TUGAS AKHIR - KI141502

DESAIN DAN ANALISIS ALGORITMA BREADTH FIRST SEARCH SEBAGAI METODE PENYELESAIAN PERMA-SALAHAN PENCARIAN CYCLE GRAF: STUDI KASUS SPOJ 28409 - ADA AND CYCLE

CHRISTYANTO LIMAN
NRP 05111440000087

Dosen Pembimbing 1
Rully Soelaiman, S.Kom.,M.Kom.

Dosen Pembimbing 2
Rizky Januar Akbar, S.Kom., M.Eng.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

[*Halaman ini sengaja dikosongkan*]

TUGAS AKHIR - KI141502

DESAIN DAN ANALISIS ALGORITMA BREADTH FIRST SEARCH SEBAGAI METODE PENYELESAIAN PERMASALAHAN PENCARIAN CYCLE GRAF: STUDI KASUS SPOJ 28409 - ADA AND CYCLE

CHRISTYANTO LIMAN
NRP 05111440000087

Dosen Pembimbing 1
Rully Soelaiman, S.Kom.,M.Kom.

Dosen Pembimbing 2
Rizky Januar Akbar, S.Kom., M.Eng.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

[*Halaman ini sengaja dikosongkan*]

UNDERGRADUATE THESES - KI141502

DESIGN AND ANALYSIS OF BREADTH FIRST SEARCH ALGORITHMS TO SOLVE FINDING CYCLE GRAPH PROBLEM CASE STUDY SPOJ 28409 - ADA AND CYCLE

CHRISTYANTO LIMAN
NRP 05111440000087

Supervisor 1
Rully Soelaiman, S.Kom.,M.Kom.

Supervisor 2
Rizky Januar Akbar, S.Kom., M.Eng.

INFORMATICS DEPARTMENT
Faculty of Information and Communication Technology
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

[*Halaman ini sengaja dikosongkan*]

LEMBAR PENGESAHAN

DESAIN DAN ANALISIS ALGORITMA BREADTH FIRST SEARCH SEBAGAI METODE PENYELESAIAN PERMASALAHAN PENCARIAN CYCLE GRAF: STUDI KASUS SPOJ 28409 - ADA AND CYCLE

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Algoritma Pemrograman
Program Studi S-1 Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh:
Christyanto Liman
NRP. 05111440000087

Disetujui oleh Dosen Pembimbing Tugas Akhir:

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

Rizky Januar Akbar, S.Kom., M.Eng. .....
NIP. 198701032014041001 (Pembimbing 2)

SURABAYA
JULI 2018

[*Halaman ini sengaja dikosongkan*]

ABSTRAK

DESAIN DAN ANALISIS ALGORITMA BREADTH FIRST SEARCH SEBAGAI METODE PENYELESAIAN PERMASA-LAHAN PENCARIAN CYCLE GRAF: STUDI KASUS SPOJ 28409 - ADA AND CYCLE

Nama	:	Christyanto Liman
NRP	:	05111440000087
Departemen	:	Departemen Informatika, Fakultas Teknologi Informasi dan Komunikasi, ITS
Pembimbing I	:	Rully Soelaiman, S.Kom.,M.Kom.
Pembimbing II	:	Rizky Januar Akbar, S.Kom., M.Eng.

Abstrak

Permasalahan SPOJ "Ada and Cycle" mengangkat permasalahan perhitungan jarak terpendek yang berawal dari sebuah ver-tex dan berakhir pada vertex yang sama untuk setiap vertex yang ada. Pada permasalahan ini diberikan parameter masukan berupa jumlah vertex (N) dan sebuah adjacency matrix (H). Tujuan Tugas akhir ini adalah menyusun strategi penyelesaian, melakukan ana-lisis dan desain algoritma dan mengevaluasi hasil dari kinerja al-goritma penyelesaian permasalahan "Ada and Cycle" pada situs SPOJ.

Metode yang dilakukan adalah pengoptimasian metode Breadth First Search agar memenuhi batasan waktu yang diberikan soal. Pengoptimasian yang dilakukan dibagi menjadi dua, yaitu pengoptimasian masukan dan pengoptimasian algoritma Breadth First Search. Dalam pengoptimasian masukan dibuat fungsi untuk mempercepat pemrosesan masukan. Dalam pengoptimasian algo-

ritma Breadth First Search, dilakukan pengurangan vertex yang ti-dak perlu dikunjungi sehingga sisi atau edge yang perlu dilewati berkurang.

Melalui pengujian dan studi kasus, didapat hasil bahwa me-tode Breadth First Search yang telah dioptimasi dapat digunakan untuk menyelesaikan kasus SPOJ "Ada and Cycle". Umpan balik kinerja yang didapat dari algoritma Breadth First Search yang te-lah dioptimasi pada situs SPOJ adalah 1,76 detik dan penggunaan memori 2,9M dengan kompleksitas O(N^2).

Kata Kunci: Breadth First Search; Cycle; Edge; Graf; In Degree; Jarak Terpendek; Out Degree; Vertex;

ABSTRACT

DESIGN AND ANALYSIS OF BREADTH FIRST SEARCH AL-GORITHMS TO SOLVE FINDING CYCLE GRAPH PROBLEM CASE STUDY SPOJ 28409 - ADA AND CYCLE

Name	:	Christyanto Liman
Student ID	:	05111440000087
Department	:	Informatics Department, Faculty of Information and Communication Technology, ITS
Supervisor I	:	Rully Soelaiman, S.Kom.,M.Kom.
Supervisor II	:	Rizky Januar Akbar, S.Kom., M.Eng.

Abstract

SPOJ "Ada and Cycle" problem brings the problem of calculating the shortest path that starts from a vertex and ends in the same vertex for every available verticles. Input given from the pro-blem are number of verticles (N) and adjacency matrix (H). The ob-jectives of this thesis are compose a strategy to solve the problem, analyze and designing the algorithm, and evaluate the result of al-gorithm's performance for solving the "Ada and Cycle" problem in SPOJ.

The method used is optimizing Breadth First Search method so it meet the time requirement given by problem. The optimization consist of two part, input optimazion and Breadth First Search algorithm optimization. In the input optimization a function is made to accelerate the input process. In the Breadth First Search algori-thm optimization done by eliminating verticles that can be skipped so the edge that need to be passed is reduced.

According to subsequent testing and case study, it appears

that optimized Breadth First Search method can be used to solve SPOJ "Ada and Cycle" problem. The performance feedback from optimized Breadth First Search method that is submitted to SPOJ is 1.76 seconds and 2.9M memory used with complexcity of O(N^2).

Keywords: Breadth First Search; Cycle; Edge; Graph; In Degree; Out Degree; Shortest Path; Vertex;

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa. Atas rahmat dan kasih sayangNya, penulis dapat menyelesaikan tugas akhir dan laporan akhir dalam bentuk buku ini.

Pengerjaan buku ini penulis tujukan untuk mengeksplorasi le-bih mendalam topik-topik yang tidak diwadahi oleh kampus, namun banyak menarik perhatian penulis. Selain itu besar harapan penulis bahwa pengerjaan tugas akhir sekaligus pengerjaan buku ini dapat menjadi batu loncatan penulis dalam menimba ilmu yang berman-faat.

Penulis ingin menyampaikan rasa terima kasih kepada banyak pihak yang telah membimbing, menemani dan membantu pe-nulis selama masa pengerjaan tugas akhir maupun masa studi.

1. Ibu Titik Supriyatin, selaku ibu penulis yang senantiasa mendukung dan mendoakan penulis dalam pengerjaan tugas akhir.
2. Bapak Suwarli Liman, selaku ayah penulis yang selalu mendukung dan mendoakan penulis dalam pengerjaan tugas akhir.
3. Bapak Rully Soelaiman S.Kom.,M.Kom., selaku pembimbing penulis. Berkat bimbingan beliau, tugas akhir ini dapat diselesaikan dengan baik. Ucapan terima kasih juga penulis sampaikan atas segala perhatian, didikan, pengajaran, dan nasihat yang telah diberikan oleh beliau selama masa studi penulis.
4. Bapak Rizky Januar Akbar, S.Kom., M.Eng., selaku pembim-bing penulis yang telah memberikan arahan semasa penger-jaan tugas akhir.
5. Kinanti Sekarayu, selaku teman yang selalu memberi semangat dan mengingatkan untuk segera menyelesaikan tugas akhir sehingga penulis terpacu untuk segera menyelesaikan tugas akhir.
6. Michael Dave, selaku teman seperjuangan di kampus yang

telah banyak bertukar pikiran dengan penulis.

7. Rekan-rekan satu angkatan 2014 mahasiswa Teknik Informatika yang tidak lelah membantu penulis semasa masa studi.
8. M. Ghazian yang telah membantu penulis dalam membuat la-poran tugas akhir ini.

Penulis menyadari bahwa buku ini jauh dari kata sempurna. Maka dari itu, penulis memohon maaf apabila terdapat salah ka-ta maupun makna pada buku ini. Akhir kata, penulis mempersembahkan buku ini sebagai wujud nyata kontribusi penulis dalam ilmu pengetahuan.

Surabaya, Juli 2018

Christyanto Liman

DAFTAR ISI

DAFTAR GAMBAR

[*Halaman ini sengaja dikosongkan*]

DAFTAR TABEL

[*Halaman ini sengaja dikosongkan*]

DAFTAR KODE SUMBER

[*Halaman ini sengaja dikosongkan*]

BAB I
PENDAHULUAN

Pada bab ini, akan dijelaskan mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, metodologi pengerjaan, dan sistematika penulisan tugas akhir.

1.1. Latar Belakang

Permasalahan SPOJ *"Ada and Cycle"* [1] mengangkat permasalahan perhitungan jarak terpendek yang berawal dari sebuah *vertex* dan berakhir pada *vertex* yang sama untuk setiap *vertex* yang ada. Pada permasalahan ini diberikan parameter masukan berupa jumlah *vertex* (N) dan sebuah *adjacency matrix* (H). Dengan para-meter ini soal meminta hasil perhitungan jarak terpendek yang ber-awal dari sebuah *vertex* dan berakhir pada *vertex* yang sama untuk setiap *vertex* yang ada.

Berdasarkan observasi awal, permasalahan SPOJ *"Ada and Cycle"* dapat diselesaikan dengan pendekatan metode *Breadth First Search*. *Breadth First Search* (BFS) adalah algoritma penelusuran graf yang arah penelusurannya mendahulukan ke arah lebar graf tersebut. BFS menggunakan *queue* untuk menyimpan dan mendapatkan *vertex* selanjutnya untuk kemudian dilakukan pencarian.

Tetapi dalam pengaplikasiannya, metode *Breadth First Sear-ch* biasa tidak dapat memenuhi batasan waktu yang diberikan soal. Sehingga pendekatan yang dilakukan adalah pengoptimasian meto-de *Breadth First Search* sehingga memenuhi batasan waktu yang diberikan soal.

Pengoptimasian yang dilakukan dibagi menjadi dua, yaitu pengoptimasian masukan dan pengoptimasian algoritma *Breadth First Search*. Dalam pengoptimasian masukan dibuat fungsi untuk mempercepat pemrosesan masukan. Dalam pengoptimasian algoritma *Breadth First Search*, dilakukan pengurangan *vertex* yang tidak per-

lu dikunjungi sehingga sisi atau *edge* yang perlu dilewati berkurang.

1.2. Rumusan Masalah

Permasalahan yang akan diselesaikan pada tugas akhir ini ada-lah sebagai berikut:

1. Bagaimana strategi penyelesaian permasalahan *Ada and Cycle* pada situs SPOJ?
2. Bagaimana menganalisis dan membuat desain dari solusi pe-nyelesaian masalah *Ada and Cycle* pada situs SPOJ?
3. Bagaimana kinerja algortima yang telah dibuat untuk perma-salahan *Ada and Cycle*?

1.3. Batasan Masalah

Masalah yang akan diselesaikan memiliki batasan-batasan berikut:

1. Studi kasus yang digunakan adalah permasalahan *Ada and Cycle* pada situs SPOJ dengan nomer soal 28409 dan kode soal ADACYCLE.
2. Implementasi dilakukan menggunakan bahasa pemrograman C++.
3. Graf yang digunakan adalah graf berarah yang tidak berbobot.
4. Batas maksimum jumlah *vertex* (N) adalah 2.000 *vertex*.
5. Nilai bobot H_{ij} adalah 1 jika terdapat jalan yang menghubungkan *vertex* i dengan j.
6. Nilai bobot H_{ij} adalah 0 jika tidak terdapat jalan yang meng-hubungkan *vertex* i dengan j.
7. Batas waktu pada program dalam 1 kali percobaan di bawah 3 detik.
8. Penyimpanan yang dibutuhkan dalam 1 kali percobaan di ba-wah 1536 *Megabyte*.

