PENENTUAN LINTASAN TERPENDEK UNTUK POSTMAN PROBLEM DENGAN ALGORITMA EDMONDS DAN JOHNSON

Disusun oleh :

AGUS ARI PRAMANA

2690100043

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA

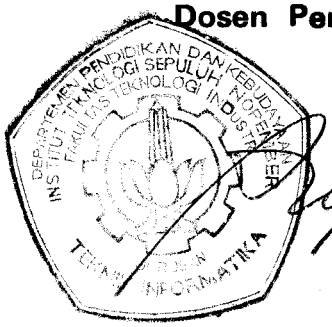
1997

PENENTUAN LINTASAN TERPENDEK UNTUK POSTMAN PROBLEM DENGAN ALGORITMA EDMONDS DAN JOHNSON

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan
Untuk Memperoleh Gelar Sarjana Teknik Informatika
Pada
Jurusan Teknik Informatika
Fakultas Teknologi Industri
Institut Teknologi Sepuluh Nopember
S u r a b a y a

Mengetahui / Menyetujui
Dosen Pembimbing



Dr. Ir. SUPENO DJANALI
NIP. 130 368 610

S U R A B A Y A
Maret, 1997

karya ini kupersembahkan kepada

Bapak dan Ibu, kedua orang tuaku tercinta

Nona tercinta, Mbok Tu, Mbok Tut, Bli De, Ayu, Ketut, saudara-saudaraku

'hanya Hyang Widhi yang bisa membalas kasih sayang mereka'

ABSTRAK

Dalam Tugas Akhir ini akan dibuat perangkat lunak pada salah satu aspek riset operasional yaitu menentukan lintasan terpendek dari *Postman Problem* dengan menggunakan algoritma pendekatan dari perpaduan antara algoritma Edmonds dan Johnson dengan kebalikan dari algoritma tadi

Algoritma Edmonds dan Johnson menyatakan bahwa setiap vertek dalam sebuah graf harus mempunyai jumlah *degree* yang genap untuk graf tidak berarah. Dan untuk graf yang berarah setiap vertek harus mempunyai *degree* yang genap, serta jumlah *indegree* sama dengan *outdegree*. Jadi algoritma ini terdiri dari tiga langkah yaitu, langkah pertama adalah mengubah graf asal sehingga setiap vertek mempunyai *degree* genap, dimana setiap arc dianggap sebagai edge. Langkah kedua membentuk *indegree* sama dengan *outdegree*. Langkah ketiga mengembalikan keadaan vertek mempunyai *degree* genap sambil mempertahankan *indegree* sama dengan *outdegree*.

Algoritma yang kedua adalah kebalikan dari algoritma di atas, yaitu langkah pertama yang dilakukan adalah langkah kedua dari algoritma Edmonds dan Johnson. Langkah kedua yang dilakukan adalah langkah pertama dari algoritma Edmond dan Johnson sambil mempertahankan *indegree* sama dengan *outdegree*.

Setelah kedua algoritma di atas digunakan untuk menyelesaikan suatu *Postman Problem*, maka dipilih penyelesaian yang paling baik.

ABSTACT

In this minithesis a software will be made on one of the operational researc aspect i. e. to determine the shortest path by using an approximation algorithms of the combination between Edmonds and Johnson algorithm with the revers of that algorithm.

Edmonds and Johnson algorithm states that each vertex in a graph must have even number of degree for an undirected graph, while for a directed graph, each vertex must have an even degree and the same of indegree and outdegree. So, this algorithm consists of three steps as follow, the first step is to change the original graph so that every vertex has an even degree, where every arc is considered as an edge. The second step is to equalize the indegree with the outdegree, and the third step is to restore the condition of the vertex with even degree while maintaining the equality between the indegree and the outdegree.

The second algorithm is the reverse of the above algorithm, that is, its first step is the second step of the Edmonds and Johnson algorithm, while its second step is the first step of Edmonds and Johnson algorithm, and at the same time the equality of the indegree and outdegree is maintained.

After both of the above algorithms are utilized to solve a Postman Problem, a best solution will be selected then.

KATA PENGANTAR

OM SWASTIASTU

Segala puji dan syukur penulis panjatkan pada Ida Sang Hyang Widhi Wasa, Tuhan Yang Maha Esa atas Asung Kertha Wara Nugraha, bimbingan dan anugrah yang tak terkira dari Nya, sehingga Tugas Akhir penulis dapat terselesaikan dengan baik.

Judul Tugas Akhir ini :

PENENTUAN LINTASAN TERPENDEK
UNTUK POSTMAN PROBLEM
DENGAN ALGORITMA EDMONDS DAN JOHNSON

yang merupakan pemenuhan salah satu persyaratan untuk menyelesaikan pendidikan dalam meraih gelar sarjana strata satu (S1) di Jurusan Teknik Informatika Fakultas Teknologi Industri Institut Teknologi Sepuluh Nopember Surabaya.

Penulis berharap agar Tugas Akhir ini dapat memberikan banyak manfaat dan tambahan pengetahuan bagi para pembaca dan pemerhati umumnya dan mahasiswa Teknik Informatika ITS khususnya.

Tidak lupa penulis mengucapkan banyak terima kasih kepada bapak Dr. Ir. Supeno Djanali M.Sc yang telah meluangkan waktunya untuk memberikan bimbingan sehingga Tugas Akhir ini dapat terselesaikan dengan baik.

OM SANTI, SANTI, SANTI OM

Surabaya, Januari 1997

Penulis

UCAPAN TERIMA KASIH

Pada kesempatan ini penulis mengucapkan terima kasih atas bantuan yang tak ternilai harganya kepada :

1. **Bapak dan Ibu** yang telah memberikan bimbingan moril dan materi serta kasih sayang, harapan dan kepercayaan yang tulus kepada putra-putranya.
2. **Mbok Tu, Bli De, Mbok Tut Mila, Adik Ayu, Adik Ketut** dan saudara-saudaraku yang selalu memacu semangatku.
3. **Dr. Ir. Supeno Djanali**, selaku dosen pembimbing.
4. **Dr. Ir. Arif Djunaidy**, selaku Ketua Jurusan Teknik Informatika.
5. **Drs. Dr. Ir. Riyanarto**, selaku dosen wali.
6. **Ketut Palawindu, Yosep**, yang memberi petunjuk kepada penulis.
7. **Ibu dan Bapak Ida Bagus Yasa** beserta keluarga, yang telah memberikan pondokan selama kuliah ini.
8. Teman-teman dan juga sekaligus saudaraku di rumah-Wisma Permai X/2 yang telah memberikan dukungan yang sangat saya butuhkan.
9. **Della, Malik, Didik, Blio, Kenyul, Atok**, dan **Suaja** serta teman-teman alumni SMAN 4 lainya.
10. **Irfan, Raharjo, Irwan, Syaiful, Andi** serta rekan-rekan C-06 lainnya.

DAFTAR ISI

Halaman

DAFTAR GAMBAR

Halaman

BAB I

PENDAHULUAN

1.1. Latar Belakang

Teori graf merupakan salah satu bidang pengetahuan yang perkembangannya sangat pesat dewasa ini setelah ditemukannya **Teknologi Komputer**. Banyak permasalahan di lingkungan sehari - hari ataupun permasalahan ilmiah yang memanfaatkan graf seperti peta rute, jaringan listrik, molekul kimia, pohon silsilah, dan lain sebagainya.

Perangkat keras yang semakin murah dan cepat juga mendukung pengembangan perangkat lunak, pengolahan graf yang pada umumnya digunakan untuk mensimulasikan obyek permasalahan sebelum akhirnya diterapkan sehingga memperoleh hasil yang optimum.

Salah satu permasalahan yang memanfaatkan graf adalah menentukan lintasan terpendek, yang bertujuan untuk meminimumkan biaya, sehingga sumber-sumber daya yang ada dapat dialokasikan dengan efisien dan menguntungkan. Permasalahan yang akan dibahas di sini adalah mendapatkan lintasan terpendek dari *Postman Problem.* Untuk menentukan lintasan terpendek maka diperlukan algoritma tertentu sehingga diharapkan penyelesaian yang didapat adalah optimum.

Satu algoritma terkadang tidak dapat menentukan penyelesaian yang optimal dari Postman Problem, atau tidak mungkin menentukan hasil yang maksimum dari

permasalahan yang ada, sehingga hasil yang didapat adalah merupakan pendekatan dari penyelesaian yang sebenarnya. Untuk itu algoritma-algoritma yang ada sedapat mungkin dipadukan untuk memperoleh penyelesaian yang memuaskan. Adapun algoritma tersebut adalah algoritma dari **Edmonds dan Johnson** yang memadukan dua algoritma yang prinsip kerjanya saling berkebalikan.

1.2. Tujuan

Pada Tugas Akhir ini perangkat lunak yang dirancang dan dikembangkan adalah perangkat lunak yang berguna untuk membantu menyelesaikan permasalahan tukang pos atau *postman problem*, dengan menggunakan prosedur lintasan terpendek atau *shortest path*, rute perjalanan dari *postman problem* dapat dicari yang terpendek.

Untuk tahap awal dicari dulu setiap vertek yang berdegree ganjil, kemudian dengan procedure *shortest path* vertek-vertek ini dijadikan ber*degree* genap. Bila terdapat graf berarah, *indegree* dan *outdegree*nya disamakan. Karena dengan menyamakan *inoutdegree*nya vertek kembali memiliki *degree* ganjil, untuk mengembalikan ke keadaan semula digunakan **prosedur Evenparity.**

Hasilnya adalah semua rute perjalanan yang harus ditempuh dari *postman problem* dengan jarak tempuh yang terpendek.

1.3. Permasalahan

Pembuatan perangkat lunak untuk *postman problem* sangatlah bermanfaat dan membantu tukang pos untuk dapat memilih rute terpendek yang harus ditempuh untuk penyampaikan semua surat yang beralamat di lokasi wilayahnya. Bukan cuma tukang pos, tapi semua biro jasa pengiriman bisa memanfaatkan program ini.

Dalam penyelesaian perangkat lunak dari *postman problem* banyak sekali melibatkan operasi matematika penjumlahan dan pengurangan. Walaupun tidak rumit, tapi memerlukan ketelitian.

1.4. Batasan

Untuk menyelesaikan *postman problem* digunakan algoritma Edmond And Johnson. Penggunaan algoritma ini didasarkan pada tahapan-tahapan prosedurnya yang jelas, sehingga memudahkan dalam mengimplementasikan ke dalam rancangan perangkat lunak.

Input data dari program ini berupa *mixed* graf yang mewakili peta rute perjalanan dari postman problem. Karena dalam perangkat lunak ini input diolah dalam bentuk matrik bujursangkar dimana penyimpanannya menggunakan array, maka indek elemen dibatasi maksimum 50, indek elemen ini mewakili jumlah vertek.

*Output*nya adalah berupa *mixed* graf yang menampilkan semua lintasan yang harus dilalui oleh tukang pos dengan total jarak tempuh yang terpendek.

1.5. Metode Penelitian

Pembuatan Tugas Akhir ini disusun dengan menerapkan langkah-langkah Metode Penelitian sesuai dengan kriteria penyusunan karya ilmiah yang meluputi :

1.5.1. Pengumpulan Data (*study literatur*)

1. Mengumpulkan *script*, makalah ataupun jurnal yang berhubungan dengan topik
2. Mengumpulkan buku-buku penunjang
3. Mempelajari semua *scrip*, makalah, jurnal, dan buku - buku yang telah dikumpulkan di atas

1.5.2. Perancangan Sistem

1. Mempelajari sistem
2. Merancang (*desain*) sistem
3. Menganalisa sistem
4. Menguji (*testing*) sistem

1.5.3. Pengembangan Perangkat Lunak

1. Implementasi algoritma ke dalam prosedur program
2. Penggabungan prosedur-prosedur sesuai dengan rancangan sistem

1.5.4. Analisa Perangkat Lunak

1. Analisa dan uji coba perangkat lunak
2. Perbaikan perangkat lunak

1.5.5. Penyusunan Buku Tugas Akhir

1.6. Sistematika Pembahasan

Sistematika pembahasan dalam Tugas Akhir ini dapat diuraikan sebagai berikut :

1. **BAB I** : Pendahuluan, yang meliputi latar belakang, permasalahan, tujuan, batasan masalah, metodelogi penelitian, sistematika pembahasan.
2. **BAB II** : Teori Dasar Tentang Graf, yang meliputi pengertian graf dan digraf, graf dan digraf Euler, graf planar, dan *path* terpendek.
3. **BAB III** : Penjelasan Algoritma Edmonds and Johnson meliputi algoritma Edmonds and Johnson untuk *mixed postman*, algoritma kedua untuk *mixed postman*, *mixed strategy* algoritma untuk *postman problem*, algoritma approksimasi untuk *planar mixed postman.*
4. **BAB IV** : Perancangan dan Pengembangan Perangkat Lunak, yang meliputi rancangan struktur data, diagram *flowchart* dari proses, dan rancangan prosedural perangkat lunak.
5. **BAB V** : Analisa Hasil Uji Coba dan Pembahasan meliputi :pengoperasian perangkat lunak, evaluasi algoritma Evendegree, evaluasi algoritma Inoutdegree, evaluasi algoritma Evenparity, evaluasi algoritma Largecycle, Analisa Hasil dan terakhir adalah kemungkinan pengembangan-pengmbangan perangkat lunak lebih jauh.
6. **BAB VI** : Penutup yang meliputi kesimpulan yang diperoleh dari Tugas Akhir ini dan beberapa saran dari penulis.

BAB II

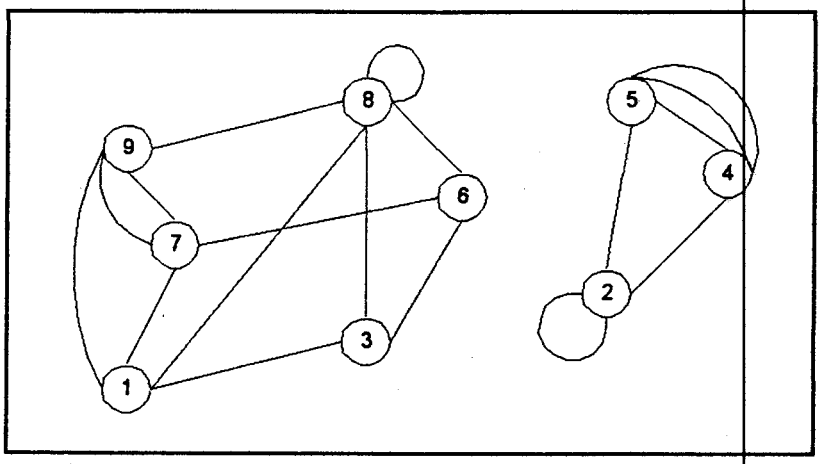
GRAF

2.1. Perngertian Graf dan Digraf

Saat ini teori graf merupakan suatu ilmu yang populer dan perkembangannya sangat pesat apalagi dengan adanya komputer. Salah satu alasan yang paling penting pesatnya perkembangan teori graf ini adalah banyak aplikasi permasalahan yang ada pada masyarakat modern dapat diselesaikan dengan menggunakan teori graf seperti pada ilmu ekonomi, ilmu manajemen, *energy modeling*, peta perjalanan, *marketing*, dan masih banyak permasalahan lainnya.

