aliran maksimal 2 seperti Gambar 2.2.



Gambar 2.2 Contoh Solusi Aliran Maksimal

Maka aliran maksimal yang didapatkan dari jaringan pada Gambar 2.1 adalah seperti di bawah ini.

s-v-t	= 2
s-v-w-t	= 1
s-w-t	= 2
jumlah	= 5

2.2 Definisi Umum

Aliran maksimal dari graf berbobot didefinisikan sebagai aliran yang memungkinkannya mengalir melalui satu-sumber, satu-tujuan jaringan aliran yang maksimal.

Beberapa pendekatan untuk menyelesaikan permasalahan aliran maksimal adalah pendekatan menggunakan algoritma *Naïve Greedy* dan menggunakan *Residual Graph* [1].

2.2.1 Algoritma *Naïve Greedy*

Pendekatan *Greedy* paling sederhana untuk masalah aliran maksimal adalah dimulai dengan semua aliran yang bernilai nol.Kemudian dengan pendekatan *greedy* dapatkan aliran dengan nilai yang semakin tinggi.Cara untuk melanjutkan dari satu jalan ke jalan berikutnya adalah mengirim banyak aliran pada beberapa jalur dari *s* ke *t*.

Algoritma Naïve Greedy

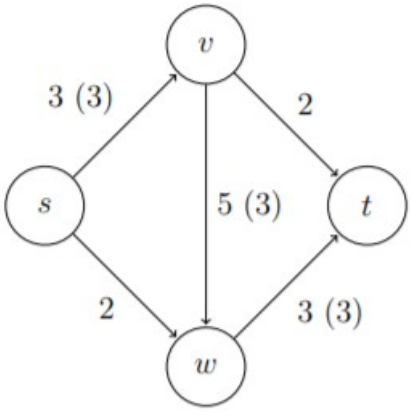
initialize $f_e = 0$ for all $e \in$ E
repeat
search for an s-t path *P* such that $f_e < u_e$ for every $e \in P$
if no such path **then**
halt with current flow $\{fe\}_{e \in E}$
else
let $\Delta = \min_{e \in P}(u_e - f_e)$
for all edges *e* of *P***do**
increase *fe* by Δ

Kode Sumber 2.1 *Pseudocode* Algoritma *Naïve Greedy*

Cara mencari jalan dari *s* ke *t* sehingga $f_e < u_e$ dapat dilakukan menggunakan BFS, algoritma pencarian yang mengunjungi suatu simpul kemudian mengunjungi semua simpul yang bertetangga dengan simpul tersebut, atau DFS, algoritma pencarian yang mengunjungi suatu simpul kemudian mengunjungi salah satu tetangganya dan seterusnya.

Algoritma diatas masih belum sempurna jika diterapkan ke graf pada Gambar 2.1. Ada kemungkinan pencarian jalan pertama mendapatkan jalan $P = s\text{-}v\text{-}w\text{-}t$ sehingga $\Delta = min\{3, 5, 3\}$

= 3 (Gambar 2.3). Karena aliran pada tepi(*s*, *v*) dan tepi (*w*, *t*) sudah penuh, maka tidak ada lagi jalan yang memenuhi kondisi $f_e<u_e$dan aliran maksimal yang didapatkan adalah 3 sedangkan sudah diketahui bahwa aliran maksimal dari graf pada Gambar 2.1 adalah 5 seperti yang ditunjukkan pada Gambar 2.2.



Gambar 2.3 Hasil yang Mungkin Didapat jika Menggunakan Algoritma*Naïve Greedy*

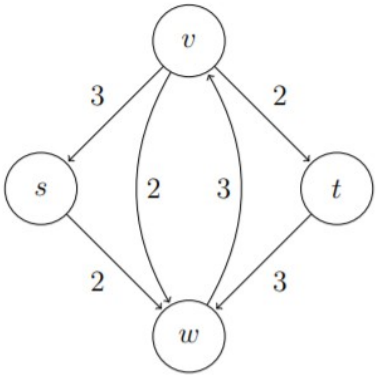
2.2.2 *Residual Graph*

Pendekatan ini memperluas algoritma *Naïve Greedy* dengan mengenalkan operasi "*undo*". Misalnya, ketika pada algoritma *Naïve Greedy*didapatkan hasil seperti Gambar 2.3 yang sebenarnya harus mengalirkan 2 unit lagi di tepi (*s*, *w*), kemudian mundur di tepi (*v*, *w*), membatalkan 2 dari 3 unit yang telah dialirkan pada iterasi sebelumnya, dan terakhir di tepi (*v*, *t*).

Dibutuhkan cara untuk menentukan operasi "undo" yang diizinkan. Ini memberi ide lahirnyacara sederhana tapi penting, yaitu *residual graph*. Idenya adalah diberikan sebuah graf *G* dan sebuah aliran *f*, lalu dibuat graf baru G_f yang memiliki simpul yang sama dengan graf *G* dan dua tepi untuk tiap tepi di graf *G*. Tepi *e*(*v*, *w*) di graf *G* memunculkan tepi depan (*v*, *w*) di graf G_f dengan kapasitas $u_e - f_e$ (sisa kapasitas) dan tepi belakang (w, v) denga kapasitas f_e (jumlah aliran yang sebelumnya dialirkan di

tepi (v, w) pada graf G) seperti pada Gambar 2.4. Jika diterapkan pada algoritma *Naïve Greedy* maka akan menghasilkan graf seperti Gambar 2.5 dan iterasi dapat dilanjutkan karena saat mencari jalan dari s ke t masih mendapatkan jalan baru dengan $\Delta = \min\{2, 3, 2\} = 2$ sehingga aliran maksimalnya adalah 3 + 2 = 5, sesuai dengan aliran maksimal yang sudah diketahui (Gambar 2.2).

Gambar 2.4(a) Tepi dan Aliran di Graf Gdan (b) Tepi yang dihasilkan di Graf Residual G_f



Gambar 2.5 Graf Residual dari Gambar 2.3

2.3 *Flow Network* dan *Maximum Flow*

Dalam teori graf, *flow network* atau aliran jaringan didefinisikan sebagai graf berarah yang melibatkan titik sumber(S) dan titik tujuan(T) dan beberapa simpul lain yang terhubung dengan tepi. Setiap tepi memiliki kapasitas masing-masing yang merupakan batas maksimal aliran yang memungkinkan mengalir di tepi tersebut.

Maximum Flow atau aliran maksimal didefinisikan sebagai jumlah aliran maksimal yang diperbolehkan mengalir dari titik sumber ke titik tujuan [2].

2.4 *Cut* dan *Minimum Cut*

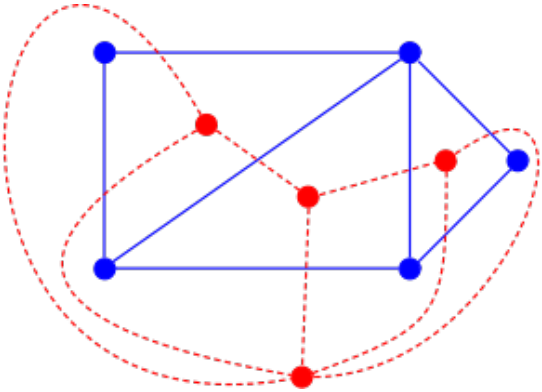
Cut adalah potongan dari simpul-simpul graf menjadi dua himpunan bagian yang terpisah. *Minimum cut* adalah potongan pada suatu fase yang memiliki ukuran atau kapasitas potongannya paling kecil dari kapasitas potongan pada fase lainnya [6].

2.5 Teorema *Max-Flow Min-Cut*

Teorema *Max-Flow Min-Cut* menyatakan bahwa potongan simpul kapasitas minimum dari suatu jaringan *N*sama dengan aliran maksimal yang dapat mengalir di sepanjang jaringan itu [7].

2.6 Graf Dual

Graf dual pada graf *G* adalah graf *G** yang diperoleh dari suatu graf bidang dengan cara setiap daerah, bangun datar yang terbentuk dari beberapa simpul dan tepi yang terhubung, diwakili dengan satu titik, dan antar titik akan terhubung langsung jika daerah tersebut saling berbatasan langsung.



Gambar 2.6 Contoh Graf G (Graf Berwarna Biru) dan Graf Dual G* (Graf Berwarna Merah) yang Terbentuk dari Graf G

2.7 Algoritma Ford-Fulkerson

Dikembangkan oleh L. R. Ford, Jr. and D. R. Fulkerson pada tahun 1956.

Masukan yang dibutuhkan adalah graf ***G***, simpul sumber ***S***, dan simpul tujuan ***T*** [2].

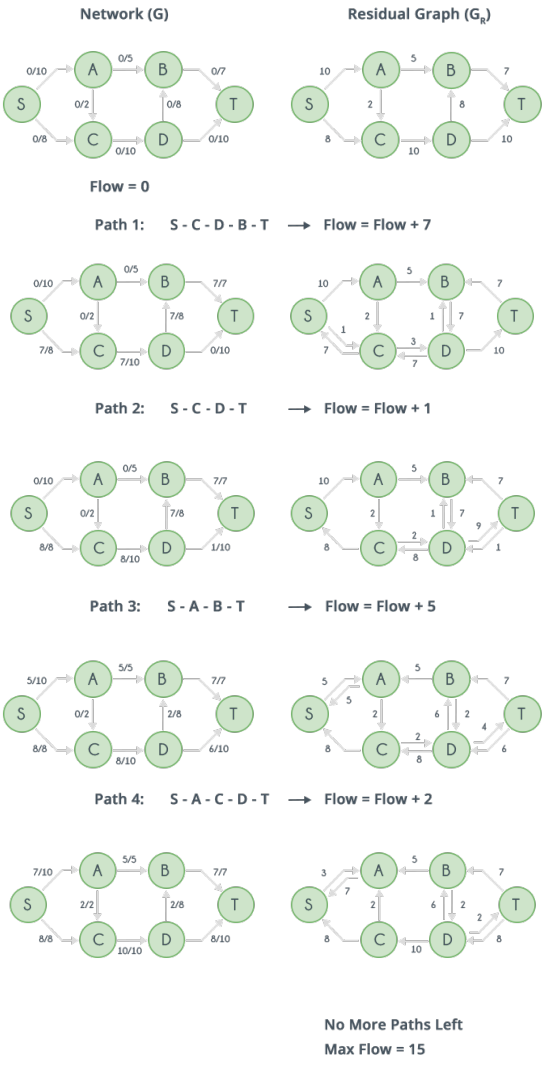
Algoritma Ford-Fulkerson

```
initialise flow in all edges to 0
while (there exists an augmenting path(P) between S and T
in residual network graph):
    Augment flow between S to T along the path P
    Update residual network graph
return
```

Kode Sumber 2.2 *Pseudocode* Algoritma Ford-Fulkerson

Augmented path adalah jalan dari sumber ke tujuan yang tidak melibatkan siklus dan hanya melewati tepi dengan bobot lebih dari 0. Jika tidak ada lagi *augmented path* maka total aliran adalah aliran maksimal.