1.4. Tujuan

Tujuan tugas akhir ini adalah sebagai berikut:

1. Menyusun strategi penyelesaian permasalahan *Ada and Cycle* pada situs SPOJ.
2. Melakukan analisis dan desain algoritma dan struktur data un-tuk menyelesaikan permasalahan *Ada and Cycle* pada situs SPOJ.
3. Mengevaluasi hasil dari kinerja algoritma dan struktur data untuk menyelesaikan permasalahan *Ada and Cycle* pada situs SPOJ.

1.5. Metodologi

Metodologi pengerjaan yang digunakan pada tugas akhir ini memiliki beberapa tahapan. Tahapan-tahapan tersebut yaitu:

1. Penyusunan proposal

   Pada tahapan ini penulis memberikan penjelasan mengenai apa yang penulis akan lakukan dan mengapa tugas akhir ini dilakukan. Penjelasan tersebut dituliskan dalam bentuk pro-posal tugas akhir.
2. Studi literatur

   Pada tahapan ini penulis mengumpulkan referensi yang di-perlukan guna mendukung pengerjaan tugas akhir. Referensi yang digunakan dapat berupa hasil penelitian yang sudah per-nah dilakukan, buku, artikel internet, atau sumber lain yang bisa dipertanggungjawabkan.
3. Implementasi algoritma

   Pada tahapan ini penulis mulai mengembangkan algoritma yang digunakan untuk menyelesaiakan permasalahan *Ada and Cycle*.
4. Pengujian dan evaluasi

   Pada tahapan ini penulis menguji performa algoritma yang

digunakan. Hasil pengujian kemudian dievaluasi untuk kemudian dipertimbangkan apakah algoritma masih bisa diting-katkan lagi atau tidak.

5. Penyusunan buku

   Pada tahapan ini penulis menyusun hasil pengerjaan tugas akhir mengikuti format penulisan tugas akhir.

1.6. Sistematika Penulisan

Sistematika laporan tugas akhir yang akan digunakan adalah sebagai berikut:

1. Bagian awal, meliputi halaman depan, halaman pengesahan, abstrak, kata pengantar, daftar isi, daftar gambar, dan daftar tabel.
2. Bagian inti, meliputi pendahuluan, tinjauan pustaka, metodologi, hasil dan pembahasan, dan kesimpulan dan saran.
3. Bagian akhir, meliputi daftar pustaka, lampiran-lampiran, dan biodata penulis.

BAB II
DASAR TEORI

Pada bab ini, dasar teori yang digunakan sebagai landasan pengerjaan tugas akhir ini akan dijabarkan.

2.1. Deskripsi Soal

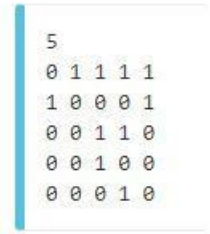
Permasalahan yang akan dibahas adalah melakukan pencari-an jalan tercepat yang dimulai dari *vertex* i dan berakhir di *vertex* i.

2.1.1. Parameter Masukan

Parameter masukan yang diberikan adalah sebagai berikut: Baris pertama merupakan bilangan N jumlah *vertex* dan N baris be-rikutnya adalah bilangan H_{ij} sebanyak N yang menyatakan ada ti-daknya jalan, dimana i menunjukan baris dan j menunjukan kolom. Seluruh bilangan H akan membentuk suatu matriks sebesar N kali N dan dapat disebut sebagai *adjacency marix*.

1. N. $0 \le N \le 2000$, sebuah bilangan yang menyatakan jumlah *vertex*
2. H_{ij} . H_{ij} = {0, 1}, sebuah bilangan yang menyatakan ada tidaknya jalan dari *vertex* i ke *vertex* j, 1 berarti ada jalan yang menghubungkan *vertex* i dan *vertex* j, 0 berarti tidak ada jalan

Pada Gambar 2.1, dapat dilihat baris pertama adalah angka 5 yang menyatakan N adalah 5. Pada baris kedua atau baris bilangan H pertama, bilangan pertama atau H 1 1 yaitu 0 menandakan ti-menuju vertex pertama. Bilangan dak ada jalan dari *vertex* pertama kedua pada baris bilangan H pertama atau H 1 2 yaitu 1 menan-vertex kedua. Bilangan dakan ada jalan dari *vertex* pertama menuju ketiga pada baris bilangan H pertama atau H 1 3 yaitu 1 menan-



Gambar 2.1: Contoh Masukan

dakan ada jalan dari *vertex* pertama menuju vertex ketiga. Bilangan keempat pada baris bilangan H pertama atau H 1 4 yaitu 1 menandakan ada jalan dari *vertex* pertama menuju vertex keempat. Bilangan kelima pada baris bilangan H pertama atau H 1 5 yaitu 1 menandakan ada jalan dari *vertex* pertama menuju *vertex* kelima.
Hal ini juga berlaku untuk semua H pada setiap baris selanjutnya.

2.1.2. Batasan Permasalahan

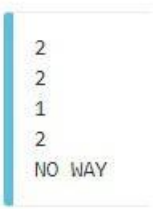
Batasan yang diberikan soal adalah batas *runtime* 3 detik, dan batas penggunaan memori 1536MB.

2.1.3. Keluaran Permasalahan

Keluaran yang diminta adalah tuliskan sebanyak N, panjang dari jalan tercepat yang berawal dari *vertex* i dan berakhir di *vertex* i. Jika tidak ada jalan maka tuliskan “NO WAY”. Contoh keluaran dari masukan pada Gambar 2.1 dapat dilihat pada Gambar 2.2.

2.2. Landasan Teori

Subbab ini akan memaparkan definisi, deskripsi dan landasan yang akan digunakan pada keseluruhan penyelesaian masalah.



Gambar 2.2: Contoh Keluaran

2.2.1. Teori Graf

Graf digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut [2], [3]. Graf terdiri da-ri simpul-simpul (*vertices*) dan sisi (*edges*) yang menghubungkan sepasang simpul. Graf adalah sebuah *ordered pair* G=(V,E) yang terdiri dari kumpulan simpul-simpul (*vertices*) atau titik V dan kumpulan sisi (*edges*) E. Sebuah sisi E dapat dinotasikan dengan E=(u,v) yang berarti ada sisi yang menghubungkan *vertex* u dan *vertex* v.

Berdasarkan ada tidaknya gelang atau sisi ganda pada suatu graf, maka graf digolongkan menjadi dua jenis:

1. Graf sederhana, yaitu graf yang tidak mengandung sisi ganda atau gelang.
2. Graf tak-sederhana, yaitu graf yang mengandung sisi ganda atau gelang.

Graf sederhana memiliki beberapa jenis:

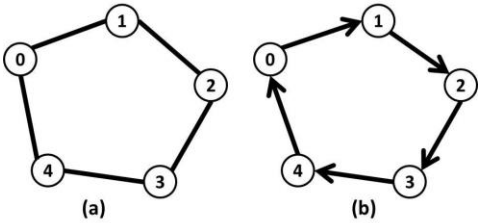
1. Graf komplit, yaitu graf dimana setiap pasang *vertices* selalu memiliki sebuah *edge*.
2. Graf *Cycle*, yaitu graf yang terdiri dari sebuah siklus, dalam kata lain beberapa *vertices* terhubung menjadi sebuah rantai tertutup.
3. Graf *Wheel*, yaitu graf *cycle* yang ditambahi sebuah *vertex*

baru yang terhubung dengan semua *vertices*.

Berdasarkan orientasi arah pada sisi, maka graf digolongkan menjadi dua jenis:

1. Graf tak-berarah, yaitu graf yang sisinya tidak mempunyai orientasi arah.
2. Graf berarah, yaitu graf yang setiap sisinya diberikan orientasi arah.

Representasi dari graf tak-berarah dan graf berarah dapat dilihat pa-da Gambar 2.3.



Gambar 2.3: Representasi Graf Tak-berarah (a) dan Graf Berarah (b)

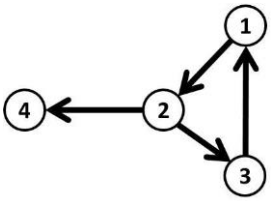
2.2.2. Properti Graf

2.2.2.1. Derajat Vertex

Derajat dari sebuah *vertex* (v) dari sebuah graf (g) adalah jum-lah dari sisi graf yang bersentuhan dengan v [4]. Derajat *vertex* juga disebut dengan *valency*. Susunan terurut dari derajat *vertex* pada se-buah graf disebut sekuens derajat.

Pada graf berarah terdapat dua jenis derajat vertex yaitu:

1. Derajat masuk (*in degree*), derajat masuk adalah susunan dari sisi graf yang arahnya masuk atau menuju suatu *vertex* [5].



Gambar 2.4: Ilustrasi Derajat *Vertex*

Berdasarkan Gambar 2.4, derajat masuk dari masing-masing *vertex* dapat dituliskan sebagai berikut

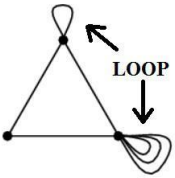
(a) *Vertex* 1 memiliki jumlah *in degree* 1 yaitu melalui *ver-tex* 3.
(b) *Vertex* 2 memiliki jumlah *in degree* 1 yaitu melalui *ver-tex* 1.
(c) *Vertex* 3 memiliki jumlah *in degree* 1 yaitu melalui *ver-tex* 2.
(d) *Vertex* 4 memiliki jumlah *in degree* 1 yaitu melalui *ver-tex* 2.

2. Derajat keluar (*out degree*), derajat keluar adalah susunan da-ri sisi graf yang arahnya keluar dari suatu *vertex*[6]. Berda-sarkan Gambar 2.4, derajat keluar dari masing-masing *vertex* dapat dituliskan sebagai berikut

(a) *Vertex* 1 memiliki jumlah *out degree* 1 yaitu menuju *ver-tex* 2.
(b) *Vertex* 2 memiliki jumlah *out degree* 2 yaitu menuju *ver-tex* 3 dan *vertex* 4.
(c) *Vertex* 3 memiliki jumlah *out degree* 1 yaitu menuju *ver-tex* 1.
(d) *Vertex* 4 memiliki jumlah *out degree* 0.

2.2.2.2. Loop

Loop atau putaran pada suatu graf berarti sebuah degenerasi sisi graf yang menghubungkan sebuah *vertex* ke dirinya sendiri, disebut juga sebagai *self-loop*. Sebuah graf sederhana tidak dapat memiliki *loop*, tetapi graf tak-sederhana dapat memiliki *loop* dan sisi ganda [7]. Ilustrasi *Loop* dapat dilihat pada Gambar 2.5.



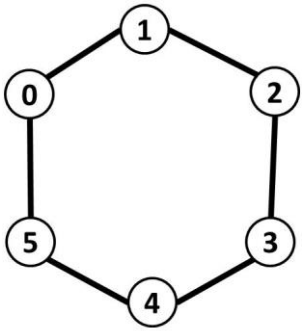
Gambar 2.5: Ilustrasi *Loop*

2.2.2.3. Lintasan

Lintasan pada graf adalah serangkaian *vertex* dan *edge* yang membentuk subgraf dimana setiap *vertex* dan *edge* nya dapat mem-bentuk sebuah garis lurus.[8], [9] Panjang dari sebuah lintasan graf adalah jumlah *edge* pada lintasan tersebut. Lintasan yang berawal dari suatu *vertex* dan berakhir pada *vertex* yang sama disebut de-ngan *Closed Walk* atau *Cycle*. Lintasan yang berawal dari suatu *vertex* dan berakhir pada *vertex* yang berbeda disebut dengan *Open Walk*.