Sebuah **graf** $G = (V, E)$ terdiri dari V yang disebut dengan himpunan **vertek** dan E yang disebut dengan himpunan **edge** seperti e adalah menghubungkan sepasang vertek v dan w. Dapat juga ditulis dengan $e = \{v, w\}$ atau $\{w, v\}$dan sering disebutkan sebagai berikut:

1. e adalah sebuah edge antara v dan w
2. e adalah **incident** pada kedua v dan w
3. e menghubungkan vertek v dan w.



gambar 2.1

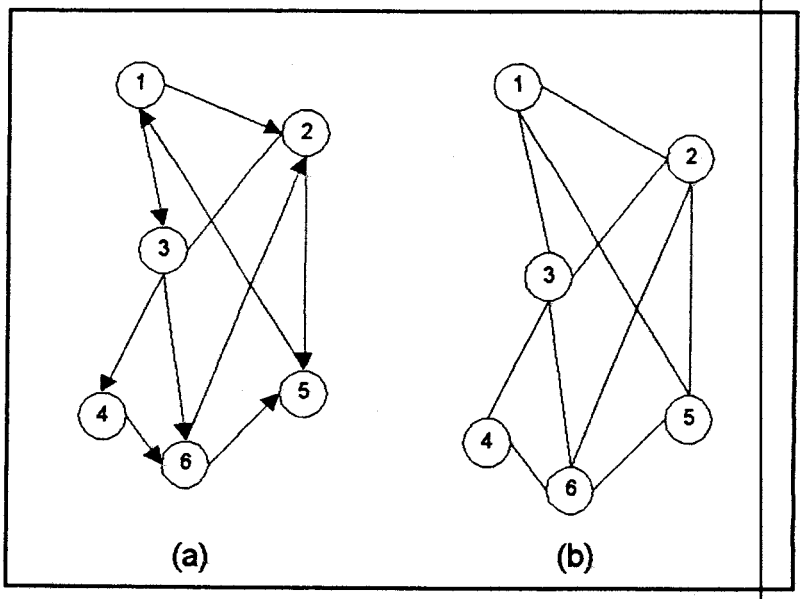
Contoh sebuah multigraf

Pada pembahasan ini kedua *v* dan *w* adalah **adjacent vertek** dan **incident** pada *e*. Sebuah edge yang menghubungkan hanya sebuah vertek disebut **loop**. Jika ada lebih dari satu edge menghubungkan pasangan-pasangan vertek di dalam graf maka graf ini disebut dengan **multigraf**. Jika ada dua atau lebih edge yang menghubungkan sepasang vertek di dalam multigraf maka edge ini disebut dengan **multiple edge**. Dalam menggambar sebuah graf, vertek digambarkan dengan lingkaran kecil yang di dalamnya berisi nama ataupun nomor dari vertek tersebut. Sedangkan edge antara dua vertek digambarkan dengan sebuah garis lurus atau sebuah kurva yang menghubungkan dua lingkaran kecil yang merupakan gambaran dari vertek. Seperti pada gambar 2.1 adalah sebuah multigraf yang mana verteknya adalah $V = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ dengan *loop* pada vertek 2 dan pada vertek 8. Juga terdapat dua edge yang menghubungkan

vertek 7 dan 9 dan tiga edge yang menghubungkan vertek 4 dan 5. Sebuah graf dikatakan **graf sederhana** jika graf tersebut tidak mempunyai *loop* ataupun *multiple* edge. Jika semua nomor pada vertek dari sebuah graf G terhubung oleh setiap edge maka G adalah sebuah **network atau weighted graf.**

Sebuah *directed* graf atau digraf $G = (V, A)$ dimana V adalah kumpulan dari **vertek** dan A adalah kumpulan dari **arc** seperti pada setiap a di dalam E adalah gabungan dari sebuah pasangan berarah dari vertek v dan w. Pada penulisan $a = (v, w)$ dapat dinyatakan dengan beberapa pengertian seperti :

1. a adalah sebuah arc dari v ke w
2. vertek v adalah **adjacent ke** vertek w
3. vertek w adalah **adjacent dari** vertek v
4. arc a adalah **incident dari** v
5. arc a adalah **incident ke** w.



gambar 2.2

(a) Contoh sebuah digraf. (b) Contoh underlying graf

Dua vertek adalah **adjacent** jika ada sebuah arc menghubungkan satu vertek ke vertek yang lainnya. Jika semua nomor pada vertek dari sebuah graf terhubung oleh setiap arc maka graf tersebut adalah sebuah **directed network** atau **weighted digraf.** Jika setiap arc pada graf dihilangkan menjadi edge maka disebut dengan **underlying graf**[1] dari digraf. Untuk menjelaskan secara gambar dari digraf sebuah arc dari vertek *v* ke *w* adalah seperti sebuah garis yang memiliki panah menghadap ke *w*. Pada gambar 2.2(*a*) digambarkan sebuah digraf dan gambar 2. 2.(*b*) adalah yang disebut dengan underlying graf. Dalam sebuah **mixed graf** terdapat paling sedikit satu buah elemen arc dan terdapat paling sedikit satu buah elemen edge.

---

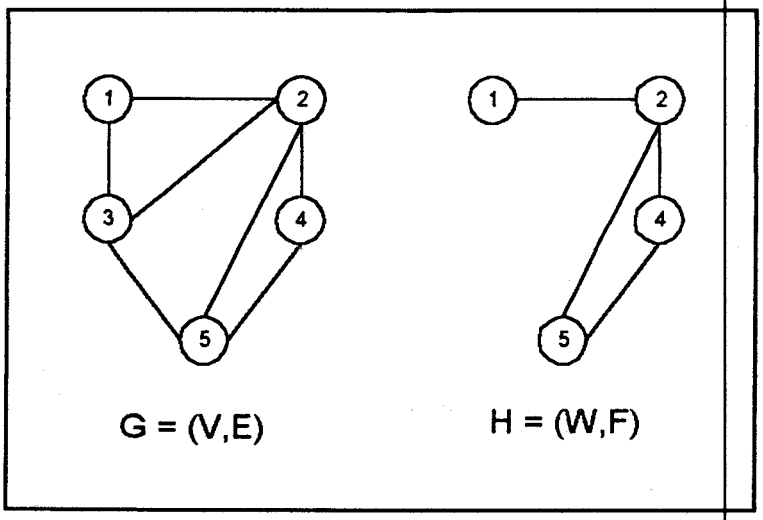
1 V. K. Balakrishnan, *Introductory Discrete Mathematics*, hal. 122

Sebuah graf sederhana dengan n vertek disebut dengan **graf lengkap** jika pada graf tersebut terdapat sebuah edge antara setiap pasangan dari vertek. Graf ini dituliskan dengan K_n. Sebuah digraf adalah digraf lengkap apabila underlying graf dari digrap tersebut adalah **lengkap**.

Sebuah graf $G' = (V', E')$ adalah **subgraf** dari $G = (V, E)$ jika V' adalah *subset* dari V dan E' adalah subset dari E. Jika setiap W adalah *subset* dari V, subgraf dari G yang mengandung W adalah graf $H = (W, F)$ dimana f adalah sebuah edge di dalam F jika $f = \{u, v\}$, dimana f adalah di dalam E dan kedua u dan v adalah di dalam W. Pada contoh gambar 2.3 dimana $W = \{1, 2, 4, 5\}$ adalah *subset* dari himpunan vertek V pada graf G dan subgraf dari G yang mengandung W adalah graf H. Sebuah subgraf lengkap dari G disebut dengan **clique**[2] di dalam G.

---

[2] V. K. Balakrishnan, *Introductory Discrete Mathematics*, hal. 123

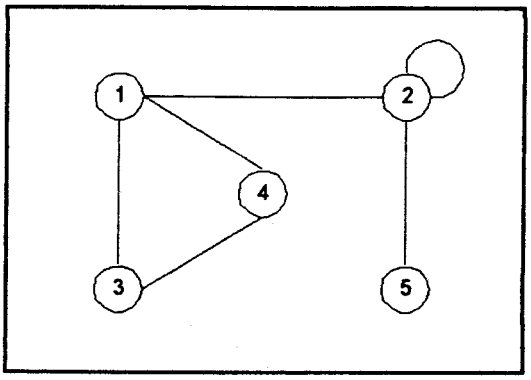


gambar 2.3

Contoh sebuah subgraf

2.1.1. Adjacency Matrices dan Incidence Matrices

Untuk tujuan menginputkan graf ke dalam komputer *adjacency matrices* dan *incidence matrices* diperlukan dalam membuat graf tanpa perlu menggambarkannya kembali. Selain itu dengan membuat gambar tidaklah praktis ketika graf tersebut memiliki jumlah vertek dan edge yang besar. Pada pembahasan ini akan dijelaskan bagaimana cara menggambarkan graf atau digraf tanpa membuat gambar graf. Cara terbaik untuk menginputkan graf yaitu dengan menggunakan sifat dan akibat dari graf tersebut. Selanjutnya, perlu dipertimbangkan pemilihan algoritma graf yang efisien untuk menggambarkan graf.



gambar 2.4

Graf

Graf $G = (V, E)$ adalah sebuah graf tanpa multiple edge dimana $V = \{1, 2, 3,..., n\}$. **Adjacency matrix** dari G adalah sebesar n x n sedang nilai matrik $A = (a_{ij})$, dimana $a_{ij} = 1$ jika terdapat edge antara vertek i dan vertek j dan $a_{ij} = 0$ jika tidak terdapat edge anatara vertek i dan vertek j. *Adjacency matrix* dari sebuah graf adalah simetris. Elemen diagonal dari matrik adalah nol jika dan hanya jika tidak terdapat loop pada graf. *Adjacency matrix* dari graf pada gambar 2.4 ditunjukkan seperti matrik A, dimana

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

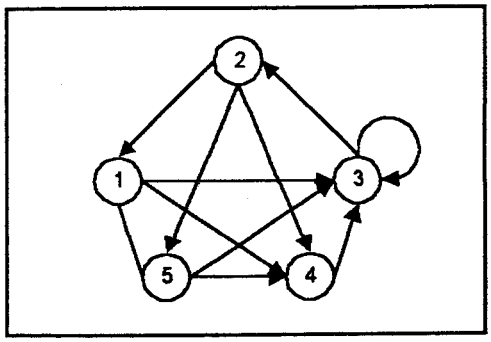
Degree dari sebuah vertek di dalam sebuah graf adalah jumlah dari edge yang *incident* pada vertek tersebut. Sebuah vertek dikatakan **ganjil** jika *degree* dari vertek tersebut adalah ganjil, begitu juga kebalikannya, sebuah vertek dikatakan **genap** jika

degree dari vertek tersebut adalah genap. Pada gambar 2.4 vertek 1, 2, dan 5 adalah ganjil, dengan *degree* berturut-turut 3, 3, dan 1. Dengan jelas dinyatakan bahwa jumlah elemen bukan nol dalam baris *i* dari *adjacency matrix* dari sebuah graf adalah *degree* dari vertek *i*, yang mana jumlahnya sama dengan semua elemen yang ada pada baris *i* atau colom *i*.

Adjacency matrix dari digraf dengan *n* vertek adalah merupakan perkalian *n* x *n* dari matrik $A = (a_{ij})$, dimana $a_{ij} = 1$ jika terdapat arc dari *i* ke *j* dan begitu juga kebalikannya. *Adjacency matrix* dari digraf pada gambar 2.5 adalah *A*, dimana

$$A = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Catatan bahwa *adjacency matrix* dari sebuah digraf tidak perlu simetris.



gambar 2.5

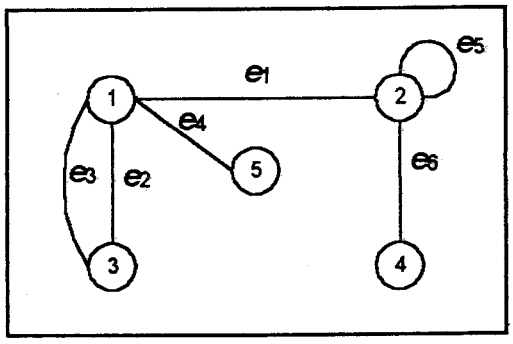
Digraf

Outdegree dari vertek dalam sebuah digraf adalah jumlah dari arc yang *incident* dari vertek tersebut dan **indegree** dari vertek adalah jumlah arc yang *incident* ke vertek tersebut. Pada gambar 2.5, *outdegree* ditunjukkan oleh vertek 2 adalah 3 dan *indegree* pada vertek 2 adalah 1. Catatan bahwa jumlah dari elemen pada baris *i* menunjukkan *outdegree* dari *i* dan jumlah dari elemen pada colom *j* menunjukkan indegree dari *j*. Seperti pada gambar 2.5, bahwa *outdegree* dari vertek 3 adalah 2 dan *indegree* adalah 4.

Matrik lain yang juga berguna untuk manampilkan graf pada komputer adalah *incidency matrix*. Tidak seperti *adjacency matrix*, *incidency matrix* mampu manggambarkan *multiple* edge dan arc paralel. Seperti $G = (V, E)$ adalah sebuah graf dimana $V = \{1, 2, \ldots, n\}$ dan $E = \{e_1, e_2, \ldots, e_m\}$. **Incidency matrix** dari G adalah n x m pada graf $B = (b_{ik})$, dimana setiap baris menunjukkan setiap vertek dan setiap kolom menunjukkan setiap edge seperti jika e_k adalah sebuah edge antara i dan j, maka

semua elemen dari kolom k adalah 0 kecuali $b_{ik} = b_{jk} = 1$. Untuk contoh dari *incidency matrix* dari graf pada gambar 2.6 adalah matrik B, dimana

$$B = \begin{array}{c} \begin{matrix} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \end{matrix} \\ \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

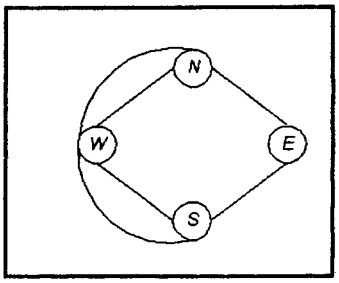


gambar 2.6

Graf dengan loop

Catatan bahwa kolom yang menunjukkan edge memiliki tepat dua elemen yang tidak nol jika tidak merupakan sebuah *loop* dan memiliki tepat satu elemen tidak nol jika merupakan sebuah *loop*. Selanjutnya jumlah elemen yang ada pada baris i merupakan *degree* dari vertek i.

Setealah diteliti bahwa di dalam baberapa graf yang tidak memiliki loop jumlah dari degree pada semua vertek adalah dua kali jumlah dari edge dengan setiap edge dihitung dua kali. Untuk contoh adalah pada gambar 2.7 dimana terlihat bahwa deg N + deg S + deg W + deg E = 3 + 3 + 5 + 3 = 14 dua kali jumlah dari edge.

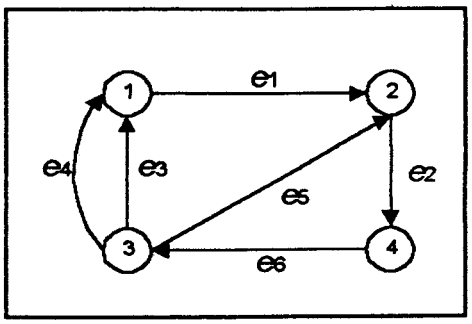


gambar 2.7

Graf tanpa loop

Incidency matrix B dari sebuah digraf (tanpa ada loop) didefinisikan sebagai berikut : jika e_k adalah arc dari i ke j, semua elemen di dalam kolom k adalah nol kecuali b_{ik} = -1 dan b_{ik} = 1. Sebagai contoh adalah graf pada gambar 2.8 yang digambarakan pada *incidency matrix B*, yaitu

$$B = \begin{array}{c} \begin{matrix} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \end{matrix} \\ \begin{bmatrix} -1 & 0 & 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & -1 & -1 & 1 \\ 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix} \end{array}$$



gambar 2.8

Digraf tanpa loop

Catatan bahwa jumlah dari semua elemen di dalam baris *i pada incidency matrix* dari digraf adalah sama dengan *indegree* dari *i* dikurang *outdegree* dari *i*. Setelah diteliti bahwa **beberapa digraf jumlah dari semua outdegree sama dengan jumlah total dari arc, demikian juga sama dengan jumlah semua indegree.** Ini karena ketika *outdegree* dijumlah setiap arc dihitung satu ketika setiap arc adalah *incident* dari satu vertek. Demikian juga ketika *indegree* dijumlah setiap arc dihitung satu ketika *incident* ke satu vertek.

2.1.2. Menggabungkan Graf

Sebuah path antara dua vertek v_1 dan v_r di dalam graf adalah rangkaian dari vertek dan edge dengan bentuk $v_1, e_1, v_2, e_2, v_3, e_3, \ldots, v_r, e_r$ dimana e_k adalah sebuah edge antara v_k dan v_{k+1}. Secara umum vertek dan edge di dalam path tidak perlu berbeda.