Simulasi penggunaan algoritma Ford-Fulkerson dalam pencarian aliran maksimal dapat dilihat pada Gambar 2.7.



Gambar 2.7 Contoh Langkah-Langkah Simulasi Algoritma Ford-Fulkerson

2.8 Algoritma Dinic

Di tahun 1970, Y. A. Dinitz mengembangkan algoritma yang lebih cepat untuk menghitung aliran maksimal di sebuah jaringan. Algoritma ini menerapkan pembuatan graf level dan graf residual dan mencari *augmenting path* bersama dengan aliran penghalang [2].

Graf level adalah graf yang nilai tiap simpulnya adalah jarak terpendek dari simpul sumber.

Aliran penghalang termasuk mencari jalan baru dari simpul *bottleneck*.

Masukan yang dibutuhkan adalah graf ***G***, simpul sumber ***S***, dan simpul tujuan ***T.***

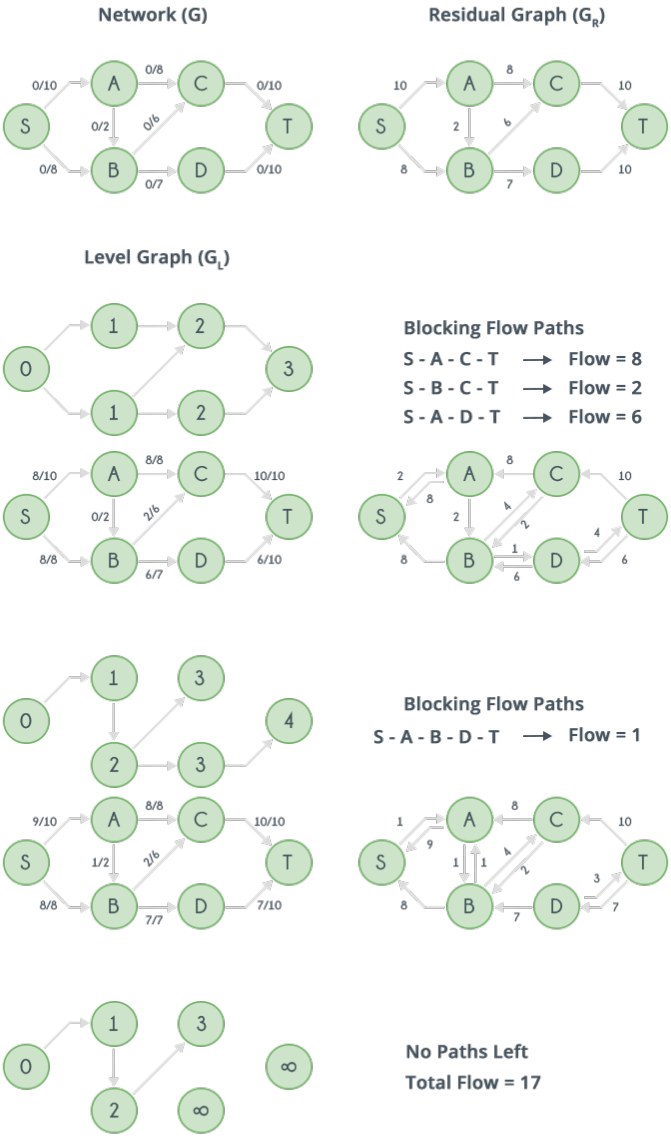
Algoritma Dinic

```
Initialize flow in all edges to 0, F = 0
Construct level graph
while (there exists an augmenting path in level graph):
find blocking flow f in level graph
      F = F + f
      Update level graph
return F
```

Kode Sumber 2.3 *Pseudocode* Algoritma Dinic

Pembaruan graf level termasuk penghapusan tepi dengan kapasitas penuh. Penghapusan simpul yang bukan sumber tapi simpul yang tidak terhubung ke tujuan.

Simulasi penggunaan algoritma Ford-Fulkerson dalam pencarian aliran maksimal di Gambar 2.8.



Gambar 2.8 Contoh Langkah-Langkah Simulasi Algoritma Dinic

2.9 Algoritma *Dijkstra*

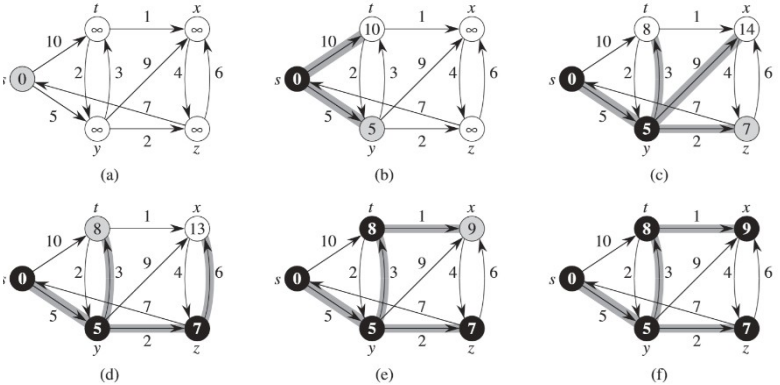
Algoritma Dijkstra menyelesaikan permasalahan jalan terpendek dengan satu sumber dari graf berarah dan berbobot $G = (V, E)$ dengan asumsi bobot semua tepi tidak bernilai negatif (bobot$w(u, v) \geq 0$ untuk tiap tepi $(u, v) \in E$).

Algoritma Dijkstra mempertahankan sebuah *set* S dari simpul-simpul yang bobot jalan terpendek dari sumber s telah ditentukan terlebih dahulu. Algoritma ini berulang-ulang memilih simpul $u \in V - S$ dengan perkiraan jalan terpendek minimum, menambahkan u ke S, dan me-*relax* semua tepi yang meninggalkan u. Diimplementasi dengan *min-priority queue* Q dari simpul-simpul dengan *key* adalah nilai d nya [1].

```
Dijkstra ( G, w, s )
1.  Initialize-Single-Source(G, s)
2.  S = ∅
3.  Q = G.V
4.  while Q ≠ ∅
5.      u = Extract-Min(Q)
6.      S = S ∪ {u}
7.      for each vertex v ∈ G.Adj[u]
8.          Relax(u, v, w)
```

Kode Sumber 2.4 *Pseudocode* Algoritma Dijkstra

Simulasi penggunaan algoritma Dijkstra dalam pencarian jalan terpendek di Gambar 2 .9.



Gambar 2.9 Contoh Langkah-Langkah Simulasi Algoritma Dijkstra

2.10 Relaxation

Algoritma Dijkstra pada subbab 2.9 menggunakan teknik yang dinamakan *relaxation* [1].Untuk tiap simpul $v \in V$, dipertahankan atribut $v.d$ yang merupakan batas atas dari bobot jalan terpendek dari sumber s ke v. $v.d$ disebut dengan perkiraan jalan terpendek. Cara mencari perkiraan jalan terpendek digunakan prosedur berikut:

Initialize-Single-Source(G, s)
1. **for each** vertex $v \in G.V$
2. $v.d = \infty$
3. $v.\pi$ = NIL
4. $s.d = 0$

Kode Sumber 2.5 *Pseudocode* Fungsi Inisialisasi Graf dengan Satu Sumber

Setelah diinisialisasi, kita memiliki $v.\pi$ = NIL untuk semua $v \in V$, $s.d = 0$, dan $v.d = \infty$ untuk $v \in V - \{s\}$.

Proses *relaxation* sebuah tepi (u, v) terdiri dari percobaan mencari apakah bisa memperbaiki jalan terpendek ke v yang telah ditemukan sejauh ini dengan melalui u dan, jika didapatkan, memperbarui nilai $v.d$ dan $v.\pi$. Langkah *relaxation* mungkin mengurangi nilai dari perkiraan jarak terpendek $v.d$ dan

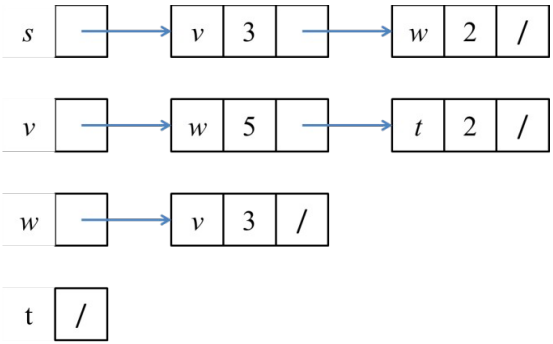
memperbarui atribut $v.\pi$ pendahulu v. berikut adalah sumber kode relaxation pada tepi (u, v) dengan waktu O(1):

```
Relax ( u, v, w )
1.  if v.d>u.d + w(u, v)
2.      v.d = u.d + w(u, v)
3.      v.π = u
```

Kode Sumber 2.6 *Pseudocode* Fungsi metode *Relaxation*

2.11 Adjacency List

Adjacency list adalah salah satu cara merepresentasikan graf $G = \{V, E\}$ sebagai *array of list.*Ukuran *array* adalah jumlah simpul.Isi dari *array*[*i*] adalah list dari simpul-simpul yang bertetangga dengan simpul *i*. Representasi ini juga bisa diterapkan untuk merepresentasikan graf berbobot.Bobot dari tepi direpresentasikan dengan *list of pair* [3].