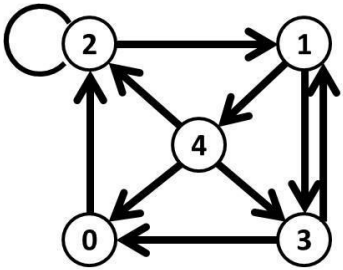
2.2.2.4. Graf Siklus

Graf siklus atau *Cycle Graph* merupakan graf yang memiliki lintasan yang berawal dan berakhir pada *vertex* yang sama. Panjang dari siklus graf terpendek pada sebuah graf disebut *girth*, sedangkan panjang dari siklus graf terpanjang pada sebuah graf disebut keliling graf (*graph circumference*) [10]. Ilustrasi Graf Siklus dapat dilihat pada Gambar 2.6.



Gambar 2.6: Ilustrasi Graf Siklus Dengan Panjang 6

2.2.3. Representasi Graf



Gambar 2.7: Ilustrasi Graf Berarah

Terdapat dua jenis representasi graf yang paling sering dipa-kai, [11] yaitu:

1. *Adjacency Matrix*, *adjacency matrix* adalah *array* 2 dimensi berukuran N kali N dimana N adalah jumlah dari *vertex* pada suatu graf. Diketahui *array* 2 dimensi adj[][], jika pada adj[i][j] = 1, berarti ada edge yang menghubungkan *vertex* i dan *vertex* j. Pada graf tak-berarah, *adjacency matrix* selalu simetris. Keuntungan dari representasi ini adalah:

(a) representasi lebih mudah untuk diimplemetasikan dan diikuti.
(b) Membuang sebuah sisi memakan waktu hanya O(1).
(c) *query* yang menyangkut tentang ada tidaknya sisi pada *vertex* i ke *vertex* j dapat dilakukan secara efisien dengan O(1).

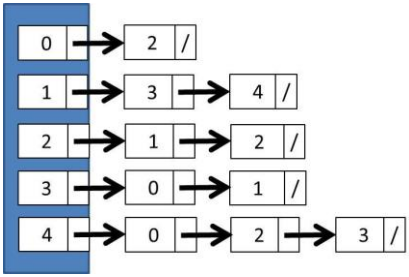
Kekurangan dari representasi ini adalah memakan banyak ru-ang ($O(N^2)$) meskipun hanya memiliki sedikit sisi dan me-nambahkan sebuah *vertex* membutuhkan waktu ($O(N^2)$). Re-presentasi *adjacency matrix* dari Gambar 2.7 dapat dilihat pa-da Gambar 2.8.

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

Gambar 2.8: Ilustrasi Adjacency Matrix

2. *Adjacency List*, dalam representasi *adjacency list* digunakan *array* dari *linked list.* Besar dari array tersebut adalah sama dengan jumlah *vertex* yang ada. Dalam *Adjacency List* dike-tahui array[],maka sebuah array[i] merepresentasikan *linked list* dari *vertex* yang terhubung (*adjacent*) dengan *vertex* ke-i. Keuntungan dari representasi ini adalah lebih menghemat ruang dengan O(|V|+|E|), dalam kasus terburuk dimana jum-lah sisi(E) adalah 2 kali dari jumlah *vertex* atau C(V,2) maka akan memakan ruang sebanyak $O(N^2)$. Selain itu untuk me-nambahkan vertex lebih mudah pada representasi *adjacency list.* Kekurangan dari representasi ini adalah *query* yang me-nyangkut tentang ada tidaknya sisi pada *vertex* i ke *vertex* j

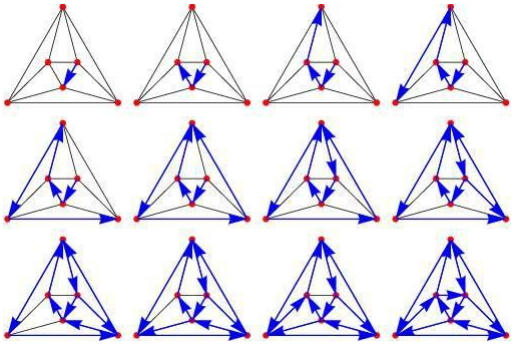
tidak efisien dan akan memakan waktu O(V). Representasi *adjacency list* dari Gambar 2.7 dapat dilihat pada Gambar 2.9.



Gambar 2.9: Ilustrasi Adjacency List

2.2.4. Siklus Eulerian

Siklus *eulerian* atau dapat disebut sirkuit *Eulerian*, sirkuit *Euler*, *Eulerian tour* atau *Euler tour*, adalah sebuah lintasan yang berawal dan berakhir pada *vertex* graf yang sama. Dalam kata lain, siklus *eulerian* adalah graf siklus yang menggunakan setiap sisi graf hanya sekali [12]. Gambar 2.10, merepresentasikan siklus *eulerian* pada graf oktahedral.



Gambar 2.10: Siklus Eulerian Untuk Graf Oktahedral

2.2.5. Breadth First Search (BFS)

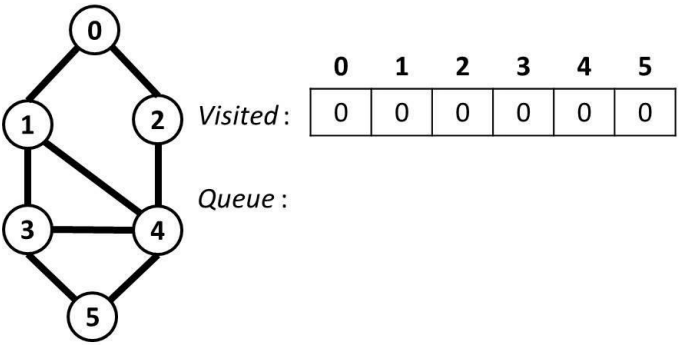
Breadth First Search (BFS) adalah algoritma penelusuran graf yang arah penelusurannya mendahulukan ke arah lebar graf terse-but [2], [13]. BFS menggunakan *queue* untuk menyimpan dan mendapatkan *vertex* selanjutnya untuk kemudian dilakukan pencarian. Kompleksitas dari BFS sendiri adalah O(V+E) dimana V merupa-kan jumlah *vertex* dan E merupakan jumlah *edge*, sehingga kom-pleksitas kemungkinan terburuk dari algoritma BFS adalah $O(N^2)$ dimana setiap *vertex* terhubung dengan semua *vertex* yang ada.[13]

BFS mengikuti aturan sebagai berikut:

1. Kunjungi *vertex* berdekatan yang belum pernah dikunjungi. Tandai *vertex* tersebut telah dikunjungi. masukkan ke dalam *queue*.
2. Jika tidak ada *vertex* yang berdekatan, keluarkan *vertex* pertama pada queue.
3. Ulangi langkah 1 dan 2 hingga *queue*

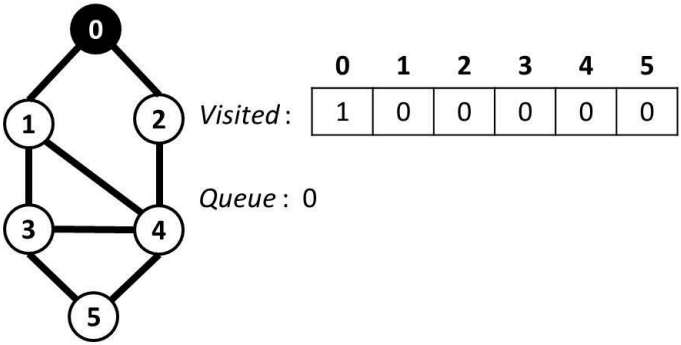
   kosong. Ilustrasi :

1. Terdapat graf seperti pada Gambar 2.11.



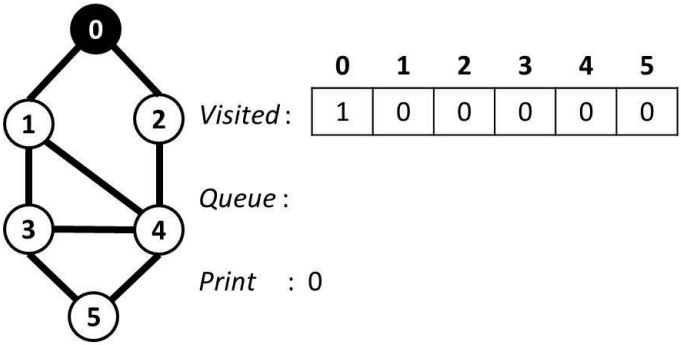
Gambar 2.11: Ilustrasi Algoritma BFS 1

2. Kunjungi *vertex* 0 dan masukkan ke dalam *queue*. Ilustrasi langkah ini dapat dilihat pada Gambar 2.12.



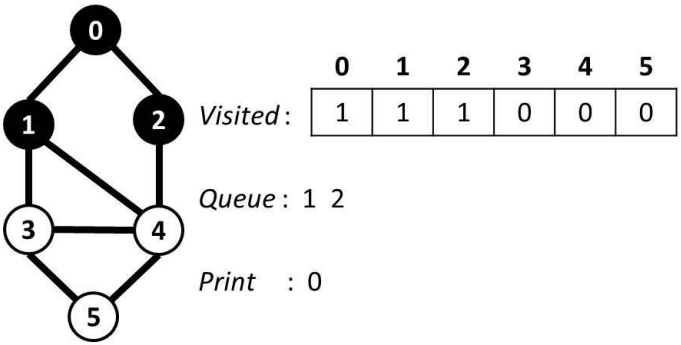
Gambar 2.12: Ilustrasi Algoritma BFS 2

3. Keluarkan *vertex* pertama *queue*. Ilustrasi langkah ini dapat dilihat pada Gambar 2.13.



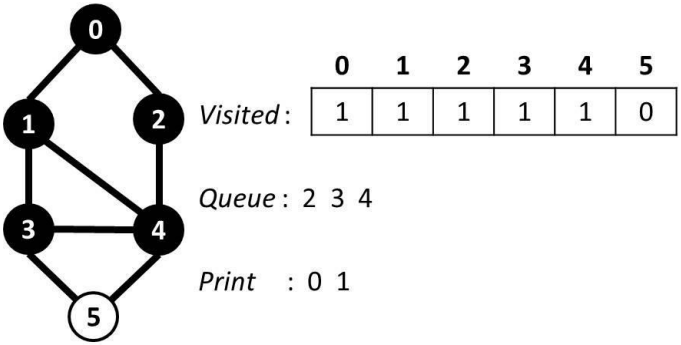
Gambar 2.13: Ilustrasi Algoritma BFS 3

4. Kunjungi *vertex* yang berdekatan dengan *vertex* 0 dan masukkan ke dalam *queue*. Ilustrasi langkah ini dapat dilihat pada Gambar 2.14.



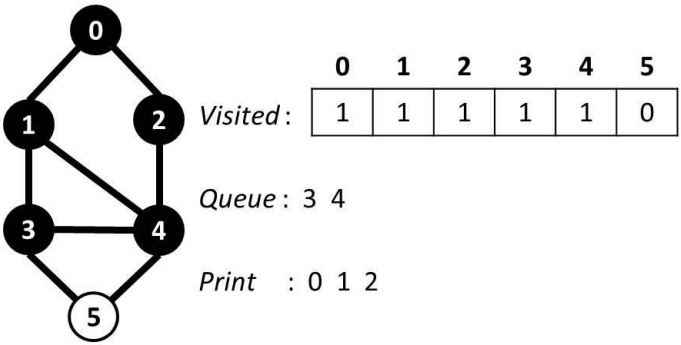
Gambar 2.14: Ilustrasi Algoritma BFS 4

5. Keluarkan *vertex* pertama *queue* dan kunjungi *vertex* yang berdekatan dengannya. Ilustrasi langkah ini dapat dilihat pa-da Gambar 2.15.



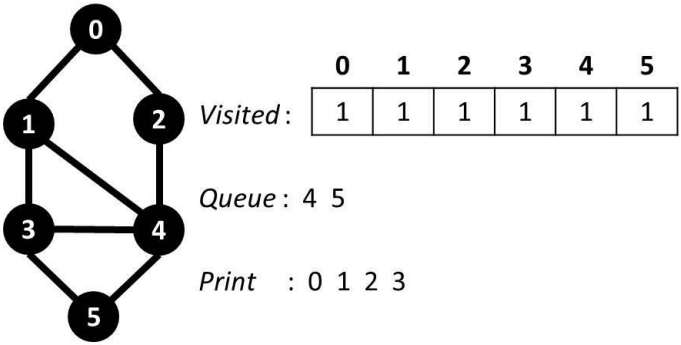
Gambar 2.15: Ilustrasi Algoritma BFS 5

6. Keluarkan *vertex* pertama *queue*. Ilustrasi langkah ini dapat dilihat pada Gambar 2.16.



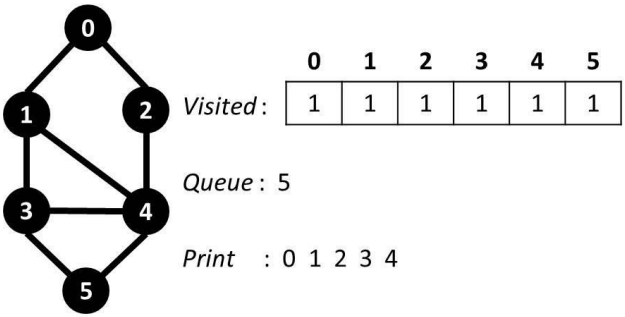
Gambar 2.16: Ilustrasi Algoritma BFS 6

7. Keluarkan *vertex* pertama *queue* dan kunjungi *vertex* yang berdekatan dengannya. Ilustrasi langkah ini dapat dilihat pa-da Gambar 2.17.