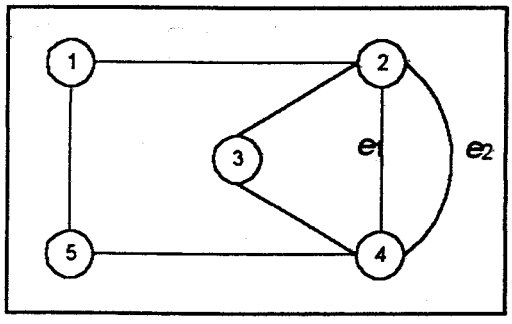
Sebuah path adalah dikatakan **sederhana** jika verteknya adalah berbeda. Di dalam path sederhana jelas dikatakan bahwa semua edge adalah berbeda. Tetapi

sebuah path dengan edge berbeda dapat mempunyai vertek yang berulang-ulang. Sebuah graf dikatakan **terhubung** jika terdapat sebuah path antara pasangan vertek di dalam graf.

Sebuah path antara sebuah vertek dan berakhir pada vertek itu sendiri disebut dengan **path tertutup**. Path tertutup pada semua edge yang berbeda disebut dengan **circuit**. Sebuah circuit dengan semua vertek yang berbeda disebut dengan **cycle**. Catatan bahwa v, e_1, w, e_2, v adalah sebuah *cycle* tetapi v, e, w, e, v bukan sebuah *circuit*, maka dari itu bukan disebut dengan *cycle*. Dua buah path tertutup dijelaskan seperti di bawah ini

$$v \overset{e_1}{\text{- - - -}} w \overset{e_2}{\text{- - - -}} v \text{ dan } v \overset{e}{\text{- - - - -}} w \overset{e}{\text{- - - - -}} v$$

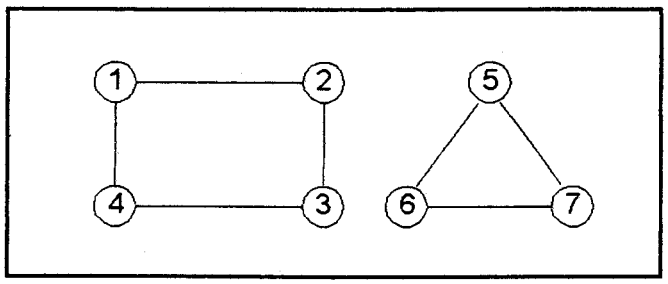
Pada gambar 2.9, 1 - - - - 2 - - - - 3 - - - - 2 - - - - 1 - - - - 5 adalah sebuah path dan 1 - - - - 2 - - - - 3 - - - - 4 - - - - 5 adalah path sederhana, dan dapat dilihat bahwa $2 \overset{e_1}{\text{- -}} 3$ - - - - 4 - - - - 5 - - - - 1 - - - - $2 \overset{e_1}{\text{- -}} 4 \overset{e_2}{\text{- -}} 2$ adalah sebuah circuit dan 2 - - - - 4 - - - - 3 - - - - 2 adalah sebuah cycle.



gambar 2.9

Graf

Jika *v* dan *w* adalah terhubung maka *w* dan *v* terhubung. Dalam kenyataannya ralasi *J* ditentukan oleh *vJw* jika *v* dan *w* yang terhubung adalah merupakan relasi yang bersifat *equivalen* yang memisahkan himpunan *V* dari vertek menjadi himpunan bagian *V* yang pasangan verteknya tidak terhubung. Subgraf yang terdiri dari himpunan bagian yang terhubung secara maksimal maka subgraf ini disebut dengan **komponen** dari graf. Jumlah dari komponen sebuah graf *G* dilambangkan dengan *K*(*G*) dan bernilai sama dengan 1 jika dan hanya jika *G* adalah terhubung. Seperti graf pada grambar 2.10 mempunyai dua komponen *G'* dan *G''*, dimana *G'* terdiri dari himpunan bagian {1, 2, 3, 4} dan *G''* terdiri dari himpunan bagian {5, 6, 7}.



gambar 2.10

Graf G

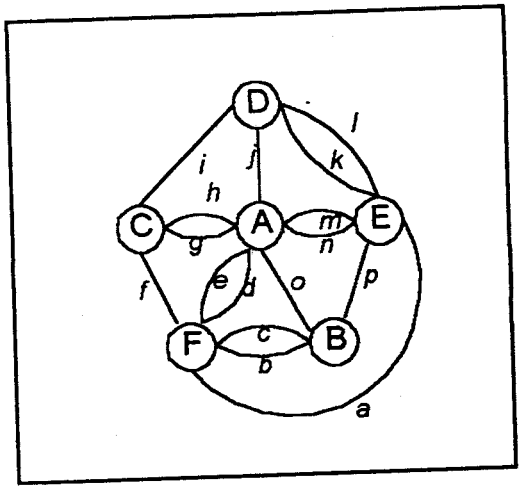
Terdapat sebuah ralasi yang menarik dan bermanfaat dari jumlah path antara pasangan dari vertek di dalam *G* dan elemen dari *adjacency matrix*. Sebuah path dengan *k* edge disebut dengan *k*-path. 1-path adalah sebuah edge. Di dalam *adjacency*

matrix n x n dari graf dengan vertek $V = \{1, 2, \ldots, n\}$, nilai dari (i, j)-elemen adalah 1 jika hanya jika jumlah dari 1-path antara i dan j adalah 1.

2.2. Graf dan Digraf Euler

Sebuah path di dalam graf adalah **path Euler** jika setiap edge dari graf dilalui tepat satu kali. Sebuah *path* Euler tertutup adalah sebuah **circuit Euler**. Graf dikatakan **graf Euler** jika graf tersebut mempunyai sebuah *circuit* Euler. Graf G terhubung tanpa ada *loop* disebut graf Euler jika dan hanya jika *degree* dari setiap verteknya adalah genap.

Perhatikan graf pada gambar 2.11, dengan menggambarkan graf yang sesuai dan menemukan path Euler padanya, dapat diamati perjalanan yang melintasi setiap edge satu kali saja dari semua edge yang ada dan kembali ke vertek awal. Contoh dapat dimulai pada vertek E dan melintasi edge dengan urutan $a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p$. Di sini jelas terlihat bahwa setiap edge hanya dilewati tepat satu kali, ini dimungkinkan jika hanya jika sebuah edge sampai pada salah satu vertek maka untuk meninggalkan vertek tersebut harus melalui edge yang lainnya.



gambar 2.11

Graf Euler

Ini berarti bahwa jika ingin ke sembarang vertek maka harus meninggalkan vertek tersebut melalui edge yang lain. Sehingga setiap kali melalui sebuah vertek diperlukan jumlah degree sama dengan kelipatan dua atau berjumlah genap.

Untuk membuktikan apakah graf tersebut merupakan graf Euler atau bukan adalah terdiri dari dua bagian yaitu :

1. Jika *G* graf Euler, maka setiap titik *G* berderajat genap.
2. Jika setiap titik *G* berderajat genap, maka *G* graf Euler.

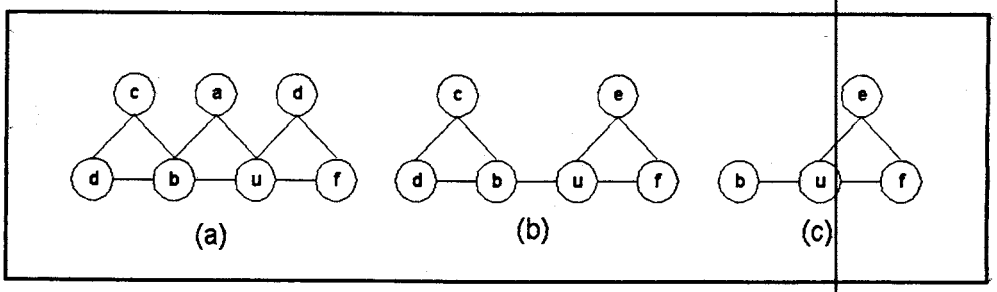
Bagian 1 menunjukkan bahwa syarat itu **diperlukan**, bagian 2 menunjukkan bahwa syarat itu **cukup**. Kekurangan dari pembuktian ini tidak bersifat konstruktif, dalam pengertian bahwa tidak ditunjukkan bagaimana mengkontruksi path Euler di dalam

graf. Salah satu cara untuk mengkontruksikan path Euler adalah dengan **algoritma Fleury**[3].

Adapun algoritma ini adalah sebagai serikut :

1. Pilih vertek awal.
2. Pada setiap tahap, lewati sembarang edge yang ada, dengan memilih sebuah edge hanya bila tidak ada alternatif lainnya.
3. Sesudah melalui setiap edge, hapus edge tersebut dan memilih edge lain yang ada.
4. Berhentilah jika sudah semua edge dihapus atau sudah tidak ada edge yang lainnya.

Penggunaan algoritma Fleury ini diilustrasikan seperti graf pada gambar 2.12(a), (b), (c) berikut ini :



gambar 2.12

Algoritma Fleury

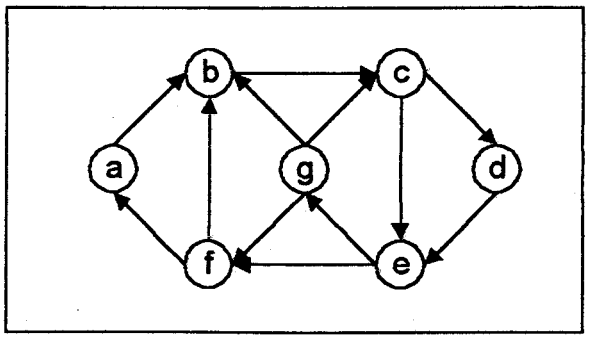
Bila dimulai dari vertek *u* pada gambar 2.12(a), boleh dipilih edge dari pasangan vertek *u* dan *a* diikuti oleh edge dari pasangan vertek *a* dan *b*. Hapus edge

---

3 Theresa M. H. Tirta, *Pengantar Graf*, hal. 133

yang sudah dilalui menghasilkan graf seperti gambar 2.12(b). Edge dari *b* ke *u* tidak dapat digunakan karena merupakan edge penghubung sehingga dipilih edge dari *b* ke *c* dan diikuti edge dari *c* ke *d* dan edge dari *d* ke *b*. Hapus edge-edge tersebut sehingga menghasilkan graf seperti gambar 2.12(c). Sekarang tidak ada pilihan lain selain edge dari *b* ke *u* harus dilaui. Menelusuri circuit *u*, *e*, *f*, dan *u* ternyata melengkapi circuit Euler. Karena itu *u*, *a*, *b*, *c*, *d*, *b*, *u*, *e*, *f*, dan *u* adalah path Euler.

Digraf *D* merupakan **digraf Euler** jika terdapat path tertutup yang memuat setiap arc dari *D* dan memiliki jumlah indegree dan outdegree yang sama untuk setiap verteknya. Seperti gambar 2.13 path Eulernya adalah *a*, *b*, *c*, *d*, *e*, *f*, *b*, *g*, *c*, *e*, *g*, *f*, *a* merupakan path tertutup.



gambar 2.13

Digraf Euler

Salah satu permasalahan yang menarik yang penyelesaiannya memanfaatkan graf Euler adalah *Chinese postman problem*[4]. Pada *Chinese postman problem* seorang

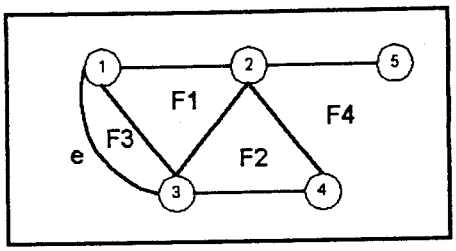
[4] Permasalahan ini dikemukakan pada tahun 1962 oleh Meigu Guan.

tukang pos berharap untuk dapat mengantar kiriman pos ke tempat-tempat di sepanjang semua jalan daerah tugasnya secara berurutan, dan kemudian kembali ke kantor pos. Bagaimana rute tersebut dapat direncanakan sehingga jarak total yang ditempuh seminimum mungkin. Jika peta daerah tukang pos itu ternyata berkorespondensi dengan graf Euler, maka tidak ada kesulitan mencari pemecahannya. Tukang pos itu tinggal memilih *path* Euler, dan *path* ini akan memberikan jarak total terpendek. Yang sering terjadi dalam praktek adalah tukang pos itu perlu mengunjungi bagian-bagian rute lebih dari satu kali, dan menginginkan jarak yang dilaluinya sependek mungkin. Dengan diasumsikan bahwa panjang setiap bagian rute diketahui. Malasalah ini dapat diselesaikan dengan menggunakan metode yang mengkombinasikan algoritma graf Euler dan algoritma *path* terpendek. Permasalahan *postman problem* ini lebih lanjut akan dijelaskan pada bab 3.

2.3. Graf Planar

Sebuah graf dikatakan **graf Planar** jika graf tersebut digambarkan tanpa ada persilangan dua edge kecuali pada verteknya. Planar graf yang tergambar pada sebuah bidang dengan tanpa adanya persilangan dua edge disebut dengan **plane graf**. Wilayah dua dimensi yang dibatasi oleh beberapa edge adalah disebut **faces**, dan vertek dan edge-edge yang membatasi sebuah *face* disebut dengan **boundaries** dari *face*. Dalam sebuah *plane* graf pada gambar 2.14 terdapat lima vertek, tujuh edge, dan empat *face*. Juga dapat terlihat bahwa *F1*, *F2*, dan *F3* adalah interior *face*, dan wilayah yang tidak

tertutup pada *F4* adalah *exterior face*. Batas dari *F1* adalah *cycle* 1 - - - - 2 - - - - 3 - - - - 1 dan batas dari *F4* adalah 1 - - - - 2 - - - - 5 - - - - 2 - - - - 4 - - - - 3 - - - - 1.



gambar 2.14

Sebuah plane graf

Nilai yang dihasilkan dari hubungan vertek, edge, dan *face* adalah mengukiti teorema dari Euler yaitu : jika sebuah plane graf terhubung mempunyai n vertek, m edge, maka jumlah dari *face* adalah p dimana $n - m + p = 2$.

2.4. Path Terpendek

Jika besaran jarak setiap arc dari digraf sudah ditentukan, ini merupakan hal dasar dalam permasalahan untuk menemukan *path* terpendek dari sebuah vertek ke vertek lain yang sudah ditentukan. Beberapa permasalahan komplek di dalam reset operasi dapat dipecahkan oleh prosedur yang disebut dengan algoritma *path* terpendek. Permasalahan *path* terpendek merupakan permasalahan dasar dan biasa ditemukan pada permasalahan optimasi. Ada dua tipe untuk pemecahan permasalahan *path* terpendek yang perlu diperhatikan :

1. mencari *path* terpendek dari vertek *v* ke vertek *w*
2. mencari *path* terpendek dari setiap vertek ke setiap vertek yang lainnya.

Dimana tipe (1) merupakan hal khusus dari tipe (2).

Di sini akan dibahas sebuah algoritma untuk memecahkan permasalahan *path* terpendek. Algoritma untuk mencari *path* terpendek dan mencari jarak terpendek dari satu vertek ke setiap vertek lainnya adalah **algoritma Dijkstra's**[5]. Untuk algoritma Dijkstra's diasumsikan bahwa semua fungsi bernilai positip, ini bertujuan untuk mempermudah perhitungan. Ada perbedaan antara permasalahan yang mengandung fungsi yang bernilai positip dan fungsi yang memiliki nilai baik positip maupun negatip. Permasalahan akan menjadi tak terbatas jika graf mempunyai sebuah *cycle* yang bernilai negatip.

Pada digraf $G = (V, E)$, dimana $V = \{1, 2, \ldots, n\}$ dan nilai dari arc (i, j) adalah $a(i, j)$ yang diasumsikan berniali positip. Jika pada digrap ini tidak ada arc dari i ke j, maka nilai $a(i, j)$ tidak dapat ditentukan. Untuk digraf G memiliki matrik n x n dengan nilai dari matrik $A = (a(i, j))$, dengan nilai diagonalnya adalah 0. Permasalahan ini adalah untuk mencari jarak terpendek dan path terpendek dari vertek 1 ke semua vertek yang ada.