Gambar 2.10 Ilustrasi *Adjacency List*

2.12 Permasalahan ADAHOSE pada SPOJ

ADAHOSE - *Ada and Hose* merupakan suatu permasalahan yang terdapat pada situs penilaian SPOJ dengan deskripsi soal dari sumber asli menggunakan bahasa Inggris seperti pada Gambar 2.11.

As you might already know, Ada the Ladybug is a farmer. She owns a square field. There is a hose wrapped around each **1x1** sub-field. All hoses are additionally combined into a bigger hose everywhere where they touch (so the water can arbitrarily flow between both of them [or even all four of them]). Each hose has its own flow-per-time attribute.

There is a big well (an infinite source of water) above the field and a big sprinkler under the field. Additionally very big hoses (for our case we can consider them infinetly big) are lead from well to top of the field and from bottom to the sprinkler. Your quest is simple: Calculate tha maximal total flow-per-time which goes from well to sprinkler.

Gambar 2.11 Deskripsi Permasalahan ADAHOSE

Pada permasalahan ADAHOSE diceritakan bahwa Ada, seorang petani, memiliki sebuah ladang berbentuk persegi.Terdapat sebuah selang yang mengelilingi tiap 1x1 bagian ladang.Tiap selang memiliki batas aliran-per-waktu sendiri-sendiri.Semua selang terhubung menjadi selang yang lebih besar dimana tiap selang bersentuhan.

Terdapat pula sumur besar diatas ladang dan alat penyiram dibawah ladang.Selain itu selang sangat besar (dapat mengalirkan unit tak terbatas aliran-per-waktu) menghubungkan antara selang-selang di ladang dengan sumur dan alat penyiram. Kita ditugaskan untuk menghitung aliran-per-waktu yang dibutuhkan dari sumur ke alat penyiram [4].

Input

The first line of input contains an integer $\mathbf{1 \le N \le 1000}$, the size of field.

The next **N** lines contains **N** integers $\mathbf{0 \le A_{ij} \le 1000}$, the size of each hose wrapped around.

Gambar 2.12 Deskripsi Masukan Permasalahan ADAHOSE

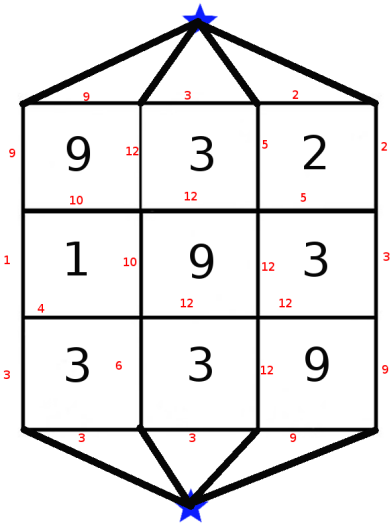
Berdasarkan gambar 2.12, dalam permasalahan ini diberikan masukan pada baris pertama satu angka dengan rentang $1 \leq N \leq 1000$ yang merepresentasikan ukuran ladang. Kemudian pada N baris berikutnya diberikan masukan berupa bilangan-bilangan yang akan mewakili ukuran selang yang mengelilingi bagian ladang. Kemudian dari persoalan tersebut akan menghasilkan keluaran berupa satu bilangan yaitu jumlah aliran maksimal yang akan keluar di alat penyiram (Gambar 2.13) [4].

Output

Output the maximal flow-per-time achievable.

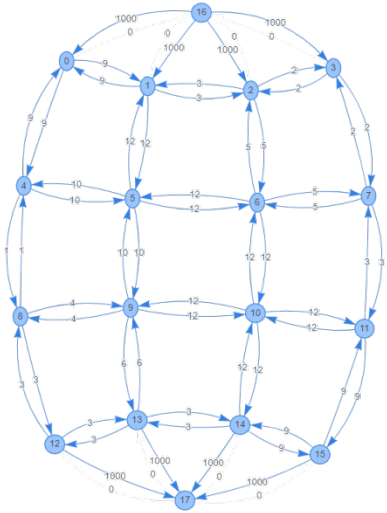
Gambar 2.13 Deskripsi Keluaran Permasalahan ADAHOSE

Sebagai contoh pada permasalahan berikut, diberikan suatu ilustrasi ladang dan selang-selang yang mengelilingi ladang dan yang menghubungkan selang di ladang dengan sumur dan alat penyiram (Gambar 2.14) [4].

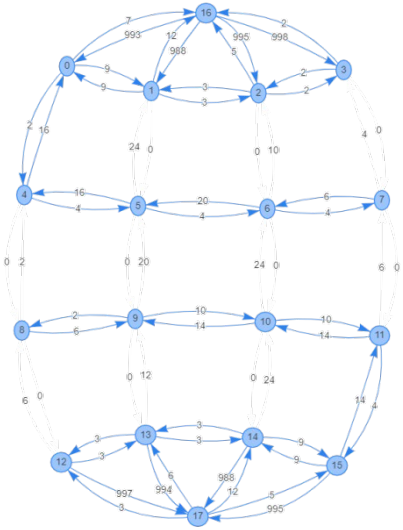


Gambar 2.14 Ilustrasi Ladang Ada dan Selang-Selang yang Mengelilinginya

Dari gambar 2.14 di atas, jika diubah menjadi graf akan terbentuk graf G seperti pada Gambar 2.15 dan setelah dilakukan pencarian aliran maksimal graf G akan berubah menjadi graf G seperti pada Gambar 2.16.



Gambar 2.15 Ilustrasi Graf *G* yang Dihasilkan dari Contoh Masukan

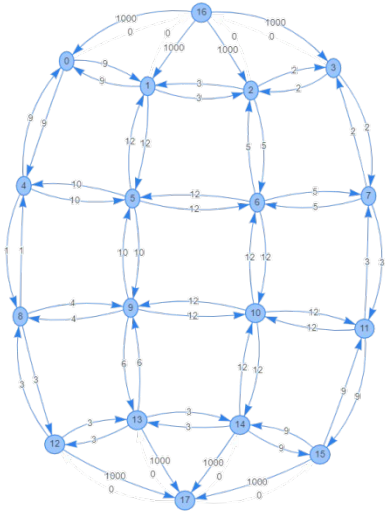


Gambar 2.16 Ilustrasi Hasil Pencarian Aliran Maksimal pada Graf *G* dengan Tepi yang Alirannya Penuh telah Dihapus

2.13 Penyelesaian Permasalahan ADAHOSE pada SPOJ dengan Algoritma Ford-Fulkerson

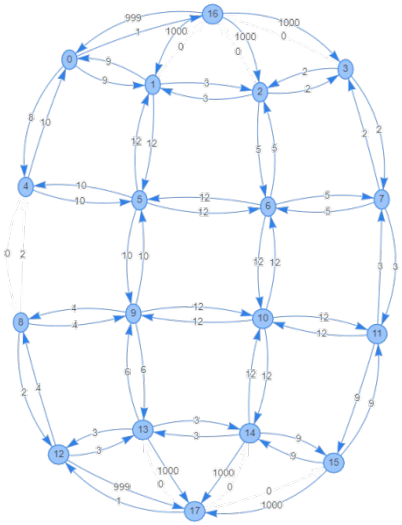
Diberikan contoh permasalahan soal ADAHOSE yaitu terdapat sebuah ladang dengan selang yang mengelilingi tiap bagian ladang dan juga terhubung ke sumur dan alat penyiram melalui selang yang sangat besar.Selang-selang tersebut diilustrasikan menjadi sebuah graf G (Gambar 2.17). Setiap sisi memiliki bobot sesuai berdasarkan jumlah aliran-per-waktu yang dapat dilewati antara 2 selang yang bersentuhan atau aliran-per-waktu 1 selang (tidak bersentuhan dengan selang lain).

Sumur direpresentasikan dengan simpul 16, dan alat penyiram direpresentasikan dengan simpul 17. Seperti yang disebutkan di deskripsi permasalahan bahwa selang yang menghubungkan sumur dengan selang di ladang dan alat penyiram dengan selang di ladang sangat besar dan pada batasan bahwa nilai maksimal pada selang adalah 1000, maka kapasitas aliran yang menghubungkan antara simpul 16 dan 17 dengan simpul yang terhubung dimisalkan dengan 1000.



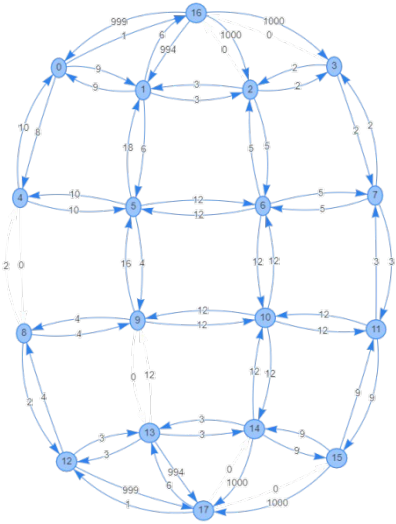
Gambar 2.17 Ilustrasi Graf G yang Dihasilkan dari Contoh Masukan

Langkah-langkah pencarian aliran maksimal dengan algoritma Ford-Fulkerson dalam permasalahan pada Gambar 2.17 dapat dijelaskan melalui beberapa iterasi seperti pada Gambar 2.18 sampai dengan Gambar 2.26.



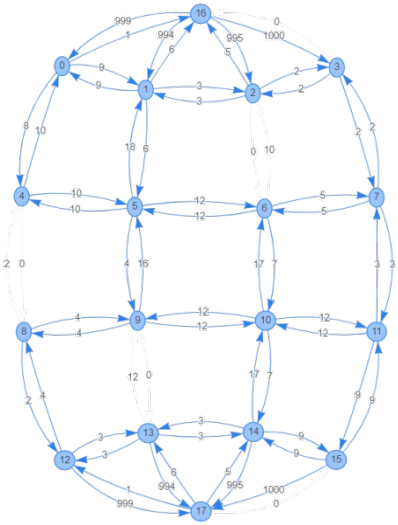
Gambar 2.18 Ilustrasi Graf Residual yang Dihasilkan Setelah Iterasi Pertama

Gambar 2.18 menunjukkan aliran residual setelah iterasi pertama pada pencarian aliran maksimal. Pada iterasi ini didapatkan *augmenting path* pada graf, yaitu 16 – 0 – 4 – 8 – 12 – 17 dengan aliran f_e = 1. Tepi (4, 5) tidak dimasukkan lagi pada pencarian *augmenting path* karena aliran pada tepi (4,5) sudah penuh.