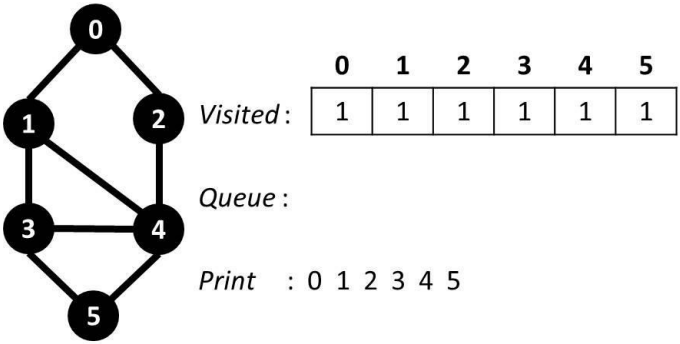
Gambar 2.17: Ilustrasi Algoritma BFS 7

8. Keluarkan *vertex* pertama *queue*. Ilustrasi langkah ini dapat dilihat pada Gambar 2.18.



Gambar 2.18: Ilustrasi Algoritma BFS 8

9. Keluarkan *vertex* pertama *queue*. Ilustrasi langkah ini dapat dilihat pada Gambar 2.19.



Gambar 2.19: Ilustrasi Algoritma BFS 9

2.3. Strategi Penyelesaian Umum

Tujuan permasalahan ini adalah mencari panjang dari jalan tercepat yang berawal dari *vertex* i dan berakhir di *vertex* i. Berdasarkan observasi awal, permasalahan SPOJ *"Ada and Cycle"* dapat diselesaikan dengan pendekatan metode *Breadth First Search*. Te-

tapi saat pengujian dilakukan pada *Sphere Online Judge* menggunakan algoritma BFS biasa, waktu pemrosesan yang didapat adalah *"time limit exceeded"*. Hal ini terjadi karena proses BFS dilakukan sebanyak jumlah *vertex* yang ada sehingga kompleksitas yang di-dapat adalah kompleksitas BFS dikali jumlah *vertex* yaitu $O(N^3)$. Dengan kompleksitas $O(N^3)$, waktu pemrosesan yang digunakan program akan melebihi batas waktu yang diharapkan yaitu 3 de-tik. Untuk mengatasi hal ini, dilakukan dilakukan pengoptimasian algoritma BFS dengan melakukan pengurangan *vertex* yang tidak diperlukan.

Subbab ini menjelaskan strategi penyelesaian permasalahan SPOJ *"Ada and Cycle"* dengan pengoptimasian algoritma BFS. Se-cara garis besar strategi penyelesaian dibagi menjadi beberapa ba-gian, yaitu:

1. Pemrosesan masukan, meliputi bagian memasukkan parame-ter masukan, pembuatan *adjacency list* dan pencatatan jumlah *degree* masuk dan *degree* keluar.
2. Proses inisialisasi awal, meliputi bagian pengecekan jumlah *degree* masuk dari *vertex* tujuan, pengosongan penanda *visited* dari tiap *vertex*, pengosongan *queue* BFS, memasukkan *vertex* awal ke dalam *queue* dan pemberian jarak awal.
3. Proses iterasi menggunakan *queue*.

Dalam pemrosesan masukan dibutuhkan variabel N yang me-rupakan banyak *vertex* dan *adjacency matrix* H menjadi parameter masukan yang sudah diberikan oleh soal. Dalam pemrosesan ma-sukan *adjacency matrix* disimpan dalam bentuk *adjacency list* dan dicatat *degree* masuk dan *degree* keluarnya.

Kemudian dilanjutkan dengan proses inisialisasi awal yaitu pengecekan apakah *vertex* awal memiliki *degree* masuk atau dalam kata lain dapat dikunjungi dari *vertex* lain, jika ya, maka dilanjut-kan dengan langkah selanjutnya, jika tidak, maka dapat dipastikan keluaran yang dihasilkan adalah *"NO WAY"*. Langkah inisialisa-

si awal selanjutnya adalah pengosongan penanda *visited* dari tiap *vertex*, pengosongan *queue* BFS dan pemberian penanda tujuan ditemukan adalah 0 atau berarti *vertex* tujuan belum ditemukan. Kemudian vertex awal dimasukkan ke dalam *queue* BFS, diberi jarak awal 0, tandai *visited* pada *vertex* awal, dan kemudian dilakukan proses iterasi pencarian jarak terpendek menggunakan *queue* BFS.

Dalam proses iterasi secara BFS, dilakukan eliminasi *vertex* pada saat mengunjungi setiap *vertex* dimana jika *vertex* tersebut su-dah dikunjungi atau jika *vertex* tersebut tidak memiliki *degree* ke-luar maka tidak perlu dimasukkan ke dalam *queue* BFS. BFS akan dihentikan jika menemui kondisi sebagai berikut:

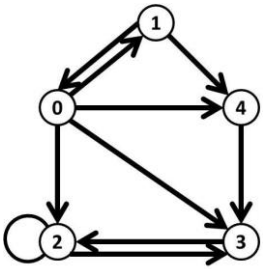
1. *Queue* kosong atau tidak ditemukan jalan untuk kembali ke *vertex* i.
2. Ditemukan *cycle* dalam graf untuk kembali ke *vertex* i.

Contoh kasus :

1. Terdapat masukan sebagai berikut: 5
   0 1 1 1 1
   1 0 0 0 1
   0 0 1 1 0
   0 0 1 0 0
   0 0 0 1 0

   Ilustrasi graf dari kasus ini dapat dilihat pada Gambar 2.20.

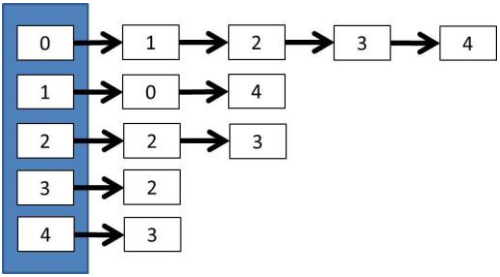
2. Langkah pertama adalah proses pembuatan *adjacency list* da-ri *adjacency matrix* masukan dan pencatatan jumlah *in degree* dan *out degree* dari masing-masing *vertex*. Sehingga didapatkan *adjacency list* seperti pada Gambar 2.21 dan Tabel 2.1.



Gambar 2.20: Ilustrasi Graf Kasus

Tabel 2.1: Jumlah In Degree dan Out Degree

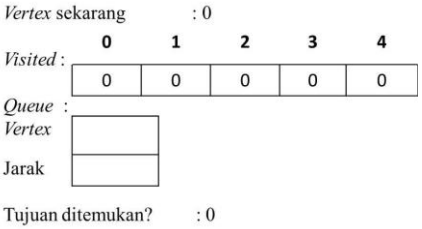
Vertex(i)	In Degree	Out Degree
0	1	4
1	1	2
2	3	2
3	3	1
4	2	1



Gambar 2.21: Ilustrasi Adjacency List

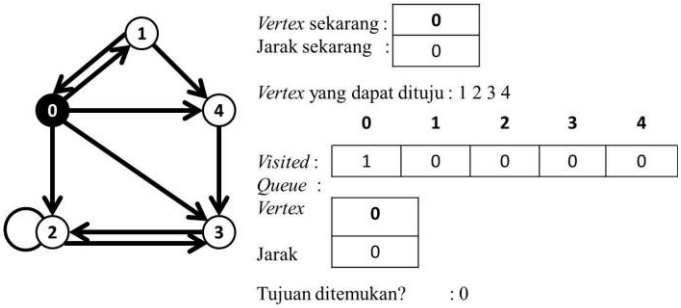
3. Langkah selanjutnya adalah proses pencarian jarak terpendek untuk setiap vertex yang ada. Proses pencarian jarak terpen-dek dibagi menjadi dua bagian yaitu proses inisialisasi dan proses iterasi dengan *queue* BFS.

4. Inisialisasi dimulai dari *vertex* i=0, cek apakah *vertex* i=0 me-miliki *in degree* lebih dari 0. *In degree* dari i=0 adalah 1, se-hingga langkah berlanjut.
5. Kosongkan semua penanda *visited* pada tiap *vertex*, kosong-kan semua isi dari *queue* BFS. Ilustrasi langkah ini dapat di-lihat pada Gambar 2.22.



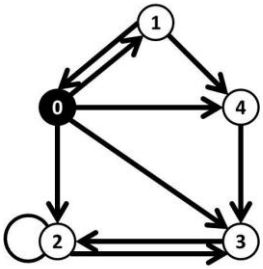
Gambar 2.22: Inisialisasi Awal Pada i=0

6. Untuk i=0, masukkan *vertex* ke dalam *queue* BFS dengan ja-rak awal 0, tandai *visited* dengan nilai 1 atau *true* dan tuju-an ditemukan adalah 0 atau *false*. Lanjutkan ke iterasi perta-ma dalam *queue* BFS. Ilustrasi langkah ini dapat dilihat pada Gambar 2.23.



Gambar 2.23: Inisialisasi Awal 2 Pada i=0

7. Iterasi pertama untuk *vertex* i=0, cek apakah *vertex* 0 terdapat loop, masukkan semua *vertex* yang dapat dikunjungi dari *vertex* i=0 dan beri jarak yaitu 1. Ilustrasi langkah ini dapat dilihat pada Gambar 2.24.



Vertex sekarang : 0
Jarak sekarang : 0

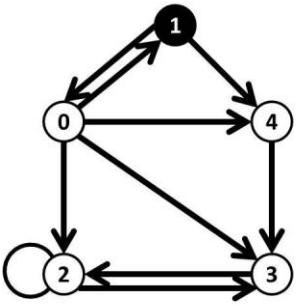
Vertex yang dapat dituju : 1 2 3 4

	0	1	2	3	4
Visited :	1	1	1	1	1
Queue :					
Vertex	**1**	**2**	**3**	**4**	
Jarak	1	1	1	1	

Tujuan ditemukan? : 0

Gambar 2.24: Ilustrasi Graf Kasus Iterasi 1 i=0

8. Iterasi kedua kunjungi *vertex* pada *queue* terdepan, cek jika dari *vertex* 1 dapat menuju *vertex* 0. Ilustrasi langkah ini dapat dilihat pada Gambar 2.25.



Vertex sekarang : 1
Jarak sekarang : 1

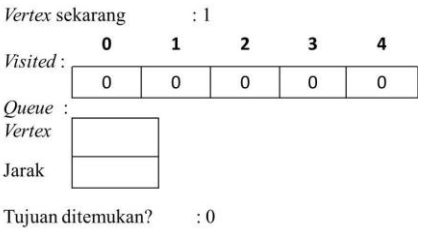
Vertex yang dapat dituju : **0** 4

	0	1	2	3	4
Visited :	1	1	1	1	1
Queue :					
Vertex	**2**	**3**	**4**		
Jarak	1	1	1		

Tujuan ditemukan? : 1

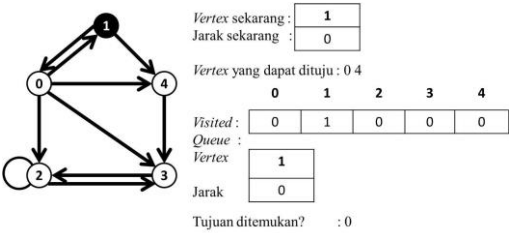
Gambar 2.25: Ilustrasi Graf kasus iterasi 2 i=0

9. Karena *vertex* 0 adalah tujuan akhir, maka *vertex* tujuan telah ditemukan, sehingga iterasi untuk i=0 berakhir dan jaraknya adalah jarak sekarang+1 yaitu 2. Keluarkan jarak=2 sebagai hasil keluaran.
10. Kemudian iterasi diulang kembali untuk i=1.
11. Cek apakah *vertex* i=1 memiliki *in degree* lebih dari 0. *In degree* dari i=1 adalah 1, sehingga langkah berlanjut.
12. Kosongkan semua penanda *visited* pada tiap *vertex*, kosong-kan semua isi dari *queue* BFS. Ilustrasi langkah ini dapat di-lihat pada Gambar 2.26.