Prosedur dari algoritma Dijkstra adalah sebagai berikut. Setiap vertek i ditentukan dengan suatu fungsi yang bersifat permanen maupun sementara. Fungsi $L(i)$ dari i adalah bersifat permanen yang merupakan jarak terpendek dari 1 ke i, sedangkan

---

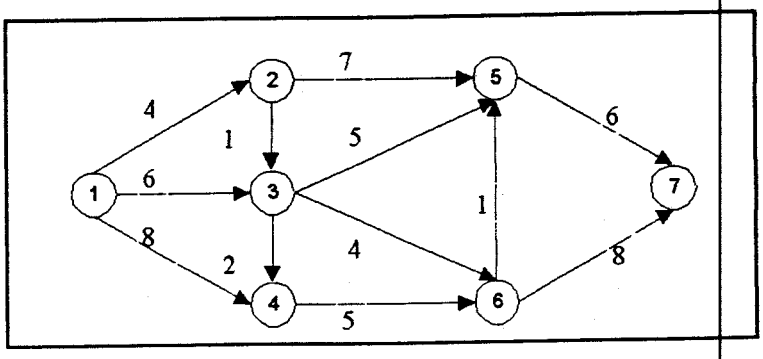
5 Ditemukan oleh Dijkstra pada tahun 1959

$L'(i)$ dari i adalah bersifat sementara yang merupakan pilihan selanjutnya untuk menetukan jarak terpendek dari 1 ke i. Pada setiap tahap dari prosedur, P merupakan himpunan vertek dengan fungsi yang bersifat permanen dan T merupakan komplemen dari P. Jika $P = \{1\}$ maka $L(1) = 0$ dan $L'(i) = a(1, i)$ untuk setiap i. Ketika $P = V$ algoritma ini dihentikan. Setiap iterasi terdiri dari dua step sebagai berikut:

Step 1: Cari sebuah vertek k pada T yang mana $L'(k)$ adalah minimum. Rubah fungsi k menjadi fungsi yang permanen dan masukkan k ke himpunan P. Berhenti jika $P = V$.

Step 2: Jika j adalah vertek pada T, bandingkan nilai $L'(j)$ dengan $L(k) + a(k, j)$ dan isi $L'(j)$ dengan nilai yang terkecil. Balik ke step 1.

Agar lebih jelas, maka ditampilkan sebuah contoh untuk mencari jarak terpendek dan path terpendek dari vertek 1 ke vertek lainnya pada digraf seperti gambar 2.15



gambar 2.15

Digraf, dicari shortestpath dari titik 1 ke titik 7

Iterasi 1

***Step 1*:**

$P = \{1\}$

$L(1) = 0$

$T = \{2, 3, 4, 5, 6, 7\}$

$L'(2) = 4$

$L'(3) = 6$

$L'(4) = 8$

$L'(5) = -$

$L'(6) = -$

$L'(7) = -$

***Step 2*:**

$P = \{1, 2\}$

$L(1) = 0$

$L(2) = 4$

$T = T - \{2\} = \{3, 4, 5, 6, 7\}$

$L'(3) = \text{Min}\{6, L'(2) + d(2, 3)\}$

$L'(4) = \text{Min}\{8, L'(2) + d(2, 4)\}$

$L'(5) = \text{Min}\{-, L'(2) + d(2, 5)\}$

$L'(6) = \text{Min}\{-, L'(2) + -\}$

$L'(7) = \text{Min}\{-, L'(2) + -\}$

Iterasi 2

Step 1:

$P = \{1, 2\}$

$L(1) = 0$

$L(2) = 4$

$T = \{3, 4, 5, 6, 7\}$

$L'(3) = 5$

$L'(4) = 8$

$L'(5) = 11$

$L'(6) = -$

$L'(7) = -$

Step 2:

$P = \{1, 2, 3\}$

$L(1) = 0$

$L(2) = 4$

$L(3) = 5$

$T = T - \{3\} = \{4, 5, 6, 7\}$

$L'(4) = \text{Min}\ \{8, L'(3) + d(3, 4)\}$

$L'(5) = \text{Min}\ \{11, L'(3) + d(3, 5)\}$

$L'(6) = \text{Min}\ \{-, L'(3) + d(3, 6)\}$

$L'(7) = \text{Min}\ \{-, L'(3) + -\}$

Iterasi 3

Step 1:

$P = \{1, 2, 3\}$

$L(1) = 0$

$L(2) = 4$

$L(3) = 5$

$T = \{4, 5, 6, 7\}$

$L'(4) = 7$

$L'(5) = 10$

$L'(6) = 9$

$L'(7) = -$

Step 2:

$P = \{1, 2, 3, 4\}$

$L(1) = 0$

$L(2) = 4$

$L(3) = 5$

$L(4) = 7$

$T = T - \{4\} = \{5, 6, 7\}$

$L'(5) = \text{Min}\ \{10, L'(4) + -\}$

$L'(6) = \text{Min}\ \{9, L'(4) + d(4,6)\}$

$L'(7) = \text{Min}\ \{-, L'(4) + -\}$

Iterasi 4

Step 1:

$P = \{1, 2, 3, 4\}$ $T = \{5, 6, 7\}$

$L(1) = 0$ $L'(5) = 10$

$L(2) = 4$ $L'(6) = 9$

$L(3) = 5$ $L'(7) = -$

$L(4) = 7$

Step 2:

$P = \{1, 2, 3, 4, 6\}$ $T = \{5, 6, 7\} - \{6\} = \{5, 7\}$

$L(1) = 0$ $L'(5) = \text{Min}\ \{10, L'(6) + d(6,5)\}$

$L(2) = 4$ $L'(7) = \text{Min}\ \{-, L'(6) + d(6,7)\}$

$L(3) = 5$

$L(4) = 7$

$L(6) = 9$

Iterasi 5

Step 1:

$P = \{1, 2, 3, 4, 6\}$ $T = \{5, 7\}$

$L(1) = 0$ $L'(5) = 10$

$L(2) = 4$ $L'(7) = 17$

$L(3) = 5$

$L(4) = 7$

$L(6) = 9$

Step 2:

$P = \{1, 2, 3, 4, 6, 5\}$ $T = \{5, 7\} - \{5\} = \{7\}$

$L(1) = 0$ $L'(7) = \text{Min}\ \{17, L'(5) + d(5, 7)\}$

$L(2) = 4$

$L(3) = 5$

$L(4) = 7$

$L(6) = 9$

$L(5) = 10$

Iterasi 6

Step 1:

$P = \{1, 2, 3, 4, 5, 6\}$ $T = \{7\}$

$L'(7) = 16$

Step 2:

$P = \{1, 2, 3, 4, 5, 6, 7\}$ T = himpunan kosong

$L(1) = 0$

$L(2) = 4$

$L(3) = 5$

$L(4) = 7$

$L(6) = 9$

$L(5) = 10$

$L(7) = 16$

Dengan diketahui jarak terpendek dari vertek 1 ke setiap vertek, sangatlah mudah untuk menentukan *path* terpendek dari 1 ke setiap vertek. Ini dapat dicapai dengan mencari jarak terpendek dari percabangan *tree* pada 1 dengan urutan sebagai berikut.

Untuk setiap vertek i, cari sebuah vertek j sedemikian bahwa:

1. terdapat sebuah arc dari j ke i di dalam digraf
2. $L(j) < L(i$
3. $L(j) + a(j, i) = L(i)$.

Masukkan arc (j, i) ke dalam tree. Pada contoh di atas terdapat arc dari 3 dan 4 ke 6.

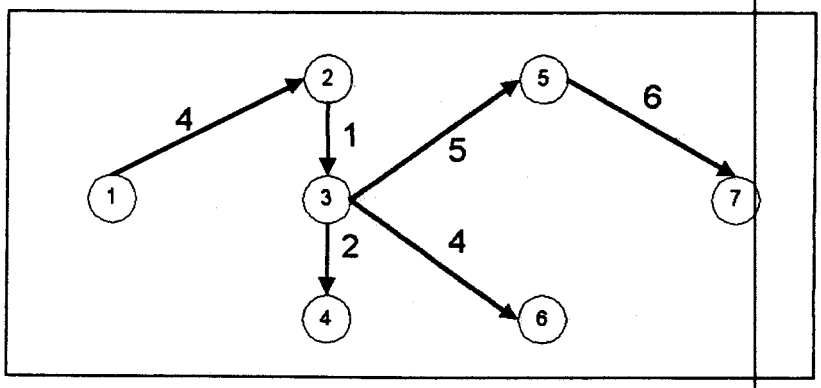
Di bawah ini dapat dilihat bahwa

$$L(3) + a(3, 6) = 5 + 4 = 9 = L(6)$$

dan

$$L(4) + a(4, 6) = 7 + 5 = 12 = L(6)$$

Dengan demikian maka arc (3, 6) yang dimasukkan dalam *tree*. Pada gambar 2.16 adalah merupakan hasil dari prosedur algoritma Dijkstra seperti yang sudah dilakukan di atas.



gambar 2.16

Hasil dari prosedur algoritma Dijkstra

Dalam perencanaan suatu *postman problem*, algoritma *path* terpendek selalu berperan untuk memilih lintasan yang akan digandakan atau lintasan yang tidak berarah menjadi berarah sehingga akan menghasilkan suatu penyelesaian yang mendekati nilai sebenarnya.

BAB III

ALGORITMA EDMONDS DAN JOHNSON

Dalam kehidupan sehari-hari sangatlah banyak dijumpai permasalahan yang membutuhkan penyelesaian yang optimal, walaupun permasalahan tersebut bisa diselesaikan dengan banyak cara tapi tetap diusahakan yang paling optimal karena pada jaman sekarang ini waktu adalah hal yang penting.

Seperti permasalahan tukang pos atau *postman problem*, setiap hari harus mengantarkan surat ke rumah-rumah yang dituju oleh alamat yang tertera pada amplop surat. Sebelum membawa ke tempat tujuan tukang pos akan melihat dulu alamat jalan mana saja yang dituju oleh surat-surat tersebut kemudian membuat peta perjalanan dari semua jalan yang harus dilalui. Untuk menghemat waktu dan bensin, harus memperhitungkan rute perjalan yang terpendek.

Ada berberapa hal penting yang menjadi perhatian dalam menentukan rute perjalanan dari tukang pos seperti adanya jalan satu arah, dan menentukan jalan yang dilalui lebih dari sekali. Pada prinsipnya rute perjalan tukang pos dimulai dari kantor pos kemudian menuju jalan-jalan yang sudah ditentukan dan berakhir kembali ke kantor pos.

Untuk mengimplementasikan rute perjalanan tukang pos dapat menggunakan **mixed graf** dengan menentukan vertek sebagai kantor atau persimpangan jalan, **edge** sebagai jalan yang tidak berarah, **arc** sebagai jalan yang

berarah, dan angka sabagai panjang dari jalan. Sedangkan rumah yang dituju tidak dicantumkan karena terletak sepanjang rute yang dilalui.

Di dalam Tugas Akhir ini dibahas cara untuk mencari lintasan terpendek dari *postman problem*. Ada beberapa algoritma yang bisa diterapkan dan dibahas pada BAB ini, tetapi yang diimplementasikan dalam program hanya **Algoritma Edmonds And Johnson**, dan sebuah algoritma lain yaitu **Algoritma Kedua untuk Mixed Postman** sebagai pembanding. Karena setiap algoritma memiliki kelebihan dan kekurang maka kedua algoritma ini bisa digunakan bersama-sama.

3.1. Algoritma Edmonds dan Johnson untuk Mixed Postman

Metode penyelesaian untuk *Chinese postman problem* harus menemukan harga tambahan minimum dari graf yang dapat memenuhi syarat-syaratnya dan barulah mengidentifikasi perjalanan dari graf tambahan tersebut.

Algoritma Edmonds dan Johnson untuk *mixed postman problem* pada dasarnya terdiri dari dua tahap. Tahap pertama yaitu mengubah graf, membuat *degree* dari setiap vertek adalah genap, memperlakukan setiap arc sebagai edge. Tahap kedua yaitu buat *indegree* dari setiap vertek sama dengan *outdegree*, sambil menjaga syarat-syarat dari setiap vertek haruslah *degree* yang genap. Perjalanan dapat dengan mudah ditemukan dari hasil graf tersebut. Algoritma Edmonds dan Johnson dalam penyelesaiannya dibagi menjadi beberapa *subroutine* seperti di bawah ini.

Algoritma EDMONDS AND JOHNSON

Input : sebuah mixed graf $G = (V, E, A)$ dan nilai fungsi c untuk $E \cup A$.

Output : perjalanan yang dapat meliputi semua edge dan arc pada G.

1. Gunakan algoritma EVENDEGREE.
2. Gunakan algoritma INOUTDEGREE menggunakan hasil dari EVENDEGREE.
3. Gunakan algoritma EVENPARITY menggunakan hasil dari INOUTDEGREE.
4. Temukan arah dari edge pada graf hasil, dan lalui semua arc dan edge yang sudah berarah.

Dalam tahap pertama dari Algorima EDMONDS AND JOHNSON adalah membuat *degree* dari graf adalah genap. Untuk itu digunakan Algoritma EVENDEGREE.

Algoritma EVENDEGREE mengidentifikasikan vertek-vertek yang mempunyai *degree* ganjil, kemudian dengan menggunakan persamaan Lintasan Terpendek, arc dan edge yang menghubungkan pasangan vertek ber*degree* ganjil digandakan. Seperti graf pada gambar 3.1, dengan Algoritma EVENDEGREE menghasilkan graf seperti gambar 3.2 (a).

Adapun Algoritma EVENDEGREE diijelaskan sebagai berikut

Algoritma EVENDEGREE

Input : berupa mixed graf $G = (V, E, A)$ dengan nilai fungsi c untuk $E \cup A$.

Output : berupa mixed graf $G' = (V, E', A')$ dimana $E \subseteq E'$, $A \subseteq A'$, dan *degree* dari setiap vertek adalah genap, dengan mengabaikan arah dari arc.

1. Indentifikasi vertek yang mempunyai degree ganjil.
2. Cari semua path terpendek antara vertek yang mempunyai *degree* ganjil, dengan mengabaikan arah dari arc.
3. Tentukan harga minimum yang sesuai dari vertek yang mempunyai *degree* ganjil dengan menggunakan jarak *path* terpendek.
4. Isikan nilai A' dengan A dan E' dengan E. Masukkan arc yang dihasilkan dari path terpendek ke dalam A', dan edge yang dihasilkan dari *path* terpendek ke dalam E'.

Output dari Algoritma EVENDEGREE adalah menghasilkan *degree* dari setiap graf genap yang sudah merupakan syarat dari graf Euler tapi karena terdapat arc pada graf maka *indegree* dan *outdegree* harus disamakan. Untuk menyamakan *indegree* dan *outdegree* digunakan Algoritma INOUTDEGREE.

Algoritma INOUTDEGREE membuat copy tambahan dari arc dan edge, dan menentukan arah dari beberapa edge yang mana dapat membuat *indegree* dan *outdegree* dari setiap vertek adalah sama. Edmonds dan Johnson sudah memformulasikan *problem* di dalam suatu alur *problem* dari jaringan harga minimum, yang mana setiap harga dari arc dan edge tambahan diminimalkan. Isikan E_1 dan E_2 dari edge yang mempunyai arah sesuai dengan E yang sudah didapat. E_2

akan digunakan untuk menentukan arah dari edge-edge tersebut dengan menggunakan algoritma INOUTDEGREE. E_1 akan digunakan untuk menentukan berapa banyak *copy* tambahan yang dikehendaki dari edge yang yang telah mempunyai arah.

Algoritma INOUTDEGREE

Input : mixed graf $G = (V, E, A)$ dengan nilai fungsi c.

Output : isikan M yang mengandung setiap arc dari G paling tidak sekali dan edge yang mempunyai arah, seperti untuk setiap vertek *indegree* sama dengan *outdegree*. Juga, isikan $U \subseteq E$ dari edge yang belum berarah dan belum dimasukkan dalam M.