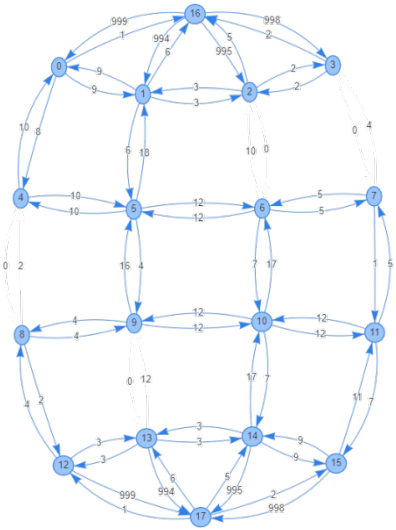
Gambar 2.19 Ilustrasi Graf Residual yang Dihasilkan Setelah Iterasi Kedua

Gambar 2.19 menunjukkan aliran residual setelah iterasi kedua pada pencarian aliran maksimal. Pada iterasi ini didapatkan *augmenting path* pada graf, yaitu 16 – 1 – 5 – 9 – 13 – 17 dengan aliran f_e = 6. Tepi (9, 13) tidak dimasukkan lagi pada pencarian *augmenting path* karena aliran pada tepi (9, 13) sudah penuh.



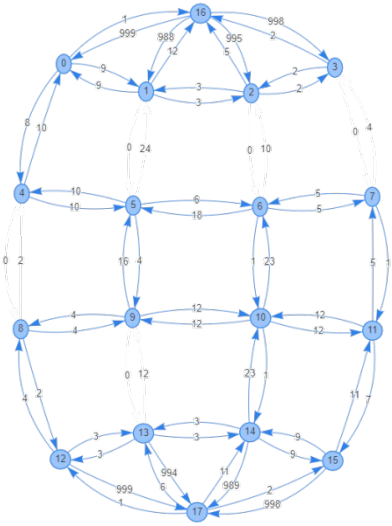
Gambar 2.20 Ilustrasi Graf Residual yang Dihasilkan Setelah Iterasi Ketiga

Gambar 2.20 menunjukkan aliran residual setelah iterasi ketiga pada pencarian aliran maksimal. Pada iterasi ini didapatkan *augmenting path* pada graf, yaitu 16 – 2 – 6 – 10 – 14 – 17 dengan aliran f_e = 6. Tepi (2, 6) tidak dimasukkan lagi pada pencarian *augmenting path* karena aliran pada tepi (2, 6) sudah penuh.



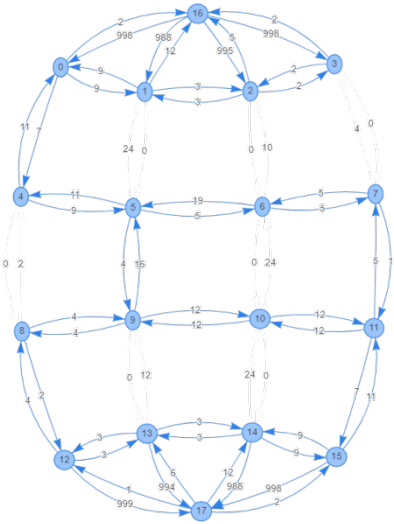
Gambar 2.21 Ilustrasi Graf Residual yang Dihasilkan Setelah Iterasi Keempat

Gambar 2.21menunjukkan aliran residual setelah iterasi keempat pada pencarian aliran maksimal. Pada iterasi ini didapatkan *augmenting path* pada graf, yaitu 16 – 3 – 7 – 11 – 15 – 17 dengan aliran $f_e = 2$. Tepi (3, 7) tidak dimasukkan lagi pada pencarian *augmenting path* karena aliran pada tepi (3, 7) sudah penuh.



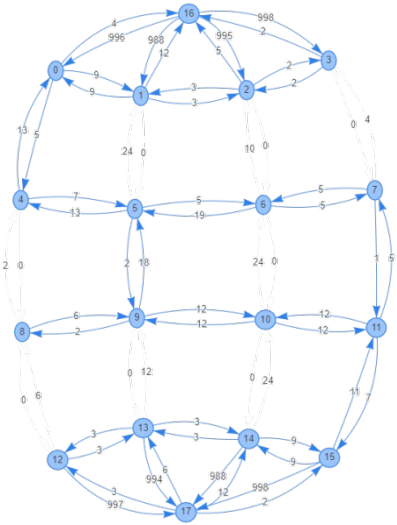
Gambar 2.22 Ilustrasi Graf Residual yang Dihasilkan Setelah Iterasi Kelima

Gambar 2.22menunjukkan aliran residual setelah iterasi kelima pada pencarian aliran maksimal. Pada iterasi ini didapatkan *augmenting path* pada graf, yaitu 16 – 1 – 5 – 6 – 10 – 14 – 17 dengan aliran f_e = 2. Tepi (1, 5) tidak dimasukkan lagi pada pencarian *augmenting path* karena aliran pada tepi (1, 5) sudah penuh.



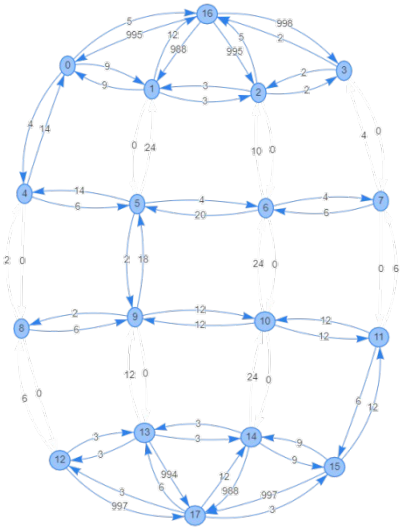
Gambar 2.23 Ilustrasi Graf Residual yang Dihasilkan Setelah Iterasi Keenam

Gambar 2.23menunjukkan aliran residual setelah iterasi keenam pada pencarian aliran maksimal. Pada iterasi ini didapatkan *augmenting path* pada graf, yaitu 16 – 0 – 4 – 5 – 6 – 10 – 14 – 17 dengan aliran f_e = 1. Tepi (6, 10) dan tepi (10, 14) tidak dimasukkan lagi pada pencarian *augmenting path* karena aliran pada tepi (6, 10) dan tepi (10, 14) sudah penuh.



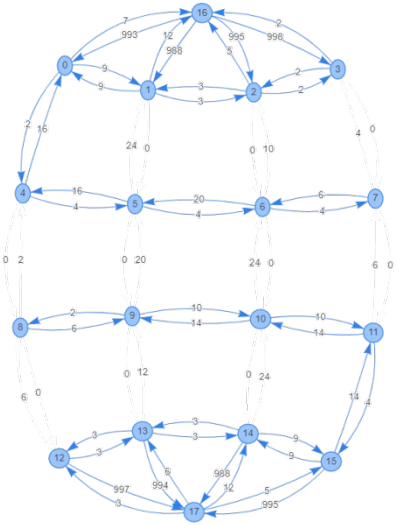
Gambar 2.24 Ilustrasi Graf Residual yang Dihasilkan Setelah Iterasi Ketujuh

Gambar 2.24 menunjukkan aliran residual setelah iterasi ketujuh pada pencarian aliran maksimal. Pada iterasi ini didapatkan *augmenting path* pada graf, yaitu 16 – 0 – 4 – 5 – 9 – 8 – 12 – 17 dengan aliran f_e = 2. Tepi (8, 12) tidak dimasukkan lagi pada pencarian *augmenting path* karena aliran pada tepi (8, 12) sudah penuh.



Gambar 2.25 Ilustrasi Graf Residual yang Dihasilkan Setelah Iterasi Kedelapan

Gambar 2.25 menunjukkan aliran residual setelah iterasi kedelapan pada pencarian aliran maksimal. Pada iterasi ini didapatkan *augmenting path* pada graf, yaitu 16 – 0 – 4 – 5 – 6 – 7 – 11 – 15 – 17 dengan aliran f_e = 1. Tepi (7, 11) tidak dimasukkan lagi pada pencarian *augmenting path* karena aliran pada tepi (7, 11) sudah penuh.



Gambar 2.26 Ilustrasi Graf Residual yang Dihasilkan Setelah Iterasi Kesembilan

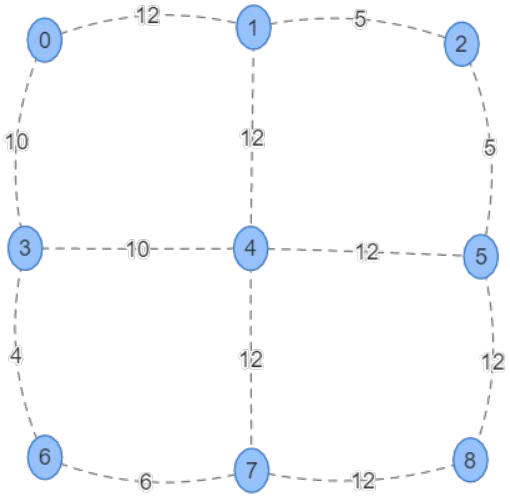
Gambar 2.26 menunjukkan aliran residual setelah iterasi kesembilan pada pencarian aliran maksimal. Pada iterasi ini didapatkan *augmenting path* pada graf, yaitu 16 – 0 – 4 – 5 – 9 – 10 – 11 – 15 – 17 dengan aliran f_e = 2. Tepi (5, 9) tidak dimasukkan lagi pada pencarian *augmenting path* karena aliran pada tepi (5, 9) sudah penuh.

Setelah iterasi kesembilan tidak ditemukan lagi *augmenting path* dari sumber (simpul 16) ke tujuan (simpul 17), maka iterasi dihentikan dan total aliran maksimal dapat dihitung dari jumlah semua f_e yang ditemukan 1 + 6 + 5 + 2 + 6 + 1 + 2 + 1 + 2 = 26.