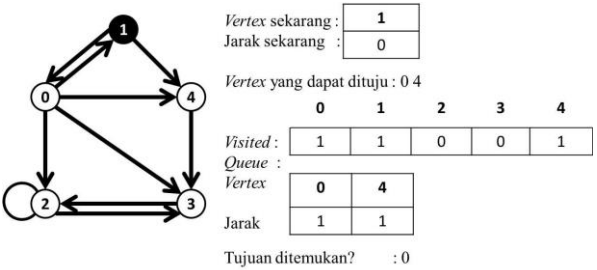
Gambar 2.26: Inisialisasi Awal Pada i=1

13. masukkan *vertex* i=1 ke dalam *queue* BFS dengan jarak awal 0, tandai *visited* dengan nilai 1 atau *true* dan tujuan ditemukan adalah 0 atau *false*. Lanjutkan ke iterasi pertama dalam *queue* BFS. Ilustrasi langkah ini dapat dilihat pada Gambar 2.27.



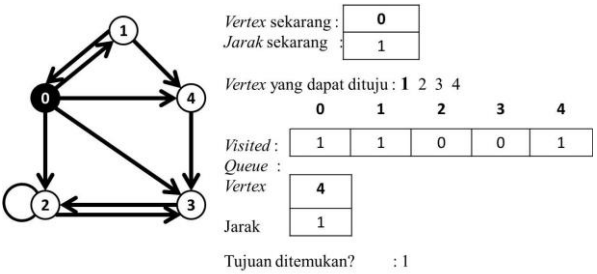
Gambar 2.27: Inisialisasi Awal 2 Pada i=1

14. Iterasi pertama untuk *vertex* i=1, cek apakah *vertex* 1 terdapat loop, masukkan semua *vertex* yang dapat dikunjungi dari *vertex* i=1 dan beri jarak yaitu 1. Ilustrasi langkah ini dapat dilihat pada Gambar 2.28.



Gambar 2.28: Ilustrasi Graf Kasus Iterasi 1 i=1

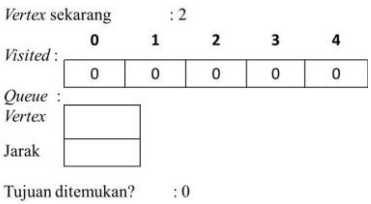
15. Iterasi kedua kunjungi *vertex* pada *queue* terdepan, cek jika dari *vertex* 0 dapat menuju *vertex* 1. Ilustrasi langkah ini dapat dilihat pada Gambar 2.29.



Gambar 2.29: Ilustrasi Graf Kasus Iterasi 2 i=1

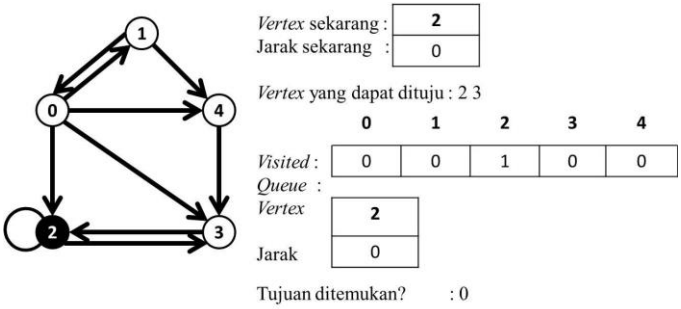
16. Karena *vertex* 1 adalah tujuan akhir, maka *vertex* tujuan telah ditemukan, sehingga iterasi untuk i=1 berakhir dan jaraknya adalah jarak sekarang+1 yaitu 2. Keluarkan jarak=2 sebagai hasil keluaran.
17. Kemudian iterasi diulang kembali untuk i=2.

18. Cek apakah *vertex* i=2 memiliki *in degree* lebih dari 0. *In degree* dari i=2 adalah 3, sehingga langkah berlanjut.
19. Kosongkan semua penanda *visited* pada tiap *vertex*, kosong-kan semua isi dari *queue* BFS. Ilustrasi langkah ini dapat di-lihat pada Gambar 2.30.



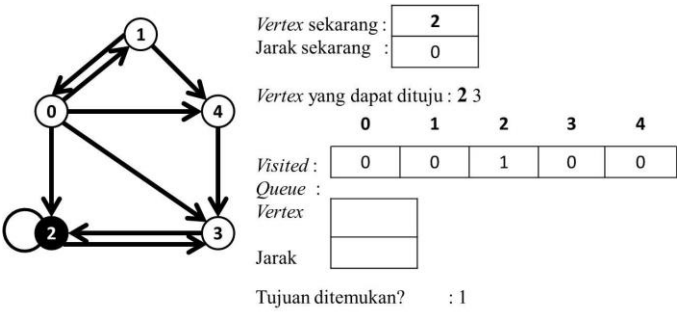
Gambar 2.30: Inisialisasi Awal Pada i=2

20. masukkan *vertex* i=2 ke dalam *queue* BFS dengan jarak awal 0, tandai *visited* dengan nilai 1 atau *true* dan tujuan ditemukan adalah 0 atau *false*. Lanjutkan ke iterasi pertama dalam *queue* BFS. Ilustrasi langkah ini dapat dilihat pada Gambar 2.31.



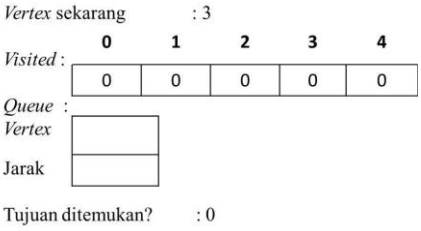
Gambar 2.31: Inisialisasi Awal 2 Pada i=2

21. Iterasi pertama untuk *vertex* i=1, cek jika dari *vertex* 2 ter-dapat loop. Ilustrasi langkah ini dapat dilihat pada Gambar 2.32.



Gambar 2.32: Ilustrasi Graf Kasus Iterasi 1 i=2

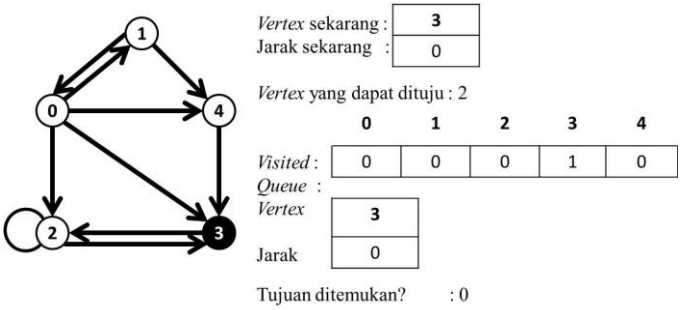
22. Karena *vertex* 2 memiliki loop, maka *vertex* tujuan telah dite-mukan, sehingga iterasi untuk i=2 berakhir dan jaraknya ada-lah jarak sekarang+1 yaitu 1. Keluarkan jarak=1 sebagai hasil keluaran.
23. Kemudian iterasi diulang kembali untuk i=3.
24. Cek apakah *vertex* i=3 memiliki *in degree* lebih dari 0. *In degree* dari i=3 adalah 3, sehingga langkah berlanjut.
25. Kosongkan semua penanda *visited* pada tiap *vertex*, kosong-kan semua isi dari *queue* BFS. Ilustrasi langkah ini dapat di-lihat pada Gambar 2.33.



Gambar 2.33: Inisialisasi Awal Pada i=3

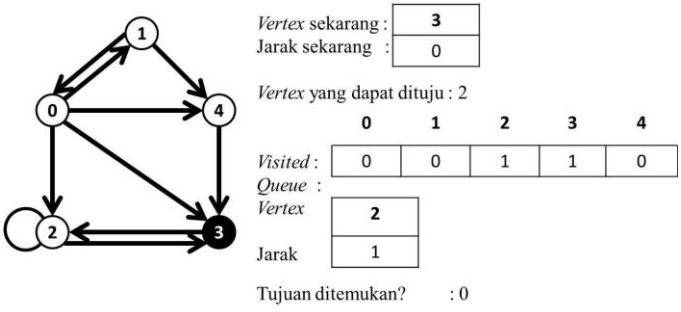
26. masukkan *vertex* i=3 ke dalam *queue* BFS dengan jarak awal 0, tandai *visited* dengan nilai 1 atau *true* dan tujuan ditemukan

adalah 0 atau *false*. Lanjutkan ke iterasi pertama dalam *queue* BFS. Ilustrasi langkah ini dapat dilihat pada Gambar 2.34.



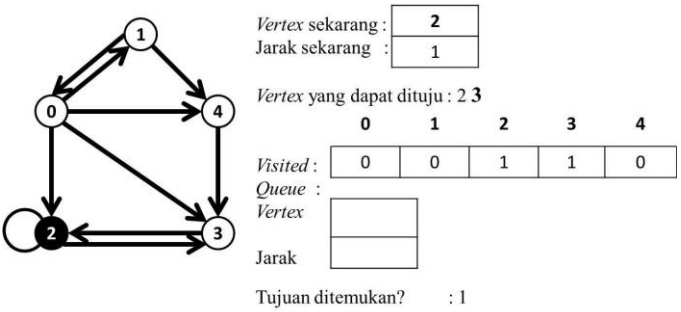
Gambar 2.34: Inisialisasi Awal 2 Pada i=3

27. Iterasi pertama untuk *vertex* i=3, cek apakah *vertex* 3 terdapat loop, masukkan semua *vertex* yang dapat dikunjungi dari *vertex* i=3 dan beri jarak yaitu 1. Ilustrasi langkah ini dapat dilihat pada Gambar 2.35.



Gambar 2.35: Ilustrasi Graf Kasus Iterasi 1 i=3

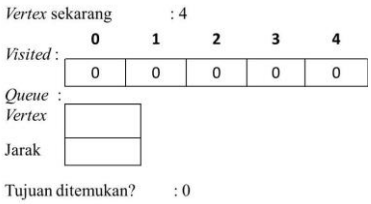
28. Iterasi kedua kunjungi *vertex* pada *queue* terdepan, cek jika dari *vertex* 2 dapat menuju *vertex* 3. Ilustrasi langkah ini dapat dilihat pada Gambar 2.36.
29. Karena *vertex* 3 adalah tujuan akhir, maka *vertex* tujuan telah



Gambar 2.36: Ilustrasi Graf Kasus Iterasi 2 i=3

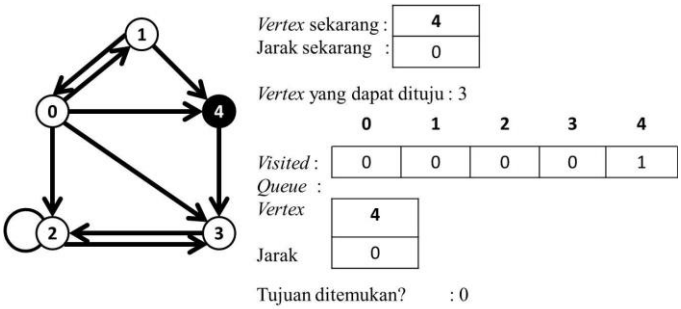
ditemukan, sehingga iterasi untuk i=3 berakhir dan jaraknya adalah jarak sekarang+1 yaitu 2. Keluarkan jarak=2 sebagai hasil keluaran.

30. Kemudian iterasi diulang kembali untuk i=4.
31. Cek apakah *vertex* i=4 memiliki *in degree* lebih dari 0. *In degree* dari i=4 adalah 2, sehingga langkah berlanjut.
32. Kosongkan semua penanda *visited* pada tiap *vertex*, kosong-kan semua isi dari *queue* BFS. Ilustrasi langkah ini dapat di-lihat pada Gambar 2.37.