1. Isikan E_1 dengan edge yang sudah mempunyai arah, seperti pada setiap edge (v, w) dalam E yang mana ada dua edge yang mempunyai arah $<v, w>$ dan $<w, v>$ dalam E_1. Isikan E_2 dengan cara yang sama. Buatlah $E_t = E_1 \cup E_2 \cup A$.
2. Untuk setiap vertek v, isikan b_1 dengan jumlah arc pada A yang menuju v dikurangi dengan jumlah arc pada A yang meningggalkan v.
3. Minimalkan Z

$$Z = \sum_{s \in A} c_s x_s + \sum_{s \in E_1} c_s x_s$$

x_s adalah non negatif integer untuk $s \in A \cup E_1$

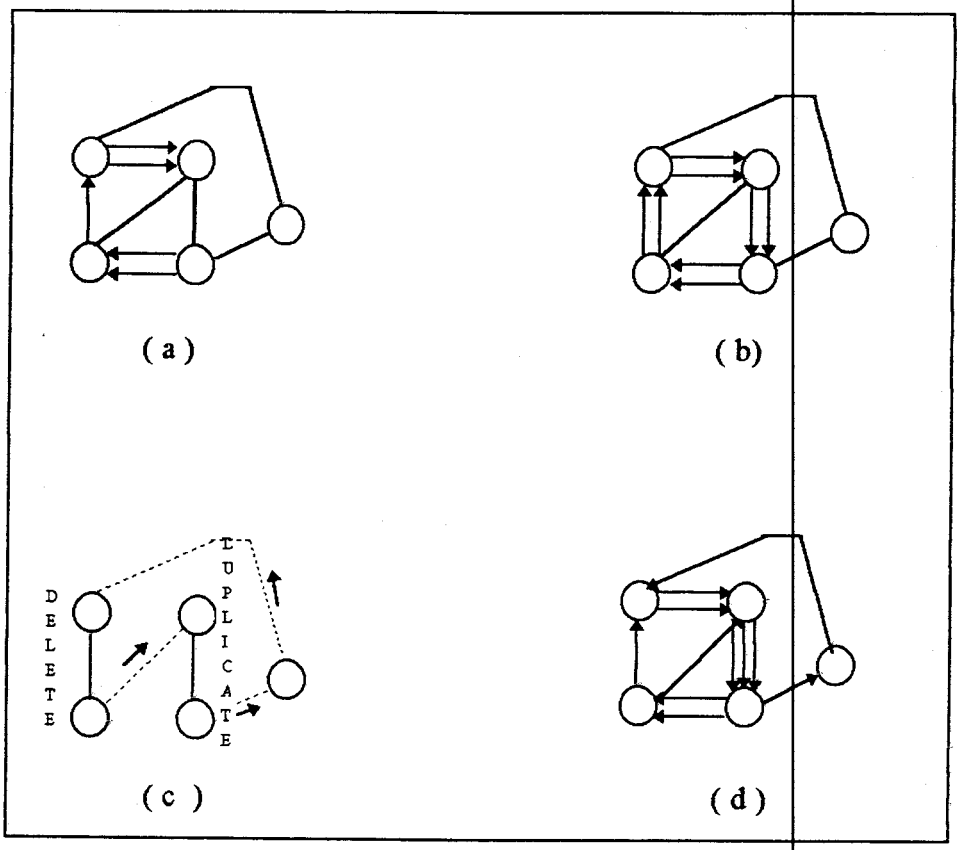
$x_s = 0$ atau 1 untuk $s \in E_2$

$$\sum \{x_s | s \in E_t \text{ keluar dari } v\} - \sum \{x_s | s \in E_t \text{ menuju } v\} = b_1$$

yang mana x_s adalah jumlah pertambahan *copy* arc dari A, atau jumlah *copy* dari edge s yang mempunyai arah dari E_1 atau E_2.

4. Identifikasikan U kosong dan M sama dengan A. Untuk setiap arc s di dalam A, masukkan x_s *copy* tambahan dari s ke M. Setiap x_s yang mempunyai arah pada E_1, masukkan x_s *copy* dari s pada M.
5. Dari setiap pasang edge berarah $s_1 = <v, w>$ dan $s_2 = <w, v>$ dalam E_2 : jika $x_{s1} + x_{s2} = 1$ maka masukkan x_{s1} *copy* dari s_1 dan x_{s2} *copy* dari s_2 pada M. Jika tidak, masukkan (v, w) pada U.

Edmonds dan Johnson memperlihatkan penyelesaian harga minimum dari alur jaringan mengharuskan setiap vertek mempunyai *degree* genap. Tetapi, ada catatan bahwa keharusan alur jaringan tidak cukup untuk memilih satu penyelesaian tentang keharusan *degree* genap. Argumen itu hanya memperlihat keberadaan dari penyelesaian harga minimum dengan *degree* genap. Juga algoritma INOUTDEGREE tidak bisa mempertahankan degree yang genap. Sebagai contoh, lihat graf pada gambar 3.1(*a*). Sebuah tambahan mungkin akan dibuat oleh algoritma INOUTDEGREE dan akan memenuhi hukum Edmonds dan Johnsn seperti terlihat pada gambar 3.1(*b*). Sekarang dilihat algoritma EVENPARITY, yang mana setiap harga dari tambahan arc dan edge tidak akan meningkat. *Output* dari EVENPARITY adalah graf yang memenuhi syarat-syarat pada setiap vertek mempunyai degree genap, dan *indegree* sama dengan *outdegree*.



gambar 3.1

tahan proses algoritma Edmonds and Johnson

Algoritma EVENPARITY

Input : Graf $G = (V, E, A)$ dan masukan M dan U, yang merupakan *output* dari algorima INOUTDEGREE.

Output : Masukkan M' dan U' yang memenuhi kriteria yang sama dengan masukkan M dan U dari INOUTDEGREE, sehingga vertek dari (V, U', M') merupakan *degree* genap.

1. Identifikasi vertek yang mempunyai *degree* ganjil dengan V' terhadap (V, U', M'). Isikan M dengan M', dan U dengan U'.
2. Misalkan $M'' \subseteq M$ merupakan tambahan dari arc dan egde yang sudah berarah yang dihasilkan oleh INOUTDEGREE.
3. Panggil ADJUSTCYCLES.

Algoritma ADJUSTCYCLES mengidentifikasi siklus yang terdiri dari lintasan alternatif pada M'' dan U, dengan masing-masing lintasan berakhir pada setiap ujung dari sebuah vertek ganjil. Lintasan-lintasan pada M'' akan dibentuk tanpa memperhitungkan arah arc dan edge yang berarah. Karena siklus tersebut tertutup, arc atau edge berarah pada siklus tersebut akan diduplikasi atau didelete, dan edge pada siklus akan menjadi terarah. Hal ini diperoleh karena *parity* dari vertek ganjil telah berubah, sedangkan *indegree* sama dengan *outdegree* untuk setiap vertek.

Algoritma ADJUSTCYCLE

1. While V' tidak kosong
2. Pilih v dari V', dan isikan v dengan v_s
3. While v ada dalam V'
4. Ambil v dari V'
5. Repeat

Ambil $[v,w]$ dari M''

If $[v,w]$ langsung menuju w then

Masukkan copy duplikat dari $<v,w>$ pada M'

Else delete sebuah copy $<w,v>$ dari M'

Isikan v ke w

6. Until v ada dalam V'

7. Ambil v dari V'

8. Repeat

Ambil (v,w) dari U'. Arahkan (v,w) dari v ke w dan masukkan ke dalam M'. Isikan v ke w

9. Until v ada dalm V' atau $v = v_s$.

Algoritma EVENPARITY akan mengubah *parity* dari vertek ganjil saja. Step ke 5 diterapkan pada jumlah genap dari arc dan edge berarah dengan suatu vertek genap, dan pada jumlah ganjil dari arc dan edge dengan vertek ganjil.

Algoritma EVENPARITY tidak akan mengubah sifat bahwa *indegree* sama dengan *outdegree* pada setiap vertek. Pengesetan M' pada satu iterasi dari step 5 atau step 8 dari ADJUSTCYCLES akan menyebabkan w memiliki selisih sementara *indegree* dikurangi *outdegree* yaitu +1. Bagaimanapun juga hal ini selalu dikoreksi pada iterasi berikutnya pada step 5 atau 8.

Harga M' dan U' tidak akan lebih besar dari M dan U. Misalkan harga tambahan dari edge dan arc pada M' lebih besar dari pada M. Maka baliklah arah

setiap edge yang diambil dari U, Masukkan dua copy dari setiap arc atau edge yang berarah yang telah didelete, dan delete dua copy dari setiap edge dan arc yang telah dimasukkan. Tambahan yang baru ini harganya harus lebih kecil dari M dan U. Ini merupakan kontradiksi, karena M dan U diperleh dengan menggunakan algoritma minimum-cost network flow. Jadi harga M' dan U' tidak lebih dari M dan U, karena M dan U dihasilkan dengan INOUTDEGREE, yang meminimasi harga tambahan arc dan edge. Jadi harga M' dan U' sama dengan harga M dan U.

Dapat dilihat suatu contoh EVENPARITY yang diterapkan pada graf pada gambar 3.1(b), yang merupakan hasil penerapan algoritma INOUTDEGREE terhadap graf pada gambar 3.1(a). Diasumsikan bahwa setiap edge dan arc mempunyai harga satuan. Gambar 3.1(c) menunjukkan vertek ganjil, dengan lintasan yang menghubungkannya ditunjukkan dengan garis putus-putus, dan lintasan dari arc dan edge terarah ditunjukkan dengan garis penuh. Pada contoh ini hanya terdapat satu siklus. Bila vertek yang ditandai dengan v_s merupakan vertek pertama, selanjutnya arc dan edge akan diset seperti pada gambar 3.1(c). gambar 3.1(d) menunjukkan graf akhir yang telah dimodifikasi.

TEOREMA 1. bila C^* merupakan harga optimum untuk *mixed postman problem*, dan $\bar{C}$ merupakan harga *tour* yang dihasilkan oleh algoritma MIXED1, maka

$$\bar{C} / C^* \leq 2$$

dan batas tersebut dapat terpenuhi. Kompleksitas waktu dari algoritma MIXED1 tidak kurang dari $O(max\{|V|^3, |A|(max\{|A|, |E|\})^2\})$.

3.2. Algoritma Kedua untuk Mixed Postman Problem

Suatu pendekatan alternatif terhadap mixed postman problem merupakan kebalikan dari pendekatan Edmonds dan Johnson. Pertama *indrgree* dan *outdegree* dari setiap vertek dibuat sama. Kemudian *degree* dari setiap vertek dibuat genap, sambil mempertahankan kerakteristik untuk setiap vertek *indegree* sama dengan *outdegree*. Sekarang diberikan algoritma yang menggunakan algoritma INOUTDEGREE untuk memperoleh batas worst-case dari 2 untuk *mixed postman problem.*

Algoritma LARGECYCLES

Input : output dari INOUTDEGREE.

Output : suatu tour yang meliputi semua edge dan arc dari G.

1. Identifikasi vertek yang ber*degree* ganjil pada graf $G' = (V, U)$.
2. Tentukan semua lintasan terpendek antara vertek ganjil pada graf $G'' = (V, E)$.
3. Lakukan *minimum-cost matching* dari vertek ganjil dengan menggunakan jarak lintasan terpendek. Masukkan edge yang digunakan dalam matching ke dalam U.
4. Tentukan traversal dari arc dan edge berarah pada M dan edge yang tak berarah pada U.

Algoritma MIXED

Input : mixed graf $G = (V, E, A)$ dan fungsi harga c pada $E \cup A$.

Output : tour yang meliputi semua edge dan arc pada G.

1. Call INOUTDEGREE.
2. Call LARGECYCLE.

TEOREMA 2. Bila C^* adalah harga tour optimum untuk mixed postmen problem, dan $\bar{C}$ adalah harga tour yang dihasilkan dengan algoritma MIXED, maka

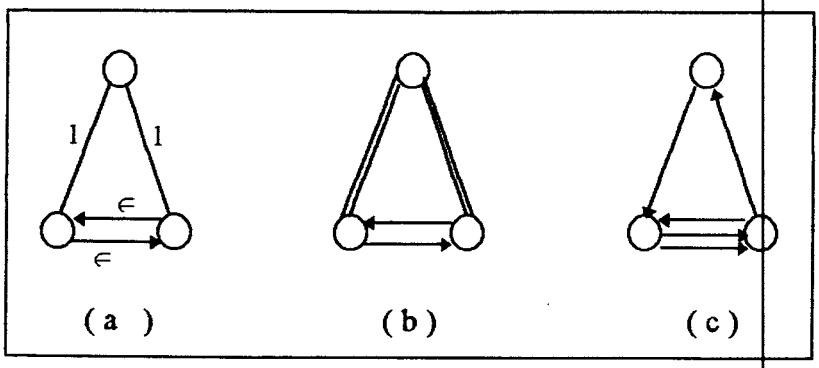
$$\bar{C} / C^* \leq 2$$

dan batas tersebut dapat dipenuhi. Kompleksitas maktunya sama dengan algoritma Edmond And Johnson.

Bila diperhatikan contoh worst-case untuk algoritma MIXED pada gambar 3.2, dapat disimpulkan bahwa penentuan algoritma tersebut kurang baik karena fungsi jarak yang digunakan pada matching dari LARGECYCLE. Harga yang menghubungkan dua vertek dengan suatu lintasan adalah $\in$, dan harus diperhitungkan dalam fungsi jarak. Jarak ditentukan sebagai

$$c'(v, w) = \min\{c(v, w), \max\{c\langle v, w\rangle, c\langle w, v\rangle\}\}$$

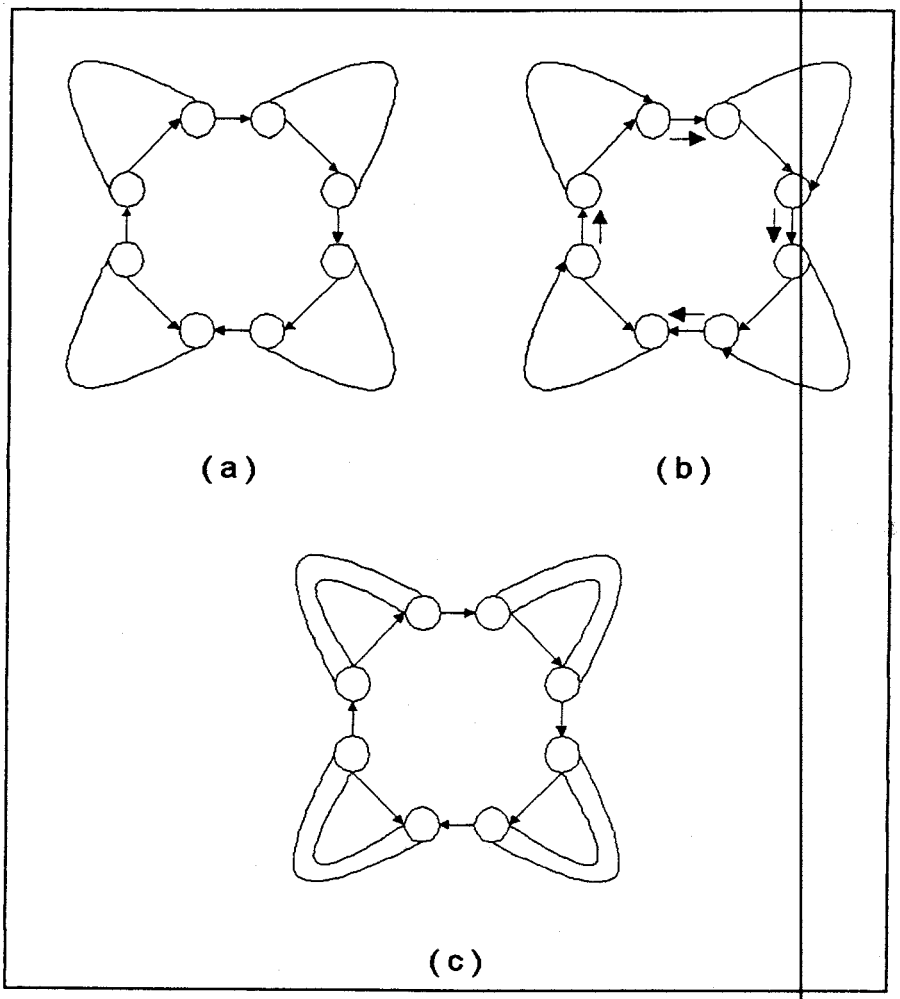
dimana $c\langle v, w\rangle$ adalah harga lintasan berarah terpendek dari v ke w dalam $E \cup A$, dan $c(v, w)$ adalah seperti sebelumnya yaitu merupakan harga lintasan terpendek dari v ke w dalam E.



gambar 3.2

Contoh kasus terjelek dari algoritma MIXED

Fungsi jarak akan membuat algoritma MIXED menghasilkan penyelesaian optimum untuk contoh pada gambar 3.2. Edge yang telah matching dimasukkan ke dalam *U*. Bila traversal ditentukan, jika edge terdiri dari arc, maka arc yang sesuai dengan arah traversal yang digunakan. Ketika fungsi jarak berhasil dengan baik pada contoh ini, pada contoh lain belum tentu berhasil dengan baik. Perhatikan graf pada gambar 3.3(*a*). Solusi optimum menggandakan arc yang kecil dan ditunjukkan pada gambar 3.3(*b*). Fungsi jarak yang telah diubah tidak akan menggunakan arc yang kecil, melainkan harga dari edge. Solusi yang diperoleh ditunjukkan pada gambar 3.3(*c*). Dengan pemilihan harga yang sesuai, contoh tersebut mendekati batas worst-case dari 2.



gambar 3.3

Contoh kasus terburuk, dilihat dari fungsi jaraknya

Masalah yang sama dapat dilihat pada Algoritma EDMONDS AND JOHNSON. Dengan jarak arc digunakan dalam matching dari step 1 pada EDMONDS AND JOHNSON masalah tersebut dapat dilihat pada gamabar 3.4. Bila matching pada step 1 dikerjakan dengan menggunakan fungsi jarak berubah yang sama, maka masalah pada gambar 3.4 dapat diatasi. Bagaimanapun juga,

algoritma tersebut akan sama dengan Algoritma MIXED yang dimodifikasi. Contoh *worst-case* pada gambar 3.3 juga akan menghasilkan karakteristik yang buruk untuk Algoritma EDMONDS AND JOHNSON.