2.14 Penyelesaian Permasalahan ADAHOSE pada SPOJ dengan Algoritma Dijkstra

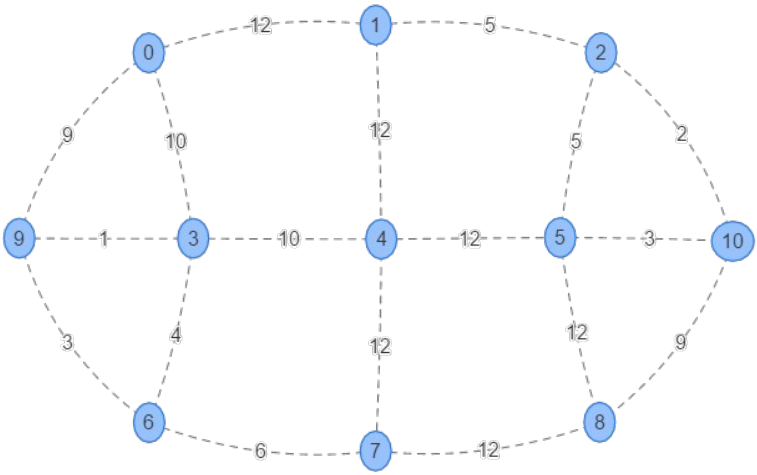
Dari permasalahan ADAHOSE diberikan masukan berupa sebuah bilangan N dan sebuah matriks 2 dimensi A dengan ukuran $N \times N$. Dari matriks tersebut kemudian akan dibuat sebuah graf tak berarah G, dimana simpul-simpulnya merupakan semua

elemen matriks *A* yang bertetangga dengan element di atas, kanan, bawah, dan atau kirinya. Bobot dari tiap tepi-tepi graf *G* adalah jumlah dari elemen pada matriks *A* yang terhubung dengan tepi tersebut seperti pada Gambar 2.27.



Gambar 2.27 Graf *G* Sebelum Ditambah Simpul Terakhir

Kemudian ditambah lagi dua simpul. Simpul pertama bertetangga dengan elemen matriks A di kolom paling kiri dan simpul kedua bertetangga dengan elemen matriks A di kolom paling kanan dengan bobot elemen matriks yang terhubung. Graf *G* yang terbentuk akan menjadi seperti graf pada Gambar 2.28.



Gambar 2.28 Graf *G* Setelah Ditambah Dua Simpul Terakhir

Langkah selanjutnya adalah mencari jalan terpendek dari simpul 9 ke simpul 10 atau dari simpul $N \times N$ ke simpul $N \times N+1$ dengan menggunakan Algoritma Dijkstra. Nilai jalan terpendek tersebut adalah jumlah aliran maksimal dari permasalahan ADAHOSE.

BAB III
ANALISIS DAN PERANCANGAN

Pada bagian ini akan dijelaskan analisis dan desain sistem yang digunakan untuk menyelesaikan permasalahan pada Tugas Akhir ini

3.1 Analisis Penyelesaian Permasalahan ADAHOSE

Pada subbab ini akan dijelaskan analisis penyelesaian permasalahan *ADAHOSE – Ada and Hose* menggunakan beberapa algoritma dan struktur data yang telah dijelaskan pada bab 2.

3.1.1 Perbandingan Algoritma

Dari beberapa algoritma yang telah dijelaskan pada bab 2, akan dicari algoritma dengan kompleksitas terkecil untuk digunakan sebagai penyelesaian permasalahan ADAHOSE.

3.1.1.1 Algoritma Ford-Fulkerson

Penyelesaian permasalahan ADAHOSE dengan algoritma Ford-Fulkerson menggunakan graf yang sama seperti pada Subbab 2.13 yaitu simpul adalah persimpangan selang-selang dan tepi adalah selang-selang.

Kompleksitas dari algoritma Ford-Fulkerson adalah $O(EV^3)$ dengan E adalah jumlah tepi dan V adalah jumlah simpul. Sehingga jika jumlah simpul adalah $(N+1)^2+2$, dan jumlah tepi adalah $4N^2$ dengan N adalah ukuran ladang, maka kompleksitas algoritma Ford-Fulkerson menjadi $O(N^{12})$.

3.1.1.2 Algoritma Dinic

Penyelesaian permasalahan ADAHOSE dengan algoritma Dinic menggunakan graf yang samaseperti pada pemnyelesaian dengan algoritma Ford-Fulkerson.

Kompleksitas dari algoritma Dinic adalah $O(EV^2)$ dengan E adalah jumlah tepi dan V adalah jumlah simpul. Sehingga jika jumlah simpul adalah $(N+1)^2+2$, dan jumlah tepi adalah $2N^2$ dengan N adalah ukuran ladang, maka kompleksitas algoritma Dinic menjadi $O(N^8)$.

Tabel 3.1 Jumlah Simpul dan Tepi pada Graf untuk Algoritma Ford-Fulkerson dan Dinic

N	V	E
3	18	64
4	27	100
8	83	324

3.1.1.3 Algoritma Dijkstra

Penyelesaian permasalahan ADAHOSE dengan algoritma Dijkstra menggunakan graf seperti pada Subbab 2.14 yaitu simpul adalah bagian ladang dan tepi adalah ketetanggaan bagian ladang.

Kompleksitas dari algoritma Dijkstra adalah $O(V + ElogV)$ dengan E adalah jumlah tepi dan V adalah jumlah simpul. Jika jumlah simpul adalah $N^2 + 2$, dan jumlah tepi adalah $2(2N^2 - N)$ dengan N adalah ukuran ladang, maka kompleksitas algoritma Dijkstra menjadi $O(N^2 + 2N^2logN)$

Tabel 3.2 Jumlah Simpul dan Tepi pada Graf untuk Algoritma Dijkstra

N	V	E
3	10	30
4	17	56
8	65	240

Dilihat dari kompleksitas ketiga algoritma, maka algoritma yang digunakan pada Tugas Akhir ini adalah algoritma Dijkstrakarena merupakan metode pencarianjalan terpendekyang cepat dan menghasilkan solusi yang benar.

3.1.2 Struktur Data yang Digunakan

Pada tugas akhir ini digunakan *adjacency list* sebagai representasi grafnya.Adjacency list membutuhkan struktur data yang dapat menyimpan hubungan antara simpul dan tetangganya serta menyimpan ukuran bobot pada tepi yang dihasilkan dari hubungan dua simpul.

Untuk menyimpan hubungan antar simpul dan bobot tepinya, struktur data yang dapat digunakan adalah *array* yang elemennya berisi vector dengan ukuran sebanyak jumlah simpul yang terbentuk.*Vector* tersebut akan diisi dengan *pair of integers* yang elemen pertama dan keduanya bertipe *integer*. *Array* digunakan untuk merepresentasikan simpul-simpul yang terbentuk, *vector* digunakan untuk merepresentasikan tepi yang terbentuk. Elemen pertama dari *pair of integers* adalah nomor simpul yang terhubung ke simpul yang direpresentasikan oleh *index* dari *array* yang berisi *vector* yang menyimpan *pair* tersebut.

3.2 Desain Penyelesaian Permasalahan ADAHOSE

Pada subbab ini dijelaskan mengenai desain penyelesaian permasalahan *ADAHOSE – Ada and Hose*.

3.2.1 Definisi Umum Sistem

Sistemakan menerima masukan berupa satu buah bilangan N yang merepresentasikan ukuran ladang. Kemudian sistem akan menerima masukan sejumlah N baris dengan setiap baris berisi N buah bilangan yang merepresentasikan ukuran selang yang mengelilingi 1 unit persegi ladang dan kemudian akan disimpan ke sebuah matriks berukuran $N \times N$ sesuai dengan tiap unit persegi dari ladang.

```
     main()
1.  N = Input()
2.  for i = 0 toN
3.      for j = 0 toN
4.          A[i][j] = GetNum( <int>)
```

Kode Sumber 3.7 *Pseudocode*Fungsi*Main* ADAHOSE untuk Menerima Masukan

Dari matriks tersebut akan dibentuk sebuah graf berdasarkan tiap unit persegi lading dan tepi adalah hubungan unit ladang dengan tetangga atas, kanan, bawah, dan kirinya. Kemudian ditambah satu simpul terakhir yang merepresentasikan semua selang paling kiri di ladang dan berhubungan dengan semua unit ladang di sisi paling kiri.

```
     main() cont.
1.  for i = N-1 to 0
2.      for j = N-1 to 0
3.      u = i * N + j
4.          ifj> 0
5.          v = i*N+j-1
6.          weight = A[i][j] + A[i][j-1]
7.          graf[u].push(v, weight)
8.          graf[v].push(u, weight)
9.          if i > 0
10.         v = (i-1)*N+j
11.         weight = A[i][j] + A[i-1][j]
12.         graf[u].push(v, weight)
13.         graf[v].push(u, weight)
14. for i = 0 to N
15.     u = i*N
16.     v = N*N
17.     weight = A[i][0]
18.     graf[u].push(v, weight)
19.     graf[v].push(u, weight)
```

Kode Sumber 3.8 Lanjutan *P*seudocode Fungsi Main ADAHOSE untuk Merubah Masukan Menjadi Graf

Selanjutnya sistem akan menjalankan algoritma Dijkstra untuk mencari jalan terpendek dari semua simpul di kolom terakhir ke simpul terakhir dan jarak dari tiap-tiap simpul ke simpul terakhir akan disimpan.

```
main( ) cont.
1.  new priority_queue pq
2.  new dist[v+2]
3.  dist = INF
4.  pq.push(0, v)
5.  dist[v] = 0
6.  while pq != empty
7.      u = pq.pop.second
8.      for x in graf[u]
9.          v = x.first
10.         weight = x.second
11.         if dist[v] > dist[u] + weight
12.         dist[v] = dist[u] + weight
13.         pq.push(dist[v], v)
```

Kode Sumber 3.9 Lanjutan Pseudocode Fungsi Main ADAHOSE untuk menjalankan algoritma Dijkstra

Terakhir adalah mencari nilai terkecil jalan terpendek dari simpul-simpul di kolom paling kanan ke simpul terakhir ditambah ukuran ladang di kolom paling kanan untuk menentukan aliran maksimal dari permasalahan ADAHOSE.

```
main( ) cont.
1.  ret = INF
2.  for i to N
3.      ret = min(ret, dist[i*N+N-1] +
        A[i][N-1]
4.  Output(ret)
```

Kode Sumber 3.10 Lanjutan Pseudocode Fungsi Main ADAHOSE untuk mencari nilai terkecil dari jarak terpendek

3.2.2 Desain Algoritma

Sistem terdiri dari dua fungsi, yaitu fungsi GetNum. Pada subbab ini akan dijelaskan desain algoritma dari kedua fungsi tersebut.