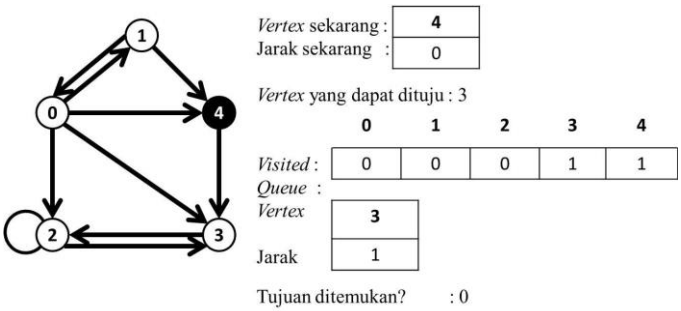
Gambar 2.37: Inisialisasi Awal Pada i=4

33. masukkan *vertex* i=4 ke dalam *queue* BFS dengan jarak awal 0, tandai *visited* dengan nilai 1 atau *true* dan tujuan ditemukan adalah 0 atau *false*. Lanjutkan ke iterasi pertama dalam *queue* BFS. Ilustrasi langkah ini dapat dilihat pada Gambar 2.38.



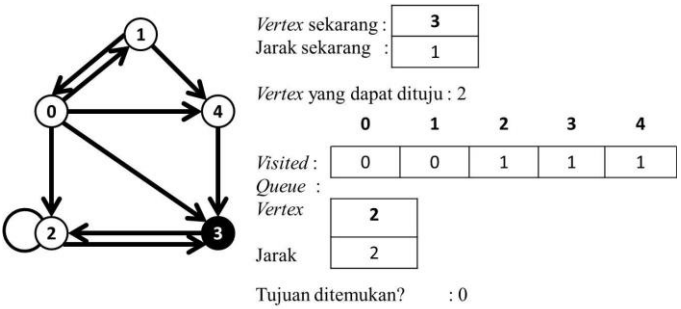
Gambar 2.38: Inisialisasi Awal 2 Pada i=4

34. Iterasi pertama untuk *vertex* i=4, cek apakah *vertex* 4 terdapat loop, masukkan semua *vertex* yang dapat dikunjungi dari *vertex* i=4 dan beri jarak yaitu 1. Ilustrasi langkah ini dapat dilihat pada Gambar 2.39.



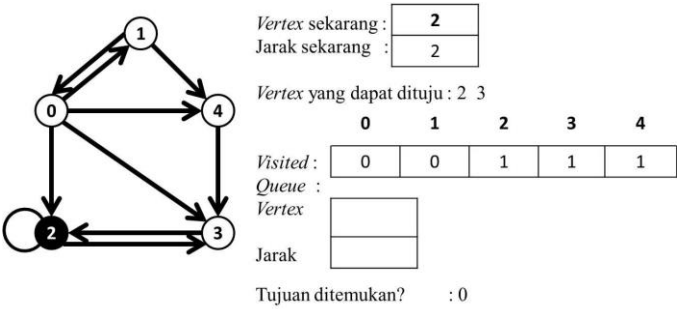
Gambar 2.39: Ilustrasi Graf Kasus Iterasi 1 i=4

35. Iterasi kedua kunjungi *vertex* pada *queue* terdepan, cek jika dari *vertex* 3 dapat menuju *vertex* 4, jika tidak, masukkan se-mua *vertex* yang dapat dikunjungi dari *vertex* i=3 dan beri jarak sekarang+1 yaitu 2. Ilustrasi langkah ini dapat dilihat pada Gambar 2.40.
36. Iterasi kedua kunjungi *vertex* pada *queue* terdepan, cek jika



Gambar 2.40: Ilustrasi Graf Kasus Iterasi 2 i=4

dari *vertex* 2 dapat menuju *vertex* 4. Cek apakah ada *vertex* yang dapat dikunjungi lagi melalui *vertex* 2. Karena *vertex* 2 hanya bisa menuju *vertex* 2 dan 3 yang sudah pernah dikunju-ngi (penanda *visited*=1) maka BFS dihentikan dengan jarak 0 atau tidak ditemukan jalan. Ilustrasi langkah ini dapat dilihat pada Gambar 2.41.



Gambar 2.41: Ilustrasi Graf Kasus Iterasi 3 i=4

37. Karena pada *vertex* 4 tidak ditemukan jalan kembali, maka hasil keluarannya adalah *string "NO WAY"*.

[*Halaman ini sengaja dikosongkan*]

BAB III
DESAIN

Pada bab ini akan dijelaskan desain algoritma yang akan digunakan untuk menyelesaikan permasalahan.

3.1. Deskripsi Umum Sistem

Sistem akan menerima dua nilai masukan: N dan H_{ij} . Nilai masukan ini mengikuti penjelasan pada subbab 2.1.1 Parameter Masukan. Lalu sistem akan membuat *adjacency list* dan mencatat jumlah *in degree* dan *out degree* dari setiap *vertex* berdasarkan masukan yang ada. Kemudian sistem akan melakukan pencarian cycle menggunakan fungsi BFS dan menghitung total jarak terpendek yang ditempuh untuk setiap vertex. Kemudian sistem akan mengeluarkan hasil dari perhitungan jarak menggunakan BFS.

Proses BFS dilakukan sebanyak jumlah *vertex* yang ada sehingga kompleksitas yang didapat adalah kompleksitas BFS dikali jumlah *vertex*. Pada kasus terburuk dari algoritma BFS biasa dima-na semua *vertex* terhubung didapatkan kompleksitas $O(N^2)$ untuk satu kali proses BFS, sehingga total kompleksitas yang didapat ada-lah $O(N^3)$. Dengan kompleksitas $O(N^3)$, waktu pemrosesan yang digunakan program akan melebihi batas waktu yang diharapkan ya-itu 3 detik. Saat pengujian dilakukan pada *Sphere Online Judge* menggunakan algoritma BFS biasa, waktu pemrosesan yang dida-pat adalah *"time limit exceeded"* dan memori yang digunakan ada-lah 126MB. Hal ini menunjukan bahwa program berjalan dengan waktu melebihi batas waktu yang disediakan yaitu 3 detik dan me-mori yang digunakan sudah dibawah batas memori yang diberikan yaitu 1536MB.

Untuk mengatasi hal ini, dilakukan pengurangan *vertex* yang tidak diperlukan untuk mempercepat jalannya program. *vertex* yang tidak diperlukan antara lain *vertex* yang tidak dapat dikunjungi da-

rimanapun atau tidak memiliki *degree* masuk karena sudah pasti tidak ada jalan untuk kembali ke *vertex* tersebut, *vertex* yang tidak memiliki *degree* keluar dan *vertex* yang sudah pernah dikunjungi. Dengan mengurangi *vertex* tersebut maka kasus terburuk dapat dihindari, sehingga kompleksitas yang didapat adalah $O(N^2)$.

3.2. Desain Program Utama

Subbab ini akan menjelaskan desain fungsi yang akan digunakan untuk menyelesaikan permasalahan.

3.2.1. Desain Fungsi Main

Pada fungsi ini, semua tahapan komputasi berjalan. Terma-suk pada tahapan ini yaitu pencarian *cycle* menggunakan BFS. Fung-si ini menerima parameter masukan dari pengguna, dan mengelu-arkan hasil jalan terpendek kepada pengguna. Pertama, pengguna memasukkan nilai N yang merupakan jumlah *vertex*, kemudian N baris berikutnya pengguna memasukkan nilai H_{ij} sebanyak N yang adalah *adjacency matrix*. Untuk setiap H_{ij} yang bernilai 1 sistem akan mencatat *vertex* j dan dimasukkan ke dalam *adjacency list* pa-da *vertex* i, kemudian sistem akan menambahkan nilai *out degree* dari *vertex* i dan *in degree* dari *vertex* j. Setelah proses *input* sele-sai dan *adjacency list* dari graf H telah terbentuk, maka sistem akan menjalankan fungsi BFS untuk setiap *vertex* yang ada.

Pseudocode dari desain fungsi main dapat dilihat pada Gam-bar 3.1.

3.2.2. Desain Fungsi BFS

Subbab ini akan menjabarkan desain fungsi BFS menurut penjelasan pada subbab 2.3 Strategi Penyelesaian Umum dalam ben-tuk *pseudocode*. Fungsi ini menerima masukan berupa *adjacency list*.Fungsi ini menggunakan sebuah *integer ans* untuk menampung

```
   MAIN
 1 let N = Input()
 2 memset(in_deg,0,sizeof in_deg);
 3 for i = 1 to N
 4   for j = 1 to N
 5     let h = Input()
 6if h = 1
 7let adjlist[i].push_back(j)
 8       out_deg[i]++;
 9             in_deg[j]++;
10 for i = 1 to N
11   BFS (i, j, adjlist, in_deg,out_deg)
```

Gambar 3.1: Desain Fungsi *Main*

hasil perhitungan dari fungsi ini. Keluaran fungsi ini adalah sebuah *integer ans* atau *string "NO WAY"* jika nilai *ans* adalah -1.

Fungsi BFS diawali dengan mendeklarasikan nilai *integer ans* dengan nilai -1 yang berarti belum ditemukan jalan tercepat yang merupakan keluaran fungsi BFS. Kemudian *degree* masuk pada *ver-tex* yang menjadi awalan sekaligus tujuan dicek, jika *vertex* terse-but memiliki *degree* masuk maka proses iterasi BFS dilakukan, jika *vertex* tersebut tidak memiliki *degree* masuk maka sudah dipastikan *vertex* tidak dapat dikunjungi melalui *vertex* lain, sehingga keluaran yang dihasilkan pasti *string "NO WAY"* dan tidak perlu dilakukan proses iterasi BFS.

Proses iterasi BFS diawali dengan mengosongkan semua penanda *visited* pada setiap *vertex* dan mengosongkan *queue* BFS jika terdapat sisa isi dari iterasi sebelumnya. Lalu *vertex* awal dimasukkan ke dalam *queue* BFS dipasangkan dengan jarak awal yaitu 0. Tandai *visited vertex* awal dengan nilai *true* dan penanda tujuan telah ditemukan (*found*) dengan nilai *false*. Kemudian iterasi BFS dilakukan hingga *queue* BFS kosong atau jika tujuan telah ditemu-kan.

BFS diawali dengan menyimpan nilai paling depan pada *queue* dan mengeluarkannya dari *queue*. Kemudian cek semua *vertex* yang dapat dikunjungi dari *vertex* sekarang menggunakan *adjacen-cy list* yang telah dibuat sebelumnya. Jika *vertex* yang dapat dikun-jungi merupakan *vertex* tujuan, maka tandai *found* dengan nilai *true* dan isi variabel *ans* dengan jarak sekarang+1. Jika *vertex* yang da-pat dikunjungi bukanlah *vertex* tujuan, maka dilakukan pengecek-an jika *vertex* tersebut belum pernah dikunjungi dan *vertex* tersebut memiliki *degree* keluar, masukan *vertex* tersebut ke dalam *queue* dan dipasangkan dengan jarak sekarang+1. Setelah iterasi BFS se-lesai maka keluarkan nilai *ans* jika jalan ditemukan atau *string "NO WAY"* jika tidak ditemukan jalan.

Pseudocode dari desain fungsi ini dapat dilihat pada Gambar 3.2 dan Gambar 3.3.

```
   BFS (i, j, adjlist, in_deg,out_deg)
 1 let ans = -1
 2 if in_deg != 0
 3   memset(visited,false,sizeof visited)
 4while !bfs.empty()
 5       bfs.pop()
 6     bfs.push(make_pair(i,0))
 7let visited[i] = true
 8let found = false
 9while !bfs.empty() && !found
10       store pair<int,int> now = bfs.front()
11       bfs.pop()
12     let size = adjlist[now.first].size()
13         for j = 0 to size && !found
14           if adjlist[now.first][j] = i
15               ans = now.second+1
16                 found = true
17                 break
```

Gambar 3.2: Desain Fungsi *BFS* (a)

```
18        else if !visited[adjlist[now.first][j]]
              && out_deg[adjlist[now.first][j]] != 0
19          visited[adjlist[now.first][j]] = true
20                bfs.push( make_pair( adjlist[now.first][j] ,
                      now.second+1))
21      if ans != -1
22        print ans
23      else
24        print "NO WAY"
```

Gambar 3.3: Desain Fungsi *BFS* (b)

[*Halaman ini sengaja dikosongkan*]

BAB IV
IMPLEMENTASI

4.1. Lingkungan Implementasi

Lingkungan implementasi dan pengembangan yang dilakuk-an adalah sebagai berikut.