3.3. Mixed Strategy Algorithm untuk Mixed Postman Problem

Bila diperhatikan contoh *worst-case* untuk Algoritma EDMONDS AND JOHNSON dan MIXED, akan didapatkan sebagai berikut : Algoritma MIXED akan menghasilkan tour optimum untuk contoh *worst-case* untuk Algoritma EDMONDS AND JOHNSON pada gambar 3.4. Algoritma EDMONDS AND JOHNSON akan menghasilkan *tour* optimum untuk contoh *worst-case* untuk MIXED pada gambar 3.2. Karakteristik ini memberi kesan bahwa setiap algoritma menangani kasus ekstrim dengan baik dimana algoritma yang lain tidak dapat mengatasinya. Maka dari itu harus dipertimbangkan pendekatan *mixed* strategi dengan menggunakan kedua algoritma tersebut.

Notasi berikut akan digunakan dalam analisa karakteristik worst-case dari algoritma. C^* menyatakan harga tour optimum, dan C_M merupakan harga total dari edge dan arc dalam M yang diperoleh dengan Algoritma INOUTDEGREE pada graf awal.

Algoritma GENERALMIXED

Input : mixed graf G.

Output : tour yang meliputi semua edge dan arc.

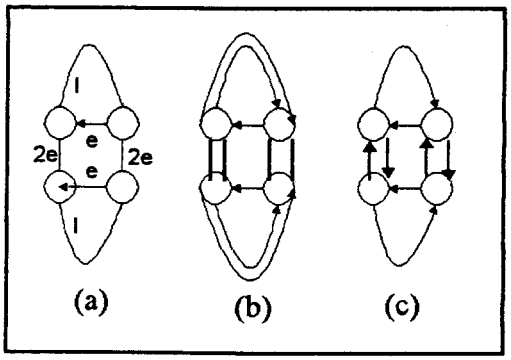
1. Call EDMONDS AND JOHNSON
2. Call MIXED
3. Pilih tour dengan harga yang lebih kecil.

Teorema 3. Algoritma GENERALMIXED menghasilkan tour yaitu

$$\bar{C} / C^* \leq 5 / 3$$

kompleksitas waktu GENERALMIXED sama dengan Algoritma EDMONDS AND JOHNSON.

Pada contoh di atas tidak ditemukan algoritma GENERALMIXED dengan batas *worst-case* 5 / 3. Yang terdekat yang dapat diperoleh adalah $3 / 2^6$, yang mana dapat diperoleh dengan suatu graf yang merupakan kombinasi dari gambar 3.2 dan gambar 3.4.



gambar 3.4

Kasus terburuk dari algoritma Edmonds and Johnson

---

6 Greg N. Frederickson, *Approximation Algorithms for Some Postman Problems*, hal. 547

BAB IV

PERANCANGAN DAN PENGEMBANGAN PERANGKAT LUNAK

Perangkat lunak yang dirancang untuk aplikasi *postman problem* dengan Algoritma Edmonds and Johnson adalah didesain dengan pedoman bahasa pemrograman **Turbo Pascal versi 7**[7] di bawah sistem operasi **DOS**. Untuk mendukung tampilan agar lebih menarik dan dapat beroperasi di bawah sistem operasi **Windows**[8] , input dan output dari aplikasi postman problem dipadukan dengan pemrograman **Borland Delphi 1.0**[9] sehingga mempermudah *user* atau pengguna dalam pengoperasiannya.

4.1. Rancangan Input-Output

Penentuan *Input-Output* merupakan langkah awal dalam proses *design* atau merancang perangkat lunak yang didasarkan pada perangkat lunak yang hendak dikembangkan.

---

7 Turbo Pascal merupakan produk dan hak cipta milik Borland International Corp.
8 Windows merupakan produk dan hak cipta milik Microsoft Corp.
9 Borland Delphi 1.0 merupakan produk dan hak cipta milik Borland International Corp.

Pada perangkat lunak yang dikembangkan, input adalah berupa **mixed graf** $G = (V, E, A)$ yang mewakili rute perjalanan dari *postman problem*, dan nilai fungsi c untuk $E \cup A$.

Agar mempermudah perancangan graf ini diterjemahkan berupa **matriks** bujur sangkar dengan banyaknya baris dan kolom ditentukan oleh jumlah vertek yang ada. Dimana baris dan kolomnya adalah mewakili vertek-vertek dari graf dan isi dari matrik adalah mewakili panjang graf. Sedangkan untuk menandai apakah garis yang menghubungkan antar vertek berupa *directed* graf atau *undirected* graf adalah dengan menginputkan integer tambahan, yaitu 0, 1, dan -1. Misalkan garis berupa *undirected* graf maka input **Edge** sama dengan 1 dan **Arc** sama dengan 0, sedangkan untuk *directed* graf input input **Arc** sama dengan 1 atau -1 tergantung arahnya dan **Edge** sama dengan 0.

Output juga berupa **mixed graf** yang meliputi perjalanan yang melalui semua edge dan arc pada G.

Untuk lebih jelasnya dapat dilihat pada bab 5 Hasil Uji dan Pembahasan.

4.2. Rancangan Struktur Data

Untuk merancang struktur data dari perangkat lunak *Postman Problem*, untuk Tugas Akhir ini, disesuaikan dengan struktur data Pascal yang sudah umum. Program diawali dengan nama program, yaitu `Program Postman_Problem;`, diikuti

deklrasi piranti yang diawali dengan kata baku **unit**, yaitu `uses crt;`, diikuti deklarasi konstanta, deklarasi tipe data, deklarasi perubah dari program utama, prosedur-prosedur, fungsi-fungsi, dan akhirnya bagian program utamanya sendiri. Dari deklarasi konstanta sampai dengan program utama akan dijelaskan dibagian lain di bawah ini.

Deklarasi prosedure dan fungsi sama dengan deklarasi program utama. Artinya baik prosedur atau fungsi juga bisa mempunyai deklarasi konstanta, tipe dan lainnya, yang berbeda dengan deklarasi yang diperuntukkan bagi program utama.

4.2.1. Deklarasi Konstanta

Konstanta yang digunakan untuk perancangan perangkat lunak postman problem ini adalah untuk memberi batasan jumlah vertek yang bisa dimuat. Adapun bentuk deklarasi konstanta ini adalah seberti berikut :

```
const  Max = 50;
```

Pembatasan nilai konstanta maksimum sebesar 50, diperkirakan bahwa dalam perancangan rute perjalanan *postman problem*, vertek yang dilalui tidak lebih dari 50. Tapi jumlah konstanta maksimum ini dapat dirubah menurut kebutuhan.

4.2.2. Deklarasi Tipe Data

Ada beberapa deklarasi tipe data yang digunakan dalam perancangan perangkat lunak ini. Seperti deklarasi tipe data array dimensi satu, deklarasi tipe data array dimensi banyak, deklarasi tipe boolean, dan deklarasi tipe data pointer.

Untuk lebih jelasnya adalah sebagai berikut :

- **Tipe Data Array Dimensi Satu**

Digunakan untuk mengidentifikasikan vertek, apakah vertek tersebut mempunyai *degree* ganjil atau genap, atau apakah vertek tersebut *Indegree* dan *Outdegree*nya sudah sama. Ini diperlukan untuk menentukan proses prosedur Evendegree, prosedur Inoutdegree, prosedur Evenparity, dan prosedur Largecycle.

Adapun bentuk dari deklarasi ini adalah :

```
type Degree = array[1..Max] of integer;
```

- **Tipe Data Array Dimensi Banyak**

Digunakan untuk mengidentifikasikan panjang jarak antara dua buah vertek dari rute postman problem yang ada. Ini dibutuhkan untuk mencari jarak terpendek dalam prosedur Shortestpath. Selain itu array ini juga digunakan untuk menandai perbedaan dari edge, arc, ArcM atau Edge U.

Adapun bentuk dari deklarasi ini sebagai berikut :

```
type Realarr = array[1..Max,1..Max] of real;
```

adalah untuk menyimpan variable jarak dengan tipe real, sedangkan tipe data array di bawah adalah untuk menyimpan variable Arc, Edge, ArcM, dan EdgeU dengan tipe integer.

```
type Table = array[1..Max,1..Max] of integer;
```

- **Tipe Data Boolean**

Digunakan pada prosedur Shortestpath, untuk menandai vertek yang sudah pernah dilalui, agar arah proses prosedur Shortestpath tidak balik lagi ke vertek awal.

Adapun bentuk dari deklarasi ini adalah :

```
type Flag = array[1..Max] of boolean;
```

- **Tipe Data Pointer**

Tipe data pointer yang digunakan di perancangan perangkat lunak postman problem ini ada dua. Yang pertama adalah digunakan untuk menyimpan data dari prosedur Shortestpath, baik berupa data *rute*, *jarak*, dan *vertek yang dilalui*. Data-data inilah yang diolah dalam prosedur Shortestpath, sehingga menghasilkan rute perjalanan dengan jarak yang terpendek.

Adapun bentuk deklarasi ini adalah :

```
tipe Path = ^Short;
     Short = record
         Rute     : Degree;
         Jarak    : integer;
         Vertex   : integer;
         Berikut  : Path;
       end;
```

Sedangkan tipe data pointer yang kedua digunakan untuk menyimpan data dari hasil prosedur Shortestpath, dan disimpan dalam keadaan urut berdasarkan jarak. Ini digunakan pada prosedur Evendegree untuk menentukan tambahan edge atau arc agar *degree* dari vertek menjadi genap, pada prosedur Inoutdegree untuk menentukan tambahan arc sehingga *Indegree* dan *Outdegree*nya sama, pada prosedur Evenparity untuk menentukan edge mana yang dirubah menjadi arc, dan arc mana yang dihapus

dan ditambah, dan prosedur Largecycle kegunaannya sama dengan pada prosedur Evendegree, cuma yang ditambah adalah edgenya saja.

Adapun bentuk deklarasi ini adalah :

```
type Stack = ^Even;
      Even = record
          StartD,
          FinishD : integer;
          Rute2   : Degree;
          Jarak2  : integer;
          Next    : Stack;
       end;
```

4.2.3. Deklarasi Perubah

Deklarasi perubah untuk perangkat lunak postman problem mewakili nilai data tertentu yang akan dioperasikan dalam perancangan perangkat lunak ini. Seperti sudah dijelaskan, setiap perubah harus dinyatakan tipe datanya.

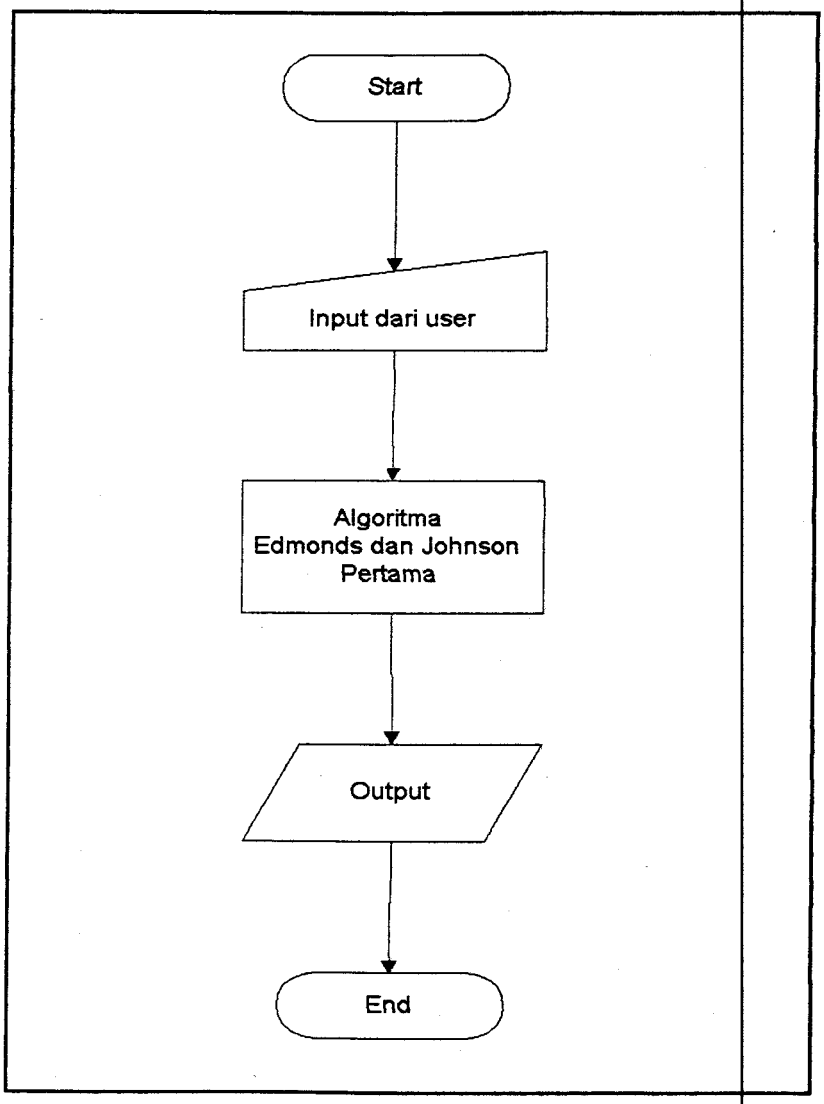
Pada parancangan perangkat lunak ini ada beberapa deklarasi perubah yang digunakan. Deklarasi perubah *Vertice* digunakan untuk menentukan jumlah vertek yang diinputkan, perubah *MatrixG* digunakan untuk menentukan panjang jarak antara dua vertek, perubah *Arc*, *Edge*, *ArcM*, *ArcM1*, *ArcM2*, *EdgeU*, dan *EdgeU1* digunakan untuk manandai edge dan arc, perubah *Odd* untuk menandai vertek berdegree ganjil.