3.1.2.1 Desain Fungsi *GetNum*

Fungsi `GetNum` digunakan untuk mendapatkan nilai atau angka dari satu atau sekumpulan karakter (*char*) yang akan dikonversikan menjadi nilai bilangan (*integer*). Fungsi `GetNum` merupakan fungsi yang mengimplementasikan algoritma *Fast I/ O*. *Pseudocode* Fungsi `GetNum` ditunjukkan pada Kode Sumber 3.5.

```
GetNum ( )
1.  res = 0
2.  while true
3.      c = Input( )
4.      if c = ' '      || c = ' \n '
5.          continue
6.      else
7.          break
8.  res = c - '0'
9.  while true
10.  c = Input()
11.     if (c ≥ ' 0 ' & c ≤ ' 9 ')
12.         res= 10 * res + c - ' 0 '
13.     else
14.         break
15. return res
```

Kode Sumber 3.11*Pseudocode* Fungsi *GetNum* ADAHOSE

BAB IV
IMPLEMENTASI

Pada bab ini akan dijelaskan tentang implementasi yang dilakukan berdasarkan algoritma yang telah dirancang pada Bab 3 di atas.

4.1 Lingkungan Implementasi

Lingkungan implementasi menggunakan sebuah komputer dengan spesifikasi perangkat lunak dan perangkat keras seperti tercantum pada Tabel 4.1 di bawah ini.

Tabel 4.3 Spesifikasi Lingkungan Implementasi

No.	Jenis Perangkat	Spesifikasi
1	Perangkat Keras	• *Processor* Intel Core i7-7500U CPU @ 2.70GHz • *Memory* 8GB 2133MHz DDR4
2	Perangkat Lunak	• Sistem operasi Windows 10 Pro • *Integrated Development Environment Code:Block 17.12*

4.2 Implementasi Penyelesaian Permasalahan ADAHOSE

Pada subbab ini dijelaskan mengenai implementasi dari penyelesaian permasalahan ADAHOSE sesuai desain yang dibuat pada Subbab 3.1.

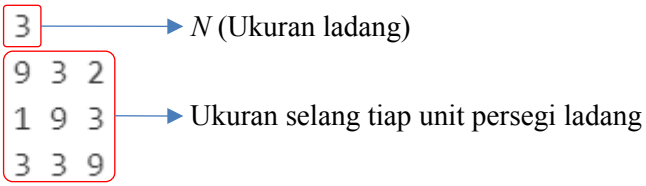
4.2.1 Rancangan Data

Pada subbab ini dijelaskan mengenai desain data masukan yang diperlukan untuk melakukan proses algoritma serta data keluaran yang dihasilkan oleh program.

4.2.1.1 Data Masukan

Data masukan adalah data yang akan diproses oleh program sebagai masukan dengan menggunakan algoritma dan struktur data yang telah dirancang dalam Tugas Akhir ini. Data masukan dari persoalan ini berupa suatu bilangan *N* yang merepresentasikan ukuran ladang. Nilai *N* memiliki rentang mulai dari 1 sampai dengan 1000 ($1 \leq N \leq 1000$). Mengikuti jumlah *N*, masukan berikutnya berupa bilangan-bilangan yang merepresentasikan ukuran selang di tiap unit persegi ladang.

Berikut contoh data masukan pada permasalahan ini (Gambar 4.1)



Gambar 4.29 Format Masukan Permasalahan ADAHOSE

Dari contoh data masukan (Gambar 4.1), angka 3 pada baris pertama merepresentasikan ukuran ladang.Kemudian masukan pada baris-baris berikutnya merepresentasikan ukuran selang. Sebagai contoh, pada baris kedua angka 9 merupakan ukuran selang yang mengelilingi ladang di baris pertama dan kolom pertama, angka 3 merupakan ukuran selang yang mengelilingi ladang di baris pertama dan kolom kedua. Dan seterusnya.

4.2.1.2 Data keluaran

Data keluaran yang dihasilkan oleh program berupa nilai aliran maksimal atau jumlah maksimal aliran yang dapat keluar di alat penyiram setelah melewati selang-selang dengan batasan ukuran selang.

4.2.2 Penggunaan *Library*, Konstanta, dan Variabel Global

Pada subbab ini dijelaskan mengenai *library*, *template* konstanta, dan variabel global yang digunakan dalam sistem. Pada Kode Sumber 4.1, terdapat 1 *library* yang digunakan yaitu *stdc++*. *Library* tersebut berisi semua modul yang terdapat di *standard library*. Didefinisikan konstanta INF yang bernilai 0x3f3f3f3f atau nilai maksimal yang dapat ditampung oleh *4-byte signed int* daniPair yang didefinisikan sebagai kependekan dari *pair<int, int>* atau sebuah *pair* yang tipe data element pertama dan keduanya adalah *int*.

```
1.    #include<bits/stdc++.h>
2.    # define INF 0x3f3f3f3f
3.    typedef pair<int, int> iPair;
4.    int N,V;
5.    vector<iPair > *adj;
```

Kode Sumber 4.12 Potongan Kode Implementasi *library*, Konstanta, dan Variabel Global

Pada Tabel 4.2akan dijelaskan mengenai variabel-variabel global yang akan digunakan dalam implementasi program penyelesaian permasalahan ADAHOSE.

Tabel 4.4Tabel Daftar Variabel Global Permasalahan ADAHOSE

No	Nama Variabel	Tipe	Penjelasam
1	*N*	*int*	Digunakan untuk menyimpan ukuran ladang
2	*V*	*int*	Digunakan untuk menyimpan jumlah simpul yang dibutuhkan graf
3	*adj*	*int* *	Digunakan sebagai pointer yang menunjuk ke array

			yang berisi ketetanggaan graf

4.2.3 Implementasi Fungsi *Main*

Fungsi *Main* diimplementasikan sesuai *pseudocode* pada subbab 3.1.1.Kode sumber fungsi *Main* ditunjukkan pada Kode Sumber 4.2. Pada baris keenam, dibuat *adjacency list* baru dengan ukuran *V+1* dimana nilai *V* adalah *N*N*. Pada baris kesepuluh fungsi *Main*akan memanggil fungsi `GetNum` untuk mendapatkan nilai masukan.

```
1.     int main()
2.     {
3.         N = getNum<int>();
4.         int A[N][N];
5.         V = (N*N);
6.         adj = new vector<iPair>[V+1];
7.
8.         for(int i=0;i<N;i++) {
9.             for(int j=0;j<N;j++) {
10.                A[i][j] = getNum<int>();
11.            }
12.        }
13.
14.        for (int i = N-1; i >= 0; i--)
15.        {
16.            for (int j = N-1; j >= 0; j--)
17.            {
18.                int u = i*N+j;
19.                if (j > 0)
20.                {
21.                    int v = i*N+j-1, wt = A[i][j] +
   A[i][j-1];
22.                    adj[u].push_back(make_pair(v, w
   t));
23.                    adj[v].push_back(make_pair(u, w
   t));
24.
25.                }
26.
27.                if (i > 0)
```

```
                {
                    int v = (i-1)*N+j, wt = A[i][j]
    + A[i-1][j];
```

Kode Sumber 4.13 Potongan Kode Implementasi Fungsi Main Permasalahan Adahose (Bagian 1)

```
                    adj[u].push_back(make_pair(v, w
    t));
                    adj[v].push_back(make_pair(u, w
    t));
                }

            }
        }

        for (int i = 0; i < N; i++)
        {
            int u = i*N, v = V, wt = A[i][0];
            adj[u].push_back(make_pair(v, wt));
            adj[v].push_back(make_pair(u, wt));
        }


        priority_queue< iPair, vector <iPair> , pai
    r_compare > pq;
        vector<int> dist(1000000, INF);

        pq.push(make_pair(0, V));
        dist[V] = 0;

        while (!pq.empty())
        {
            int u = pq.top().second;
            pq.pop();

            for (auto x : adj[u])
            {
                int v = x.first;
                int weight = x.second;

                if (dist[v] > dist[u] + weight)
                {
```

```
64.                     dist[v] = dist[u] + weight;
65.                     pq.push(make_pair(dist[v], v));

66.                 }
67.             }
68.         }
```

Kode Sumber 4.3 Potongan Kode Implementasi Fungsi Main Permasalahan Adahose (Bagian 2)

```
69.         int ret = INF;
70.         for (int i = 0; i < N; i++) {
71.             ret = min(ret, dist[i * N + N - 1] + A[
    i][N - 1]);
72.         }
73.         printf("%d\n", ret);
74.
75.         return 0;
76.     }
```

Kode Sumber 4.4 Potongan Kode Implementasi Fungsi Main Permasalahan Adahose (Bagian 3)

4.2.4 Implementasi Funsi GetNum

Fungsi GetNum diimplementasikan sesuai pseudocode pada Subbab 3.1.2.1. Fungsi ini digunakan untuk mendapatkan nilai dari satu atau sekumpulan karakter yang nantinya akan dikonversikan menjadi suatu nilai bilangan. Kode sumber fungsi GetNum ditunjukkan pada Kode Sumber 4.3.

```
1.      template<typename T>
2.      T getNum()
3.      {
4.          T res=0;
5.          char c;
6.          while(1) {
7.              c=getchar();
8.              if(c==' ' || c=='\n') continue;
9.              else break;
10.         }
11.         res=c-'0';
12.         while(1) {
13.             c=getchar();
```

```
14.             if(c>='0' && c<='9') res=10*res+c-'0';
15.             else break;
16.         }
17.         return res;
18.     }
```

Kode Sumber 4.5 Potongan Kode Implementasi Fungsi GetNum Permasalahan ADAHOSE

4.2.5 Implementasi *Structpair_compare*

Fungsi GetNum diimplementasikan sesuai pseudocode pada Subbab 3.1.2.1. Fungsi ini digunakan untuk mendapatkan nilai dari satu atau sekumpulan karakter yang nantinya akan dikonversikan menjadi suatu nilai bilangan. Kode sumber fungsi GetNum ditunjukkan pada Kode Sumber 4.3.

```
1.     struct pair_compare {
2.         bool operator()(const iPair &a, const iPair
   &b){
3.             return a.first > b.first;
4.         };
5.     };
```

Kode Sumber 4.6 Potongan Kode Implementasi *Struct pair_compare*Permasalahan ADAHOSE

[Halaman ini sengaja dikosongkan]

BAB V
UJI COBA DAN EVALUASI

Pada bab ini akan dijelaskan tentang uji coba da evaluasi dari implementasi sistem yang telah dilakukan pada bab sebelumnya, yaitu Bab 4.