1. Perangkat Keras
   (a) Processor AMD A10-7300 Radeon R6,10 Compute Co-res 4C+6G (4CPUs), 1.9Ghz.
   (b) Random Access Memory 4096MB
2. Perangkat Lunak
   (a) Sistem Operasi Windows 8.1 Pro 64-bit
   (b) IDE CodeBlocks 16.01
   (c) Bahasa Pemrograman C++
   (d) MinGW 64-bit

4.2. Implementasi Program Utama

Subbab ini menjelaskan implementasi program utama. Program ini merupakan program yang digunakan untuk menyelesaikan permasalahan *ADA CYCLE*.

4.2.1. Penggunaan Library, Tipe Data, dan Struct

Program ini menggunakan beberapa *library* dan tipe data se-perti yang ditunjukkan pada Kode Sumber 4.1.

Variabel-variabel yang digunakan dalam implementasi program dijelaskan pada tabel 4.1 Penggunaan Variabel (a) dan tabel 4.2 Penggunaan Variabel (b).

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int n, x, ans, size;
5 int in_deg[2000+5], out_deg[2000+5];
6 bool found, visited[2000+5];
7 vector<int> adjlist[2000+5];
8 queue< pair<int,int> > bfs;
```

Kode Sumber 4.1: *Header* Program Utama

Tabel 4.1: Penggunaan Variabel (a)

No.	Variabel	tipe data	Penjelasan
1	n	int	Menyimpan nilai N yaitu sebuah bilangan yang menyatakan jumlah *vertex*
2	x	int	Menerima masukan H_{ij} atau *adjacency matrix* untuk proses pembuatan *adjacency list*
3	ans	int	Menyimpan hasil dari pencarian jalan terpendek untuk setiap *vertex*
4	size	int	Menyimpan ukuran panjang *adjacency list* pada suatu *vertex*
5	in_deg	int[]	Menyimpan jumlah *degree* masuk pada setiap *vertex*
6	out_deg	int[]	Menyimpan jumlah *degree* keluar pada setiap *vertex*
7	found	bool	Digunakan sebagai penanda jika *vertex* tujuan sudah ditemukan
8	visited	bool[]	Digunakan sebagai penanda *vertex* telah dikunjungi

Tabel 4.2: Penggunaan Variabel (b)

No.	Variabel	tipe data	Penjelasan
9	adjlist	vector<int>[]	digunakan untuk menyimpan *adjacency list* dari graf
10	bfs	queue	Digunakan untuk pemrosesan *queue* bfs yang menampung sepasang *integer vertex* dan jaraknya.

4.2.2. Implementasi Fungsi Input

Subbab ini menjabarkan implementasi fungsi *input* untuk me-mangkas waktu pemrosesan masukan. Dalam kode sumber fung-si ini diberi nama *FastInt*. nama Penggunaan fungsi *library get-char_unlocked()* digunakan karena fungsi tersebut merupakan fung-si tercepat dalam pemrosesan masukan. Kegunaan *getchar_unlock-ed()* sendiri adalah sama dengan fungsi *getchar()* sehingga sistem menerima masukan berupa sebuah karakter dan kemudian diubah menjadi angka sebagai *return value* yang digunakan sebagai ma-sukan sesuai dengan subbab 2.1.1.

```
1 int FastInt() {
2     int res=0;
3     char c=getchar_unlocked();
4while (c>='0' && c<='9') {
5         res=res*10+((int)(c-'0'));
6         c=getchar_unlocked();
7     }
8     return res;
9 }
```

Kode Sumber 4.2: Fungsi FastInt

4.2.3. Implementasi Fungsi Main

Subbab ini menjabarkan implementasi fungsi *main* yang dijelaskan pada subbab 3.2.1. Fungsi ini mencakup pemrosesan masukan, inisiasi dan proses BFS. Proses BFS dilakukan sebanyak jumlah *vertex* yang ada sehingga kompleksitas yang didapat adalah kompleksitas BFS dikali jumlah *vertex*, yatitu $O(N^2)$. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.3 dan 4.4.

```
 1 int main(){
 2     n=FastInt();
 3     memset(in_deg,0,sizeof in_deg);
 4for (int i=0;i<n;i++) {
 5         out_deg[i]=0;
 6         for (int j=0;j<n;j++) {
 7             x=fast_int();
 8             if (x==1) {
 9                 adjlist[i].push_back(j);
10                 out_deg[i]++;
11                 in_deg[j]++;
12             }
13         }
14     }
15
16     for (int i=0;i<n;i++) {
17         ans=-1;
18         if (in_deg[i]!=0) {
19             memset(visited,false,sizeof visited);
20             while (!bfs.empty()) bfs.pop();
21             bfs.push(make_pair(i,0));
22             visited[i]=true;
23             found=false;
24             while (!bfs.empty() && !found) {
25                 pair<int,int> now=bfs.front();
                       bfs.pop();
```

Kode Sumber 4.3: Fungsi Main(a)

```
26                  size=adjlist[now.first].size();
27                  for (int j=0;j<size &&
                        !found;j++) {
28                      if (adjlist[now.first][j]==i)
                            {
29                          ans=now.second+1;
30                          found=true;
31                          break;
32                      }
33                      else if (!visited[adjlist
                            [now.first] [j]] &&
                            out_deg[adjlist
                            [now.first][j]] != 0) {
34                          visited[adjlist
                                [now.first][j]] =
                                true;
35                          bfs.push( make_pair(
                                adjlist
                                [now.first][j],
                                now.second+1) );
36                      }
37                  }
38              }
39          }
40
41          if (ans!=-1) printf("%d\n",ans);
42          else printf("NO WAY\n");
43      }
44  return 0;
45  }
```

Kode Sumber 4.4: Fungsi Main(b)

Pada Kode Sumber 4.3 baris 4 hingga 14 dilakukan proses *input* $H_{(ij)}$ yang diwakili oleh variabel *x*, pembuatan *adjacency list* dan pencatatan jumlah *degree* masuk dan *degree* keluar. Pada baris 8 hingga 12, jika *x* bernilai 1 pada i dan j tertentu, yang berarti ada jalan yang menghubungkan *vertex* i ke *vertex* j, maka *vertex* j akan ditambahkan ke *adjacency list* i, kemudian *degree* keluar pada

vertex i ditambah 1 dan *degree* masuk pada *vertex* j ditambah 1.

Pada Kode Sumber 4.3, proses inisialisasi terjadi pada baris 18 hingga 23. Pada baris 18 terjadi pengecekan jumlah *degree* ma-suk pada *vertex* yang akan dicari jalan terpendeknya. Jika *vertex* tersebut tidak memiliki *degree* masuk maka sudah dipastikan *ver-tex* tidak dapat dikunjungi melalui *vertex* lain, sehingga keluaran yang dihasilkan pasti *string "NO WAY"* dan tidak perlu dilakukan proses iterasi BFS.

Pada Kode Sumber 4.3 dan Kode Sumber 4.4, proses iterasi menggunakan queue dilakukan pada baris 24 hingga 39. Pada baris 27 dilakukan pengecekan semua *vertex* yang dapat dikunjungi dari *vertex* sekarang menggunakan *adjacency list* yang telah dibuat sebelumnya. Pada baris 28 hingga 32, jika *vertex* tujuan sudah ditemukan, maka proses BFS dinyatakan selesai dan variabel *ans* akan mencatat jumlah jarak terpendeknya. Pada baris 33 dilakukan eliminasi *vertex* dimana jika *vertex* tersebut sudah dikunjungi atau jika *vertex* tersebut tidak memiliki *degree* keluar maka tidak perlu dimasukan ke dalam *queue* BFS.

Pada Kode Sumber 4.4 baris 42 dan 43, program akan mengeluarkan hasil perhitungan jarak terpendek yang telah disimpan pada variabel *ans*. Jika variabel *ans* bernilai -1 yang berarti tidak ditemukan jalan, maka keluaran yang didapatkan adalah *string "NO WAY"*.

BAB V
UJI COBA DAN EVALUASI

5.1. Lingkungan Uji Coba

Uji coba dilakukan pada perangkat dengan spesifikasi berikut.

1. Perangkat Keras
   (a) Processor AMD A10-7300 Radeon R6,10 Compute Co-res 4C+6G (4CPUs), 1.9Ghz.
   (b) Random Access Memory 4096MB
2. Perangkat Lunak
   (a) Sistem Operasi Windows 8.1 Pro 64-bit

5.2. Skenario Uji Coba

Subbab ini akan menjelaskan hasil pengujian program penye-lesaian permasalahan *ADA CYCLE*. Ada dua metode pengujian yang akan digunakan:

1. Pengujian lokal. Pengujian ini menggunakan mesin yang di-gunakan dalam pengembangan untuk mengukur kebenaran dan informasi lain mengenai program.
2. Pengujian luar. Pengujian ini menggunakan *Online Judge* un-tuk mengukur kebenaran dan informasi lain mengenai pro-gram.

5.3. Uji Coba Kebenaran

Kebenaran metode diujikan dengan mengirim kode sumber terkait ke *Sphere Online Judge*. Umpan balik yang didapat kode sumber dengan metode BFS adalah time 1,76 memory 2,9M. Hasil uji coba kebenaran dapat dilihat pada Gambar 5.1.

Gambar 5.1: Hasil Submisi Kode Sumber ke *Sphere Online Judge*

5.4. Uji Coba Kasus Kecil

Uji Coba Kasus Kecil dilakukan pada pengujian lokal dengan memberi masukan sesuai dengan parameter masukan pada subbab 2.1.1 kedalam program menggunakan mesin yang digunakan dalam pengembangan. Uji coba ini dilakukan untuk mengecek kebenaran program secara lokal. Uji coba dinyatakan berhasil jika hasil yang dikeluarkan program sesuai dengan keluaran yang diharapkan sebagaimana dijelaskan pada subbab 2.1.3.

Masukan:

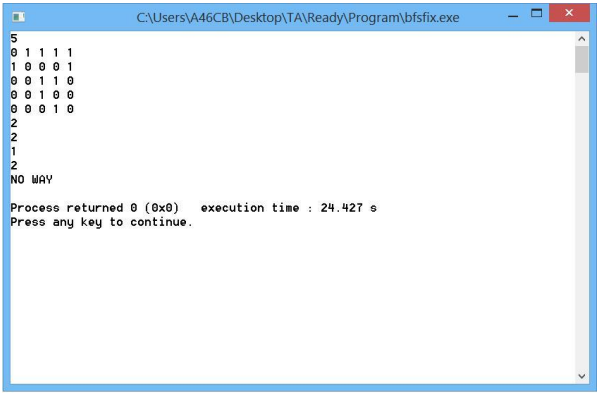
```
5
0 1 1 1 1
1 0 0 0 1
0 0 1 1 0
0 0 1 0 0
0 0 0 1 0
```

Keluaran yang diharapkan:

```
2
2
1
2
NO WAY
```

Hasil dari uji coba kasus kecil dapat dilihat pada Gambar 5.2. Hasil yang dikeluarkan program sesuai dengan keluaran yang diha-

rapkan.



Gambar 5.2: Hasil Pengujian Kasus Kecil

5.5. Uji Coba Kinerja

Subbab ini akan menjabarkan kinerja dari program penyelesaian permasalahan *ADA CYCLE* dengan metode *Breadth First Search* yang telah dioptimasi. Uji coba kinerja dibagi menjadi dua yaitu uji coba kinerja proses *input* dan uji coba kinerja algoritma.

5.5.1. Uji Coba Kinerja Proses Input

Uji coba kinerja proses *input* dilakukan dengan cara pengujian luar dengan melakukan submisi kode sumber ke *Sphere Online Judge* karena fungsi *getchar_unlocked()* tidak dapat dijalankan melalui pengujian lokal dengan mesin yang digunakan. Uji coba dilakukan dengan membuat dua kode sumber. Kode sumber pertama merupakan kode sumber program sesuai dengan implementasi program utama pada subbab 4.2. Kode sumber kedua adalah kode sumber yang sama dengan kode sumber pertama tetapi tidak

menggunakan fungsi *FastInt* sebagaimana dijelaskan pada subbab 4.2.2 melainkan menggunakan fungsi penerima masukan biasa ya-itu *scanf()*. Uji coba ini dilakukan untuk melihat perbedaan waktu yang dibutuhkan untuk memproses masukan antara fungsi *FastInt* pada subbab 4.2.2 dengan fungsi penerima masukan biasa *scanf()*.

Gambar 5.3: Hasil Submisi Kode Sumber pertama ke *Sphere Online Judge*

Gambar 5.4: Hasil Submisi Kode Sumber kedua ke *Sphere Online Judge*

Dapat dilihat pada Gambar 5.3, umpan balik yang didapat pada kode sumber pertama memakan waktu sebesar 1,76 detik, sedangkan pada Gambar 5.4, umpan balik yang didapat pada kode sumber kedua memakan waktu sebesar 4,81 detik. Sehingga penggunaan fungsi *FastInt* mempengaruhi kinerja pemrosesan masukan.

5.5.2. Uji Coba Kinerja Algoritma

Uji coba kinerja algoritma dilakukan melalui pengujian lokal dengan mengukur waktu yang dibutuhkan untuk metode *Breadth First Search* dalam menyelesaikan permasalahan *ADA CYCLE*. Pada pengujian ini, program pembuat kasus uji akan digunakan. Program pembuat kasus uji yang digunakan akan membuat nilai pada kasus uji secara acak sehingga menghasilkan graf yang acak. Kasus uji yang digunakan mengacu pada subbab 2.1.1Parameter Masukan.

Pengujian dilakukan dengan langkah berikut.

1. Rekam waktu tepat sebelum komputasi penyelesaian masalah dilakukan.

2. Lakukan komputasi terkait penyelesaian masalah
3. Rekam waktu tepat setelah komputasi penyelesaian masalah dilakukan.
4. Kurangi waktu saat selesai komputasi dengan waktu saat se-belum komputasi.
5. Ulangi untuk seluruh kasus uji dengan nilai N yang berbeda.

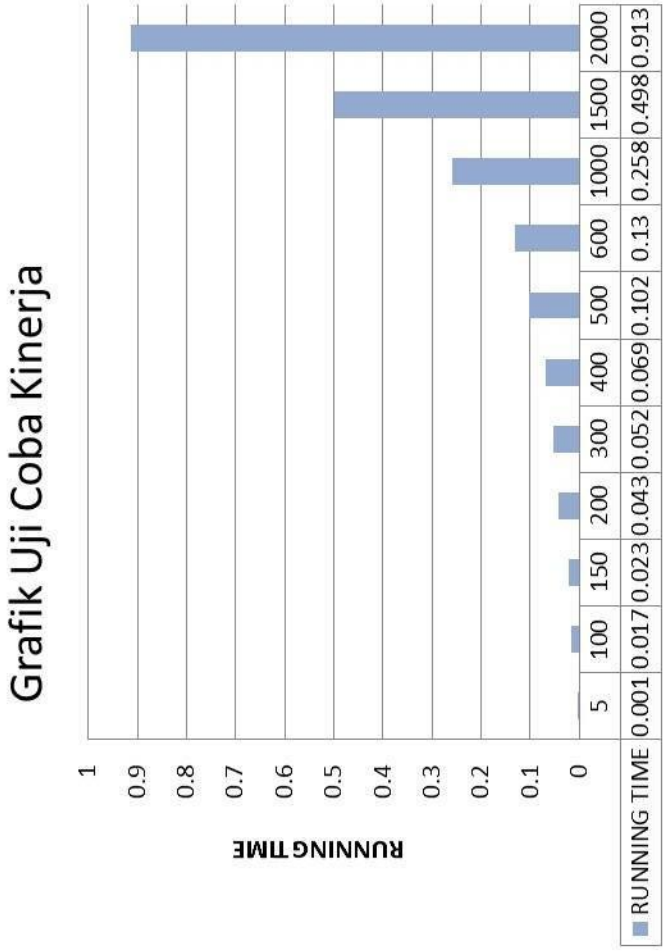
Prosedur ini dilakukan untuk memperjelas waktu yang dibutuhkan dalam menyelesaikan permasalahan. Keluaran prosedur ini yaitu lamanya waktu yang dibutuhkan untuk menyelesaikan permasalahan. Hasil uji coba kinerja program dicatat pada Tabel 5.1.

Tabel 5.1: Uji Coba Kinerja

N	Runtime(s)
5	0,001
100	0,017
150	0,023
200	0,043
300	0,052
400	0,069
500	0,102
600	0,130
1000	0,258
1500	0,498
2000	0,913

Grafik 5.5 menampilkan kinerja dari metode *Breadth First Search* yang telah dioptimasi yang digunakan pada program. Untuk setiap kenaikan N, hasil dari *running time* juga naik. Algoritma *Breadth First Search* yang telah dioptimasi berjalan dengan kom-pleksitas $O(N^2)$ sebagaimana dijelaskan pada subbab 3.1 Deskripsi Umum Sistem.

Sebagai perbandingan, Uji coba kinerja algoritma dilakukan



Gambar 5.5: Grafik Uji Coba Kinerja

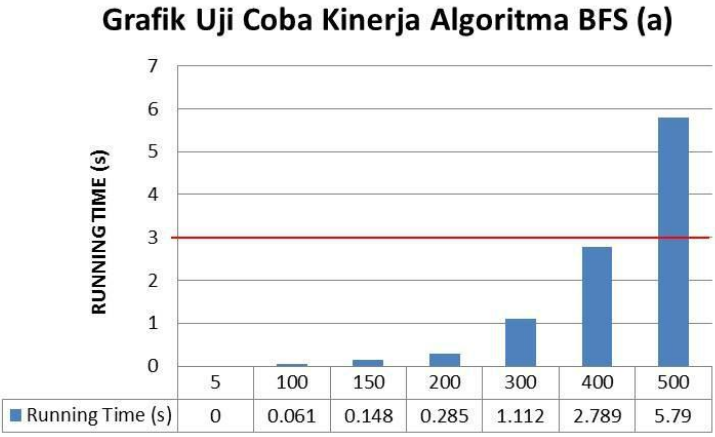
juga untuk algoritma BFS biasa. Hasil uji coba kinerja algoritma BFS biasa dicatat pada Tabel 5.2.

Tabel 5.2: Uji Coba Kinerja BFS biasa

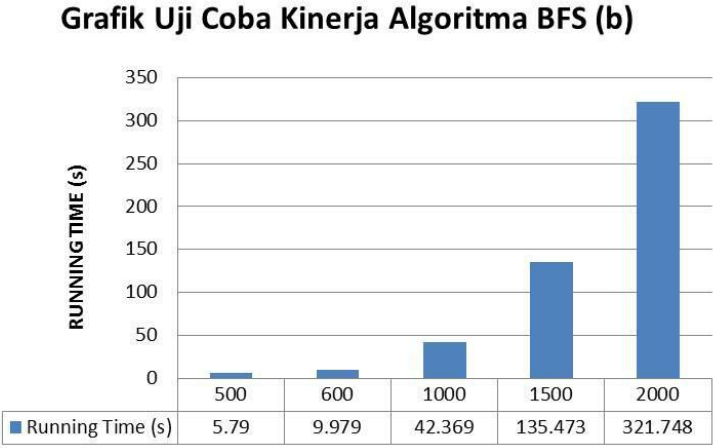
N	Runtime(s)
5	0,000
100	0,061
150	0,148
200	0,285
300	1,112
400	2,789
500	5,790
600	9,979
1000	42,369
1500	135,473
2000	321,748

Dapat diperhatikan pada Tabel 5.2, saat N=400, *running time* dari algoritma BFS biasa hampir sama dengan batas waktu 3 detik yang disediakan, yaitu 2,789 detik. Pada N=500 *running time* yang didapat adalah 5,790 detik. Pada batas *vertex* yaitu N=2000, *run-ning time* yang didapat adalah 321.748 detik, sedangkan pada pro-gram yang menggunakan BFS yang telah dioptimasi adalah 0,913 detik. Hal ini sejalan dengan kompleksitas BFS biasa pada kasus terburuk yaitu $O(N^3)$ dan kompleksitas BFS yang telah dioptimasi yaitu $O(N^2)$.

Representasi grafik uji coba kinerja BFS biasa dibagi menja-di 2 grafik karena jarak *running time* antara N terlalu besar. Grafik pada Gambar 5.6 menampilkan kinerja dari metode *Breadth First Search* biasa dari N=5 hingga N=500. Grafik pada Gambar 5.7 menampilkan kinerja dari metode *Breadth First Search* biasa dari N=500 hingga N=2000.



Gambar 5.6: Grafik Uji Coba Kinerja BFS (a)



Gambar 5.7: Grafik Uji Coba Kinerja BFS (b)

BAB VI
KESIMPULAN

Berdasarkan penjabaran di bab-bab sebelumnya, dapat disimpulkan beberapa poin terkait penyelesaian permasalahan *ADA CYCLE*.

1. Optimasi dari algoritma *Breadth First Search* (BFS) dapat di-gunakan sebagai metode untuk menyelesaikan masalah *ADA CYCLE*.
2. Umpan balik kinerja yang didapat kode sumber dengan metode *Breadth First Search* yang telah dioptimasi adalah 1,76 detik dan penggunaan memori 2,9M.
3. Kompleksitas waktu yang dibutuhkan untuk seluruh proses adalah $O(N^2)$.

[*Halaman ini sengaja dikosongkan*]

BAB VII
SARAN

Beberapa saran dapat diberikan terkait penyelesaian permasalahan *ADA CYCLE*.

1. Salah satu cara untuk mengurangi kompleksitas dari algoritma *Breadth First Search* adalah mengurangi *vertex* yang tidak diperlukan.
2. Masih terdapat metode lain untuk menyelesaikan permasalahan *ADA CYCLE* untuk dikembangkan.

[*Halaman ini sengaja dikosongkan*]

DAFTAR PUSTAKA

[1] Morass. (2018). ADACYCLE - Ada and Cycle, [Online]. Ava-ilable: https://www.spoj.com/problems/ADACYCLE/.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithm*, 3rd ed. 2009.

[3] G. F. Geeks. (2018). Graph Data Structure And Algorithms, [Online]. Available: https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/.

[4] E. W. Weisstein. (2018). Vertex Degree, [Online]. Available: http://mathworld.wolfram.com/VertexDegree.html.

[5] ——, (2018). Indegree, [Online]. Available: http://mathworld. wolfram.com/Indegree.html.

[6] ——, (2018). Outdegree, [Online]. Available: http://mathworld. wolfram.com/Outdegree.html.

[7] ——, (2018). Graph Loop, [Online]. Available: http://mathworld. wolfram.com/GraphLoop.html.

[8] ——, (2018). Graph Path, [Online]. Available: http://mathworld. wolfram.com/GraphPath.html.

[9] ——, (2018). Path Graph, [Online]. Available: http://mathworld. wolfram.com/PathGraph.html.

[10] ——, (2018). Graph Cycle, [Online]. Available: http://mathworld. wolfram.com/GraphCycle.html.

[11] G. F. Geeks. (2018). Graph and Its Representation, [Online]. Available: https://www.geeksforgeeks.org/graph-and-its-representations/.

[12] E. W. Weisstein. (2018). Eulerian Cycle, [Online]. Available: http://mathworld.wolfram.com/EulerianCycle.html.

[13] G. F. Geeks. (2018). Breadth First Search or BFS for a Graph, [Online]. Available: https : / / www . geeksforgeeks . org / breadth-first-search-or-bfs-for-a-graph/.

BIODATA PENULIS

Penulis bernama Christyanto Liman, lahir pada tanggal 20 Desember 1996 di Cianjur. Penulis telah mengenyam pendidikan di Sekolah Dasar Nasional Kutoarjo pada ta-hun 2002 hingga 2008, Sekolah Menengah Pertama Negeri 3 Purworejo pada tahun 2008 hingga 2011, dan Sekolah Menengah Atas Negeri 1 Purworejo pada tahun 2011 hingga 2014. Pada masa penulisan, penulis sedang menempuh masa studi S1 di Institut Teknologi Sepuluh Nopember, Surabaya di Departemen Informatika.

Selama masa studi, penulis memiliki ketertarikan yang dalam membuat aplikasi *game*, dan algoritma pemrograman. Di luar kesibukan akademik, penulis cukup aktif di organisasi *ITS Foreign Language Society* sebagai salah satu staf. Penulis juga berkontribu-si dalam berbagai kepanitiaan, baik dalam skala kecil (yaitu dalam kampus) maupun skala nasional.