Adapun bentuk deklarasi ini adalah :

```
var
      Vertice                          : integer;
      MatrixG, Arc, Edge, ArcM,
      ArcM1, EdgeU, EdgeU1, ArcM2      : Table;
      Odd                              : Degree;
```

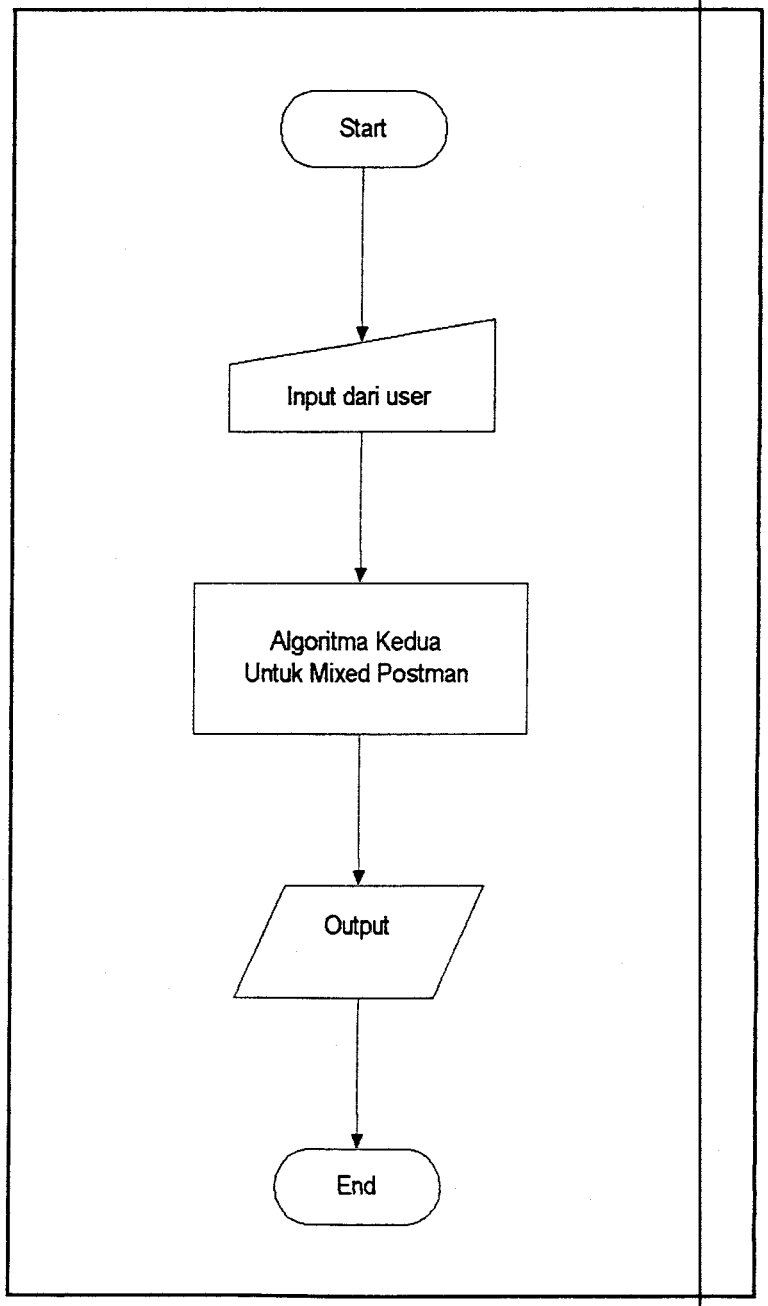
4.3. Diagram Alir Data

4.3.1. Diagram Alir Prangkat Lunak



Gambar 4.1

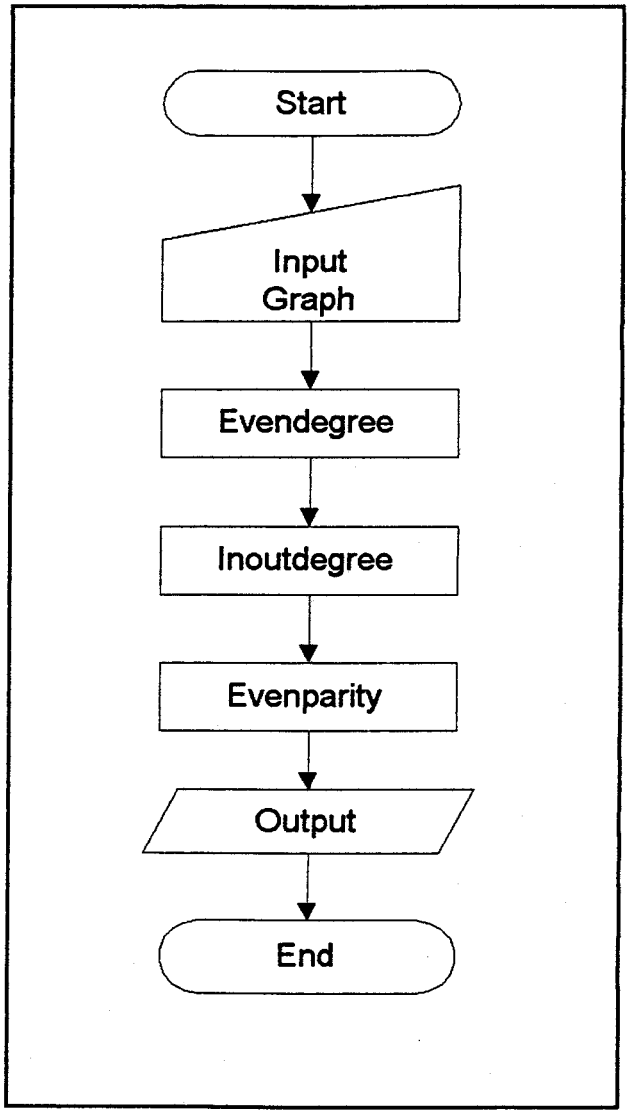
Diagram Alir Sistem Algoritma Edmods And Johnson



Gambar 4.2

Diagram Alir Sistem Algoritma Kedua dari Mixed Postman Problem

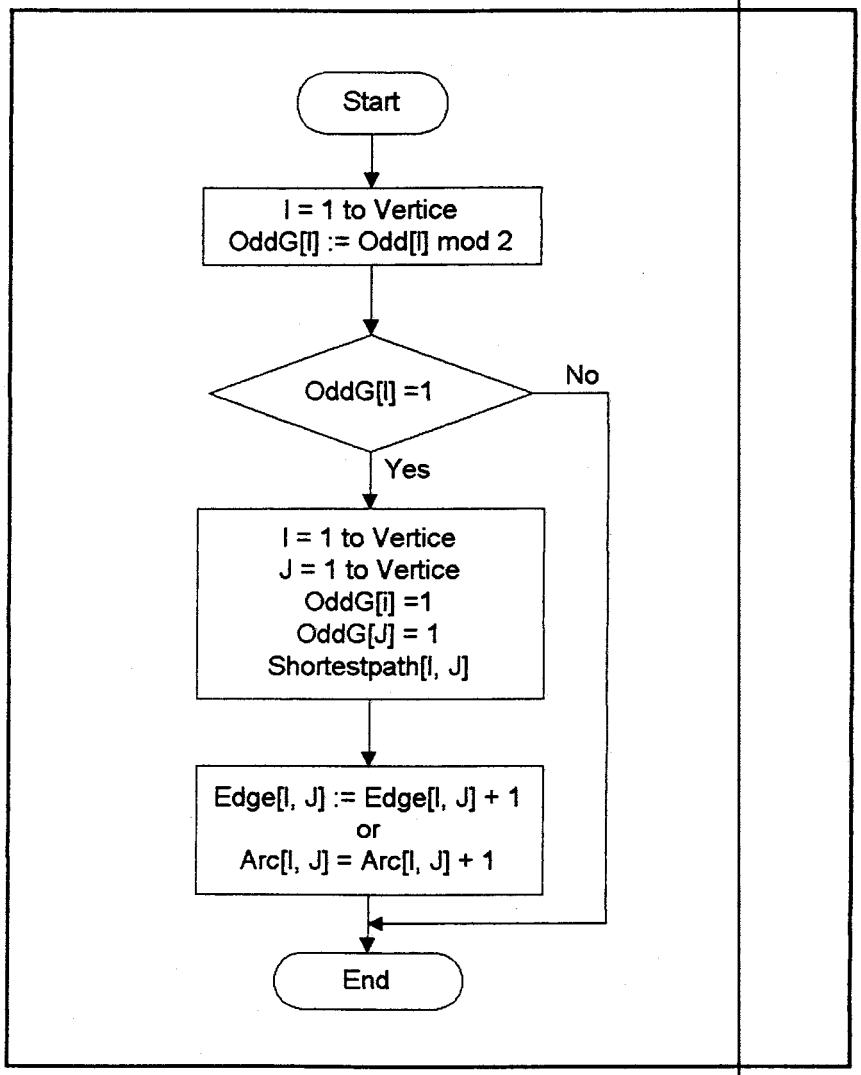
4.3.2. Diagram Alir Algoritma Edmonds And Johnson



Gambar 4.3

Bagan Umum Algoritma Edmonds And Johnson

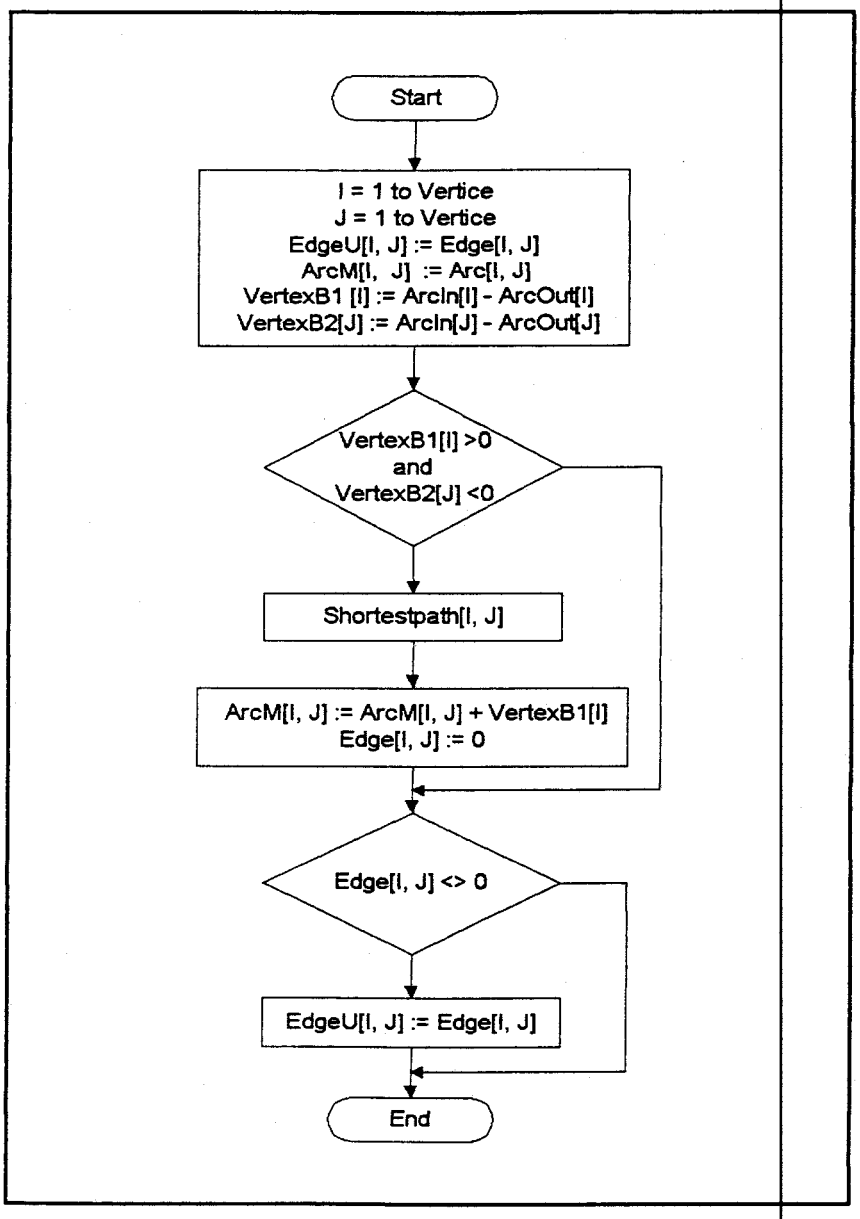
4.3.2.1. Diagram Alir Algoritma Evendegree



Gambar 4.4

Diagram Alir Algoritma Evendegree

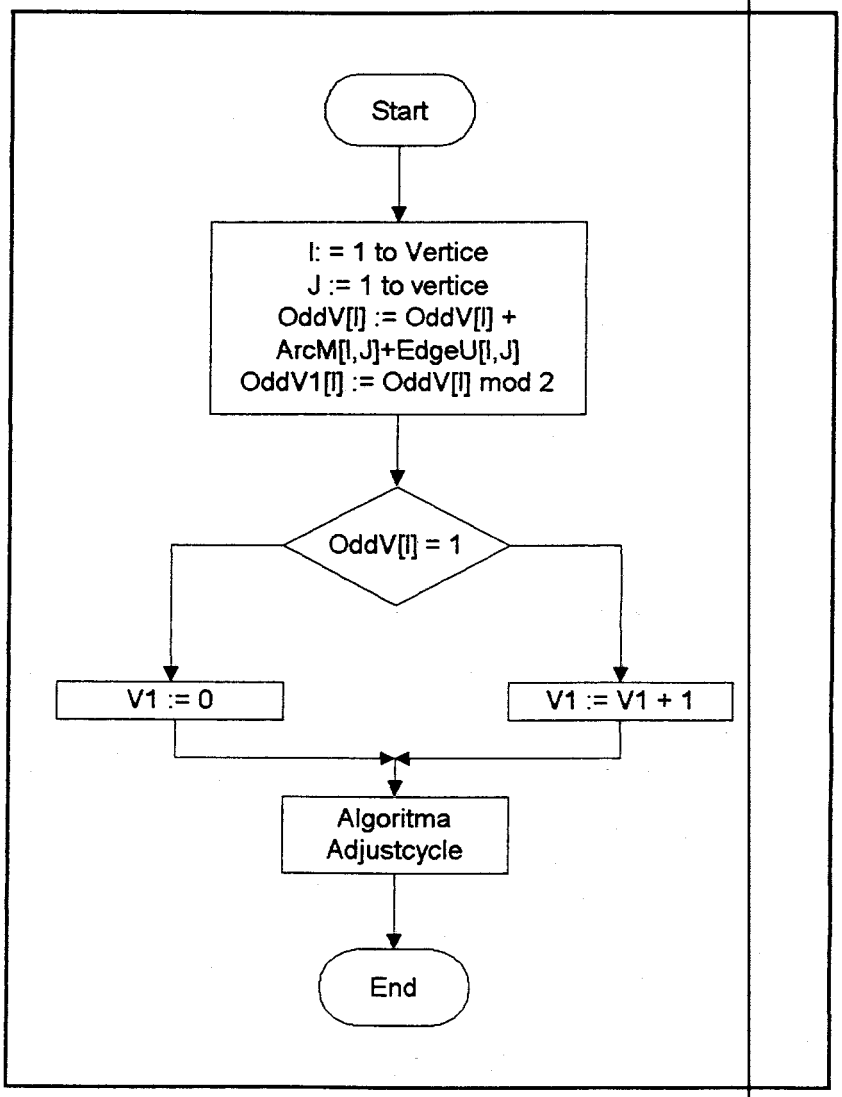
4.3.2.2. Diagram Alir Inoutdegree



Gambar 4.5

Diagram Alir Algoritma Inoutdegree

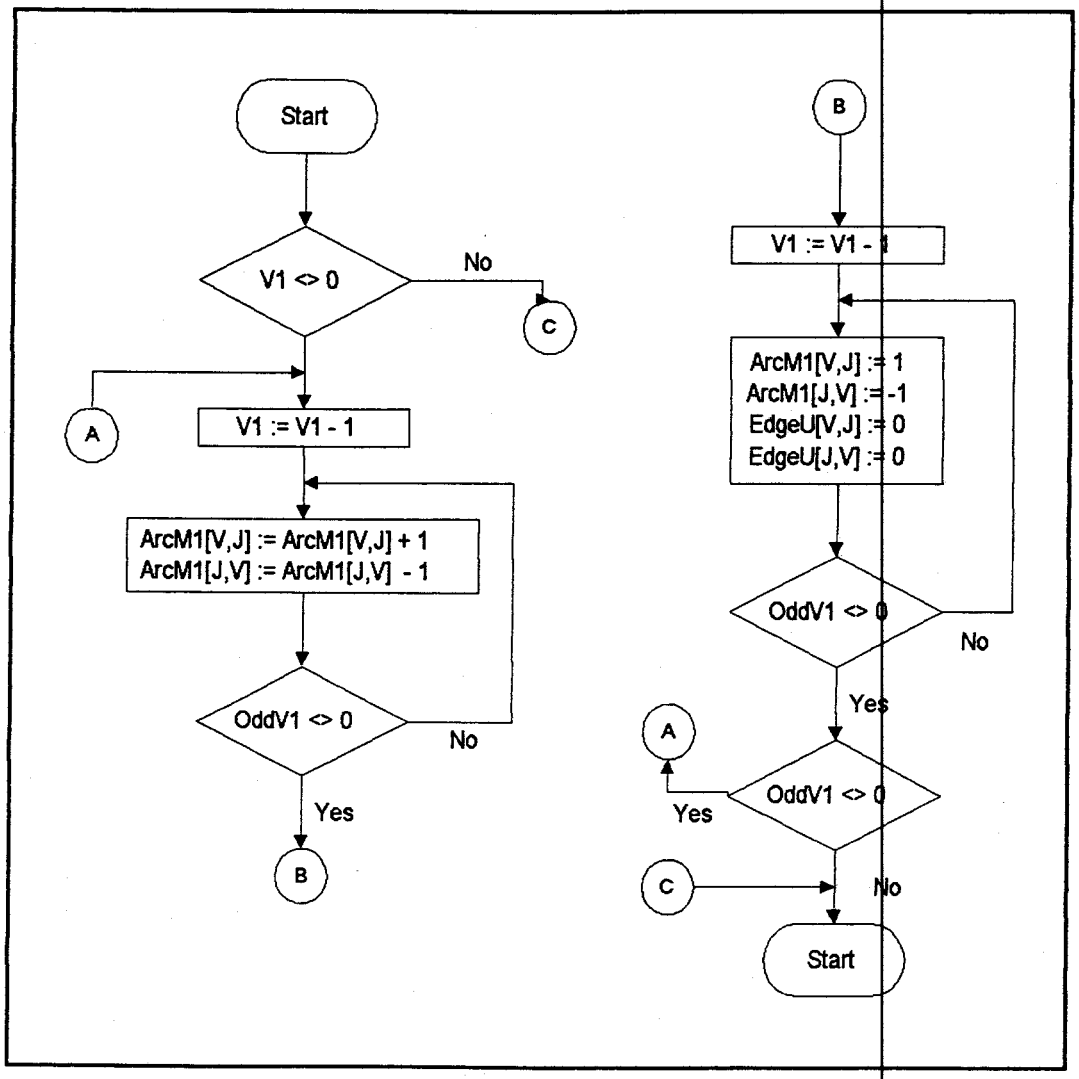
4.3.2.3. Diagram Alir Algoritma Evenparity