5.1 Lingkungan UjiCoba

Lingkungan ujicoba menggunakan sebuah komputer dengan spesifikasi perangkat lunak dan perangkat keras seperti tercantum pada Tabel 5.1 di bawah ini.

Tabel 5.5 Spesifikasi Lingkungan Ujicoba

No.	Jenis Perangkat	Spesifikasi
1	Perangkat Keras	• *Processor* Intel Core i7-7500U CPU @ 2.70GHz • *Memory* 8GB 2133MHz DDR4
2	Perangkat Lunak	• Sistem operasi Windows 10 Pro • *Integrated Development Environment Code:Block 17.12*

5.2 Skenario UjiCoba

Pada subbab ini akan dijelaskan skenario ujicoba yang dilakukan. Skenario ujicoba terdiri dari ujicoba kebenaran dan ujicoba kinerja.

5.2.1 UjiCoba Kebenaran

Ujicoba kebenaran dilakukan dengan mengirimkan kode sumber kedalam situs penilaian online SPOJ.Hasil ujicoba dengan waktu terbaik pada situs penilaian daring SPOJ ditunjukan pada Gambar 5.1.

Dari hasil ujicoba kebenaran yang dilakukan pada situs penilaian online SPOJ seperti pada Gambar 5.1, dapat dilihat bahwa kode sumber mendapat keluaran Accepted.Waktu tercepat yang dibutuhkan oleh program adalah 0.00 detik dengan memori yang dibutuhkan sebesar 19 M.

Gambar 5.30 Hasil ujicoba pada Situs Penilaian Online *SPOJ* Permasalahan Adahose

Selanjutnya akan dibandingkan antara hasil ujicoba dengan kasus sederhana dan hasil ujicoba dengan keluaran sistem. Ujicoba kasus sederhana akan diselesaikan dengan langkah-langkah penyelesaian pada Subbab 2.13.

Kasus sederhana yang akan digunakan adalah sebuah kasus dimana Ada memiliki ladang dengan ukuran 3. Ukuran selang yang mengelilingi ladang tersebut dimisalkan pada format masukan berikut (Gambar 5.2) dalam bentuk matriks 2 dimensi.

```
1.  3
2.  9 3 2
3.  1 9 3
4.  3 3 9
```

Gambar 5.31 Format Masukan Uji Kebenaran ADAHOSE

Pada Gambar 5.2 dijelaskan, angka 3 pada baris pertama merupakan ukuran ladang. Kemudian pada baris-baris berikutnya akan diberi masukan berupa bilangan yang merupakan ukuran selang yang mengelilingi ladang di baris dan kolom tersebut. Penjelasan ukuran selang bisa dilihat pada Tabel 5.2 dan Tabel 5.3. Matriks tersebut kemudian akan diubah menjadi sebuah graf tidak berarah dan berbobot dan ditambah simpul terakhir yang bertetangga dengan simpul-simpul yang terbentuk dari unit ladang di kolom paling kiri (Gambar 5.3). Selanjutnya untuk mencari aliran yang bisa keluar di alat penyiram setelah melewati

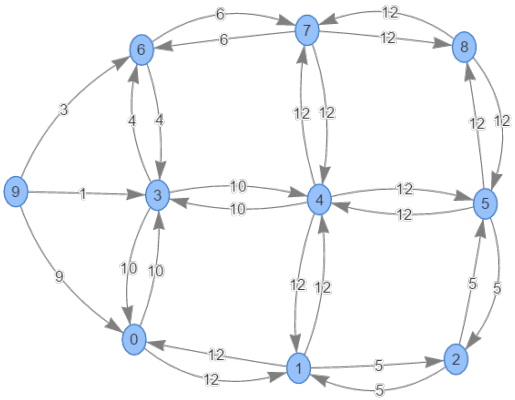
selang-selang di ladang maka perlu mencari aliran maksimal dari graf yang telah dimodelkan

Tabel 5.6 Penjelasan Ukuran Selang yang Mengelilingi Tiap Unit

Unit	Ukuran selang
0	9
1	3
2	2
3	1
4	9
5	3
6	3
7	3
8	9

Tabel 5.7 Penjelasan Ukuran Selang yang Terbentuk

Unit	Bertetangga dengan	Ukuran selang yang terbentuk (unit + tetangga)
0	1	12
	3	10
1	0	12
	2	5
	4	12
2	1	5
	5	5
3	0	10
	4	10
	6	4
4	1	12
	3	10
	5	12
	7	12
5	2	5
	4	12
	8	12
6	3	4
	7	6
7	4	12
	6	6
	8	12
8	5	12
	7	12
9	0	9
	3	1
	6	3



Gambar 5.32 Representasi Ukuran Selang dan Hubungan antar Unit dalam Bentuk Graf

5.2.2 UjiCoba Kebenaran Penyelesaian Permasalahan ADAHOSE

Ujicoba kebenaran dilakukan dengan mengirimkan kode sumber ke dalam situs penilaian *online* SPOJ dan melakukan perbandingan hasil ujicoba kasus sederhana pada langkah-langkah yang terdapat pada Subbab 2.14 dengan keluaran sistem. Permasalahan yang diselesaikan adalah *Ada and Hose* dengan kode soal ADAHOSE seperti yang dijelaskkan pada Subbab 2.14. Hasil ujicoba dengan waktu terbaik pada situs penilaian *online* SPOJ ditunjukan pada Gambar 5.1.

Gambar 5.33 Format Masukan Uji Kebenaran ADAHOSE

Dari hasil ujicoba kebenaran yang dilakukan pada situs penilaian *online* SPOJ (Gambar 5.4), dapat dilihat bahwa kode sumber mendapat keluaran *Accepted.* Waktu yang dibutuhkan oleh program adalah 3.98 detik dengan memori yang dibutuhkan sebesar 80 M.

Selain itu dilakukan pengujian sebanyak 10 kali pada situs penilaian *online* SPOJ untuk melihat variasi waktu dan variasi memori yang dibutuhkan program. Hasil ujicoba sebanyak 10 kali dapat dilihat pada Tabel 5. 4.

Dari hasil ujicoba sebanyak 10 kali, seluruh kode sumber program mendapat keluaran *Accepted* dengan waktu 0.00 dan memori yang dibutuhkan program sebesar 23 MB.

Tabel 5.8 Penjelasan Ukuran Selang yang Terbentuk

ID	RESULT	TIME	MEMORY
25149933	**accepted**	4.12	80M
25149939	**accepted**	4.08	80M
25149945	**accepted**	4.06	80M
25149957	**accepted**	4.08	80M
25149964	**accepted**	3.99	80M
25149972	**accepted**	4.07	80M
25149985	**accepted**	3.97	80M
25149988	**accepted**	4.08	80M
25149989	**accepted**	4.08	80M
25149993	**accepted**	3.94	80M

Selanjutnya akan dibandingkan hasil ujicoba kasus sederhana dengan keluaran sistem. Ujicoba dengan kasus sederhana akan diselesaikan sesuai dengan langkah penyelesaian pada Subbab 2.14. Kasus sederhana yang digunakan melibatkan ladang dengan ukuran 10 dan ukuran selang acak antara 0 sampai 1000 sesuai dengan Gambar

```
1.  3
2.  847 41 273
3.  940 249 429
4.  305 157 157
```

Gambar 5.34 Format Masukan Uji Kebenaran ADAHOSE

Kemudian dari format masukan tersebut akan diubah dalam bentuk graf tidak berarah dan berbobot seperti Gambar 5.3

untuk mencari total aliran per waktu dari sumur ke alat penyiram maka perlu mencari jumlah aliran maksimal pada graf yang telah dimodelkan yang dalam TA ini akan digunakan algoritma jalan terpendek *Dijkstra*.

Langkah-langkah pencarian jalan terpendek dengan algoritma Dijkstra dalam permasalahan seperti format masukan pada Gambar 5.5 adalah sebagai berikut:

5.2.3 Ujicoba Kinerja

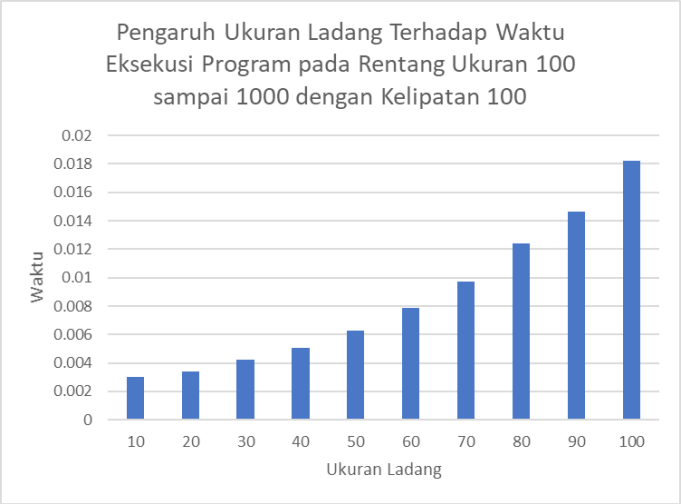
Kompleksitas waktu pencarian potongan minimal dengan menggunakan algoritma Dijkstra adalah O(V $logE$). Dengan representasi graf diatas banyaknya simpul V yang mungkin terbentuk maksimal adalah $2N^2 \times (N - 1) \times 2$ dimana N adalah ukuran ladang dan banyaknya sisi E.

Uji kinerja yang dilakukan adalah dengan menambahkan *librarystd::chrono* pada program yang telah diimplementasikan pada Bab 4. Kemudian pada bab ini dilakukan ujicoba dengan nilai ukuran ladang yang bervariasi dengan ukuran selang acak. Implementasi acak pada ukuran selang akan menggunakan library *std::rand*.

Pada ujicoba ini ukuran ladang dibuat bervariasi mulai dari 10 sampai 100 pada kelipatan 10 dan 100 sampai 1000 dengan kelipatan 100. dicatat waktu ekesekusi program dimana tiap data dalam satuan detik, sehingga ukuran ladang terhadap waktu eksekusi dapat diamati. Hasil uji kinerja tersaji pada Tabel 5.5 dan Tabel 5.6 serta dalam grafik tersaji pada Gambar 5.6 dan Gambar 5.7.

Tabel 5.9 Hasil Ujicoba Kinerja

Percobaan	Ukuran Ladang	Waktu
1	10	0.00301235
2	20	0.00338441
3	30	0.00424731
4	40	0.0050602
5	50	0.00628612
6	60	0.00786521
7	70	0.00970917
8	80	0.0123869
9	90	0.0146122
10	100	0.018201
11	200	0.062519
12	300	0.145115
13	400	0.265934
14	500	0.402648
15	600	0.585125
16	700	0.804876
17	800	1.09487
18	900	1.39798
19	1000	1.74431



Gambar 5.35 Grafik Pengaruh Ukuran Ladang Terhadap Waktu Eksekusi Program pada Rentang Ukuran 10 sampai 100 dengan Kelipatan 10

Gambar 5.36 Grafik Pengaruh Ukuran Ladang Terhadap Waktu Eksekusi Program pada Rentang Ukuran 100 sampai 1000 dengan Kelipatan 100

5.3 Analisis dan Kesimpulan Umum

Dari serangkaian hasi ujicoba didapatkan:

1. Algoritma Dijkstra yang telah didesain dan dijalankan pada permasalahan ADAHOSE dapat menyelesaikan permasalahan pencarian nilai aliran maksimal pada graf
2. Algoritma Dijkstra yang telah didesain dan dijalankan terbukti benar dalam menyelesaikan permasalahan ADAHOSE.
3. Ukuran ladang berpengaruh terhadap program yang diimplementasikan sesuai dengan kompleksitas algoritma Dijkstra.

BAB VI
KESIMPULAN

Pada bab ini akan dijelaskan kesimpulan dari hasil ujicoba yang telah dilakukan.

6.1 Kesimpulan

Dari hasil ujicoba yang telah dilakukan terhadap implementasi solusi untuk permasalahan *ADAHOSE – Ada and Hose* dalam menemukan jumlah potongan minimal, dapat diambil kesimpulan sebagai berikut:

1. Menggunakan Algoritma Dijkstra sebagi pilihan terbaik karena merupakan metode pencarianjalan terpendekyang cepat dan menghasilkan solusi yang benar.
2. Permasalahan mencari aliran maksimal pada graf tidak berarah dan berbobot dari rentang yang telah diberikan telah berhasil diselesaikan menggunakan algoritma Dijkstra yang memiliki kompleksitas $O(N^2)$ dengan N adalah ukuran ladang.
3. Implementasi algoritma Dijskstra pada permasalahan *ADAHOSE -Ada and Hose* menghasilkan solusi yang benar.
4. Waktu rata-rata yang diperlukan untuk menyelesaikan permasalahan ADAHOSE – Ada and Hose dari 10 kali ujicoba pada online judge SPOJ adalah 4.047 detik.

6.2 Saran

Saran yang diberikan untuk pengembangan sistem penyelesaian masalah pada ADAHOSE – *Ada and Hose*dapat dilihat di bawah ini.

1. Dalam pengembangan algoritma pencarian nilai aliran maksimal pada penelitian selanjutnya adalah mencari algoritma yang bisa mencari aliran maksimal pada graf yang lebih umum lagi, bukan hanya pada graf seperti pada permasalahan ADAHOSE – Ada and Hose

2. Bisa dijadikan bahan rekomendasi lebih lanjut sebagai bahan penelitian baik dalam dunia pendidikan maupun ilmu pengetahuan serta teknologi informasi.

DAFTAR PUSTAKA

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms Third Edition.* Cambridge: Massachusetts Institute of Technology.

[2] Gupta, S. (n.d.). *Maximum flow Tutorials & Notes | Algorithms | HackerEarth*. Retrieved 8 18, 2019, from HackerEarth: https://www.hackerearth.com/practice/algorithms/graphs/maximum-flow/tutorial/

[3] Jain, S., Goel, S., Singh, D., & Baranwal, S. (n.d.). *Graph and Its Representations*. Retrieved 12 12, 2019, from GeeksforGeeks: https://www.geeksforgeeks.org/graph-and-its-representations/

[4] Morass. (2017, 10 16). *ADAHOSE - Ada and Hose*. Retrieved 8 18, 2019, from SPOJ: https://www.spoj.com/problems/ADAHOSE/

[5] Roughgarden, T. (2016, January 5). *CS261: A Second Course in Algorithms Lecture #1: Course Goals and Introduction to Maximum Flow.* Retrieved 12 12, 2019, from http://theory.stanford.edu/~tim/w16/l/l1.pdf

[6] Shanghai Jiao Tong University. (n.d.). *Stoer-Wagner Algorithm - A Minimum Cut Algorithm for Undirected Graphs.* Retrieved 12 12, 2019, from Basics Lab: https://basics.sjtu.edu.cn/~dominik/teaching/2016-cs214/presentation-slides/2016-12-06-StoerWagner-BigNews.pdf

Steples-Moore, A. (n.d.). Network Flow and the Max-Flow Min-Cut Theorem. 3

[7]

LAMPIRAN A
KODE SUMBER PROGRAM

Berkut merupakan lampiran kode sumber dari program yang dihasilkan Tugas Akhir ini.

```
1.  #include<bits/stdc++.h>
2.  using namespace std;
3.  # define INF 0x3f3f3f3f
4.
5.  typedef pair<int, int> iPair;
6.  struct pair_compare {
7.      bool operator()(const iPair &a, const iPair
    &b){
8.          return a.first > b.first;
9.      };
10. };
11.
12. int N,V;
13. vector<iPair > *adj;
14.
15. template<typename T>
16. T getNum()
17. {
18.     T res=0;
19.     char c;
20.     while(1) {
21.         c=getchar_unlocked();
22.         if(c==' ' || c=='\n') continue;
23.         else break;
24.     }
25.     res=c-'0';
26.     while(1) {
27.         c=getchar_unlocked();
28.         if(c>='0' && c<='9') res=10*res+c-'0';

29.         else break;
30.     }
31.     return res;
32. }
```

Kode Sumber A.1 Kode Sumber Program (Bagian 1)

```
33. int main()
34. {
35.     N = getNum<int>();
36.     int A[N][N];
37.     V = (N*N);
38.     adj = new vector<iPair>[V+1];
39.
40.     for(int i=0;i<N;i++) {
41.         for(int j=0;j<N;j++) {
42.             A[i][j] = getNum<int>();
43.         }
44.     }
45.     for (int i = N-1; i >= 0; i--)
46.     {
47.         for (int j = N-1; j >= 0; j--)
48.         {
49.             int u = i*N+j;
50.             if (j > 0)
51.             {
52.                 int v = i*N+j-1, wt = A[i][j] +
    A[i][j-1];
53.                 adj[u].push_back(make_pair(v, w
    t));
54.                 adj[v].push_back(make_pair(u, w
    t));
55.             }
56.             if (i > 0)
57.             {
58.                 int v = ( i – 1 ) * N + j,
    wt = A[i][j] + A[i-1][j];
59.                 adj[u].push_back(make_pair(v, w
    t));
60.                 adj[v].push_back(make_pair(u, w
    t));
61.             }
62.         }
63.     }
64.     for (int i = 0; i < N; i++)
65.     {
66.         int u = i*N, v = V, wt = A[i][0];
67.         adj[u].push_back(make_pair(v, wt));
```

```
68.         adj[v].push_back(make_pair(u, wt));
69.     }
```

Kode Sumber A.2 Kode Sumber Program (Bagian 2)

```
70.     priority_queue< iPair, vector <iPair> , pai
    r_compare > pq;
71.     vector<int> dist(1000000, INF);
72.
73.     pq.push(make_pair(0, V));
74.     dist[V] = 0;
75.
76.     while (!pq.empty())
77.     {
78.         int u = pq.top().second;
79.         pq.pop();
80.
81.         for (auto x : adj[u])
82.         {
83.             int v = x.first;
84.             int weight = x.second;
85.
86.             if (dist[v] > dist[u] + weight)
87.             {
88.                 dist[v] = dist[u] + weight;
89.                 pq.push(make_pair(dist[v], v));

90.             }
91.         }
92.     }
93.
94.     int ret = INF;
95.     for (int i = 0; i < N; i++) {
96.         ret = min(ret, dist[i * N + N - 1] + A[
    i][N - 1]);
97.     }
98.     printf("%d\n", ret);
99.
100.          return 0;
101.      }
```

Kode Sumber A.3 Kode Sumber Program (Bagian 3)

[Halaman ini sengaja dikosongkan]

BIODATA PENULIS

Muhammad Pandu Praadha, lahir di Palembang tanggal 16 April 1997. Penulis merupakan anak pertama dari 2 bersaudara. Penulis telah menempuh Pendidikan formal TK YKAI Palembang, SD Islam Az-Zahrah Palembang (2003-2009), SMP Negeri 1 Palembang (2009-2012), SMA Negeri 17 Palembang (2012-2015). Penulis melanjutkan studi kuliah program sarjana di Departemen Informatika ITS.Selama kuliah di Informatika ITS, penulis mengambil bidang minat Algoritma Pemrograman (AP). Selama menempuh perkuliahan, penulis pernah mengikuti organisasi mahasiswa sebagai Kepala Departemen *Public Relation* di ITS Billiard. Penulis juga aktif dalam kegiatan kepanitian Schematics sebagai staff REEVA pada tahun 2016 dan menjadi staff ahli REEVA pada tahun 2017. Penulis dapat dihubungi melalui surel di `mpandu.praadha@gmail.com`