Gambar 4.6

Diagram Alir Algoritma Evenparity

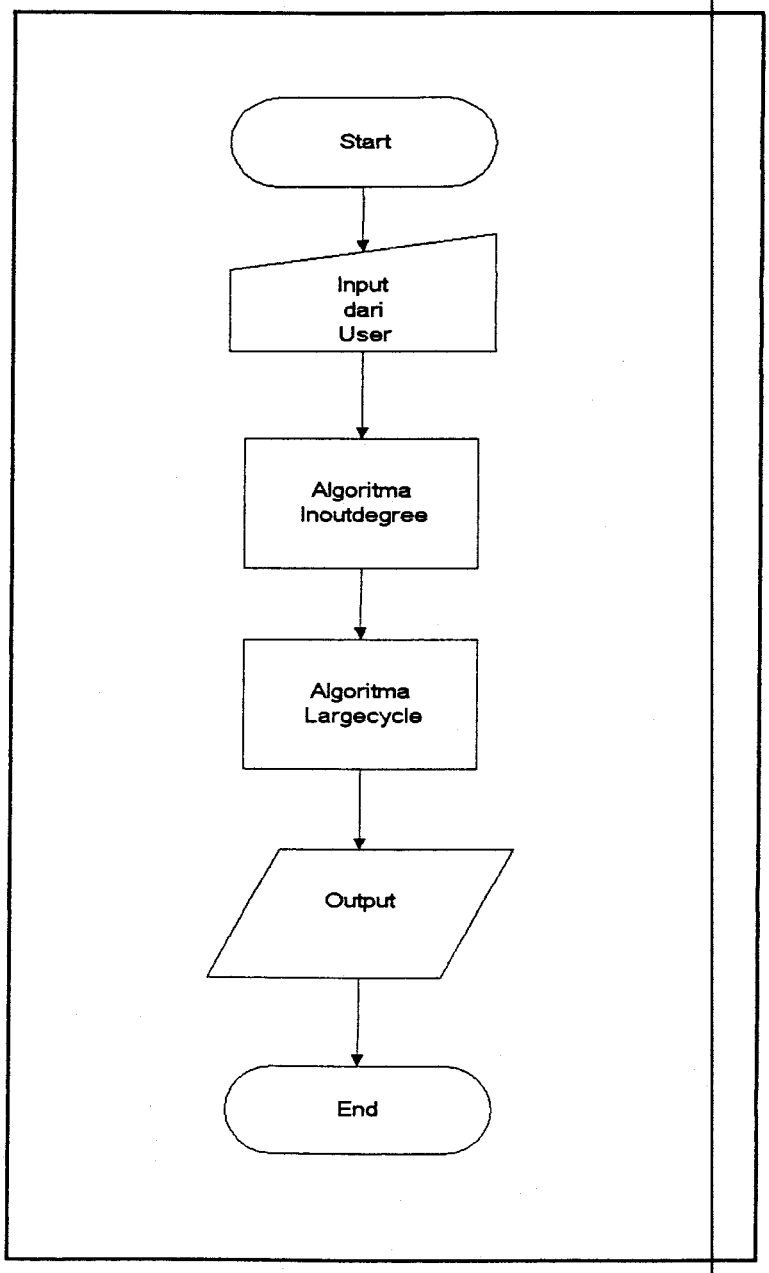
- **Diagram Alir Algoritma Adjustcycle**



Gambar 4.7

Diagram Alir Algoritma Adjustcycle

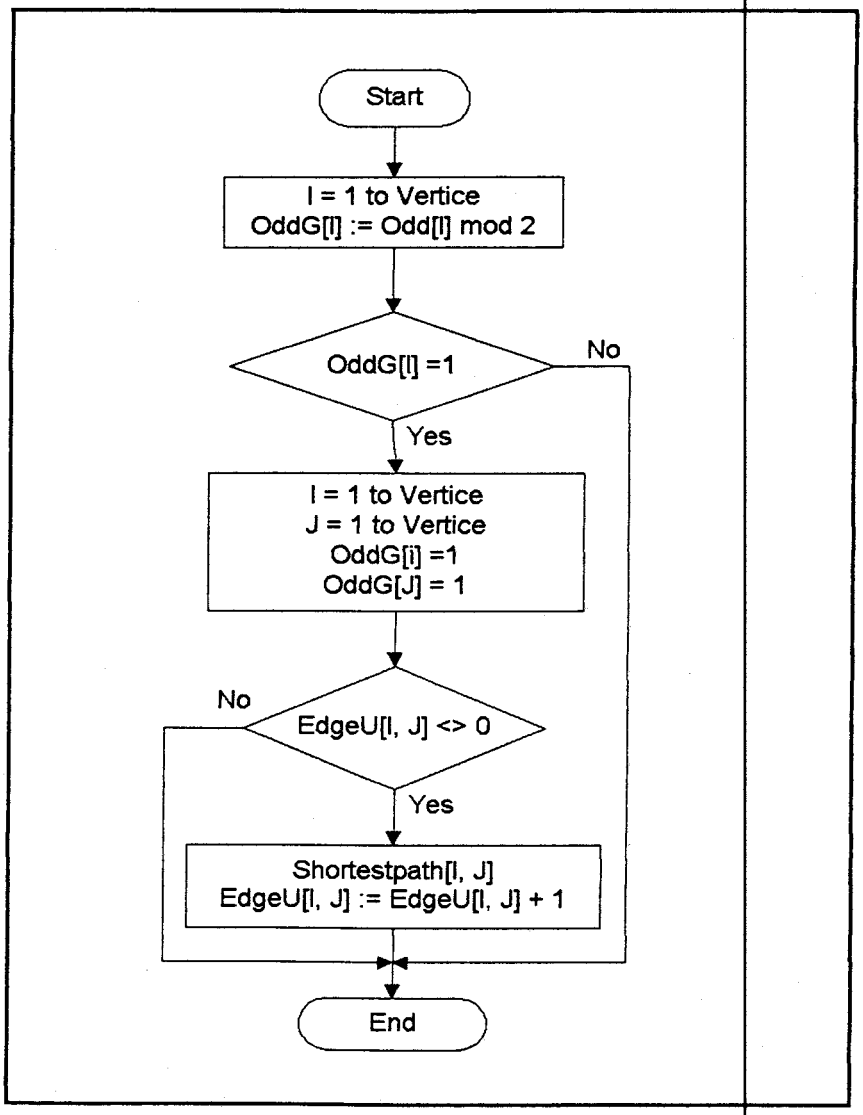
4.3.3. Diagram Alir Algoritma Kedua dari Mixed Postman Problem



Gambar 4.8

Bagan Umum Algoritma Kedua dari Mixed Postman Problem

4.3.3.1 Diagram Alir Algoritma Largecycle



Gambar 4.9

Diagram Alir Algoritma Largecycle

4.4 Rancangan Procedural Perangkat Lunak

Dalam mendisain perangkat lunak untuk *postman problem* banyak tardapat prosedur yang mendukung kinerja perangkat lunak ini. Dalam sub bab ini akan dijelaskan beberapa diantaranya, yang merupakan prosedur-prosedur yang dianggap penting. Prosedur-prosedur ini tidak dijabarkan secara implisit, namun hanya merupakan nama-nama tahapannya. Algoritma prosedur ditulis dalam kode bahasa pemrograman Pascal dengan beberapa modifikasi untuk memudahkan pembacaan.

Semua penulisan rancangan prosedural prangkat lunak postman problem disesuaikan dengan algoritma Edmonds and Johnson yang telas dijelaskan pada bab 3. Untuk mengimplementasikan algoritma Edmonds and Johnson menjadi perngkat lunak diperlukan beberapa prosedur, yang kesemuanya ini akan dijelaskan di bawah ini.

4.4.1. Algoritma Edmonds And Johnson Untuk Mixed Postman

4.4.1.1. Evendegree

Input dari prosedur Evendegree adalah berupa mixed graf $G = (V, E, A)$ dan nilai fungsi c untuk $E \cup A$. Di mana V adalah jumlah vertek, E adalah edge, A adalah arc, dan c adalah panjang edge atau arc. Kesemuanya ini diinputkan oleh user. Bentuk program untuk input adalah :

```
write('Set Vertice = ');
read(Vertice);readln;
gotoxy((J-1)*5+1,Row);
read(MatrixG[I,J]);
gotoxy(30,3);write('Directed Edge[',I,',', J, '] : ');
read(Arc[I,J]);
gotoxy(30,4);write('Undirected Edge[',I,',', J, '] : ');
```

```
read(Edge[I,J]);
if MatrixG[I,J] <> 0 then
  Odd[I] := Odd[I] + 1
```

1. Selanjutnya vertek-vertek ini diidentifikasi, untuk menentukan vertek yang mana memiliki degree ganjil. Variabel Odd berisi jumlah *degree* dari masing-masing vertek, kemudian dengan operasi **mod** dengan 2 dapat menentukan vertek mana saja yang memiliki degree ganjil. Sebagian potongan program akan ditampilkan di bawah ini.

```
Temp[I] := Odd[I] mod 2;
if Temp[I] <> 0 then
  begin
    OddG[I] := 1;
    V1 := V1 + 1
  end
else
OddG[I] := 0
```

   Untuk mengetahui bahwa vertek itu memiliki *degree* ganjil adalah dari `OddG[I]:=1`, untuk vertek yang memiliki degree genap adalah dari `OddG[I] := 0`.

2. Cari semua path terpendek antara vertek-vertek yang mempunyai *degree* ganjil, dengan mengabaikan arah dari arc. Prosedur yang digunakan adalah prosedur Shortestpath. Potongan program dari prosedur Shortestpath seperti di bawah ini.

```
Repeat
  VFlag[Start] := True;
  For i := 1 To Vertice Do
    Begin
      If EdgeOrArc Then isEdge := (MatrixG[Start,I] <> 0)
```

```
        Else isEdge:=((MatrixG[Start,I]<>0)and(Arc[Start,I]>=
                    0));
        If isEdge Then
          Begin
            If i = Finish Then
              Begin
                Rute1[l] := i;
                Tambah(Awal,Akhir,Bantu,Rute1,Jarak1 +
                       MatrixG[Start,i],l,Finish);
                If Bantu^.Berikut <> Nil Then
                  Begin
                    Repeat
                      Hapus(Awal,Akhir);
                    Until Akhir = Bantu;
                  End
              End
            Else
              Begin
                If VFlag[i] <> True Then
                  Begin
                    Rute1[l] := i;
                    Tambah(Awal,Akhir,Bantu,Rute1,
                    Jarak1 + MatrixG[Start,i],l,Finish);
                  End
              End
          End
      End;
    Jarak1 := Awal^.Jarak;
    L := Awal^.Vertex;
    Rute1 := Awal^.Rute;
    Start := Awal^.Rute[L];
    L := L + 1;
    Hapus_Depan(Awal,Akhir);
  until Awal = Nil;
```

Hasil dari prosedur Shortestpath disimpan berupa pointer, yang di dalamnya berisi jarak dari vertek-vertek yang dicari dan rute yang dialui.

3. Setelah semua path terpendek dari vertek-vertek yang memiliki *degree* ganjil didapat dengan prosedur Shortestpath dan disimpan pada pointer lain dengan mengurutkan berdasarkan jaraknya. Seperti pada potongan program di bawah ini.

```
if Awal = nil then
if Awal = nil then
  begin
    Baru^.Next := Nil;
    Awal := Baru;
    Akhir := Baru
  end
else if Jarak1 < Awal^.Jarak2 then
  begin
    Baru^.Next := Awal;
    Awal := Baru;
    Bantu := Baru
  end
else
  begin
    Bantu := Awal;
    while(Jarak1>Bantu^.Next^.Jarak2)and(Bantu^.Next<>Nil)
     do
      Bantu := Bantu^.Next;
    if Bantu^.Next = Nil then
      begin
        Bantu^.Next := Baru;
        Akhir := Baru;
        Akhir^.Next := Nil
      end
    else if Bantu^.Next <> Nil then
      begin
        Baru^.Next := Bantu^.Next;
        Bantu^.Next := Baru
      end
  end
```

4. Pasangan vertek yang memiliki path terpendek dipasangkan terlebih dahulu, agar diperoleh harga minimum yang sesuai. Masukkan path yang dihasilkan dari prosedur Shortestpath ke dalan *A*, dan edge yang dihasilkan dari prosedur Shortestpath ke dalam *E*. Seperti pada potongan program di bawah ini, akan menjelaskan seperti yang sudah diterangkan di atas.

```
If Edge[Start,Rute1[I]] = 1 then
  begin
    Edge[Start,Rute1[I]] := Edge[Start,Rute1[I]] + 1;
    Edge[Rute1[I],Start] := Edge[Rute1[I],Start] + 1
  end
else if Arc[Start,Rute1[I]] = 1 then
  begin
    Arc[Start,Rute1[I]] := Arc[Start,Rute1[I]] + 1;
    Arc[Rute1[I],Start] := Arc[Rute1[I],Start] - 1
  end
else if Arc[Start,Rute1[I]] = -1 then
  begin
    Arc[Start,Rute1[I]] := Arc[Start,Rute1[I]] - 1;
    Arc[Rute1[I],Start] := Arc[Rute1[I],Start] + 1
  end;
```

Hasil dari program di atas merupakan input dari prosedur Inoutdegree yang akan dijelaskan di bawah ini.

4.4.1.2. Inoutdegree

Prosedur Inoutdegree membuat *copy* tambahan dari arc dan edge, dan menentukan arah dari beberapa edge yang mana dapat membuat *indegree* dan *outdegree* dari setiap vertek adalah sama. Edmonds dan Johnson sudah

memformulasikan problem di dalam satu alur problem dari jaringan harga minimum, yang mana setiap harga dari arc dan edge tambahan diminimalkan.

1. Pertama-tama mengisikan E_1 dengan edge yang sudah mempunyai arah, seperti pada setiap edge (v,w) dalam E yang mana ada dua edge yang mempunyai arah $<v, w>$ dan $<w, v>$ dalam E_1. E_1 menentukan berapa banyak *copy* tambahan yang dikehendaki dari edge yang telah mempunyai arah. Kemudian mengisikan E_2 dengan cara yang sama seperti pada E_1. E_2 digunakan untuk menentukan arah dari edge-edge tersebut. Buat $E_t = E_1 \cup E_2$. Adapun potongan programnya adalah seperti di bawah ini.

```
if Edge[I,J] <> 0 then
   begin
     EdgeE1_S1[I,J] :=  Edge[I,J];
     EdgeE1_S2[I,J] :=  Edge[I,J];
     EdgeE2_S1[I,J] :=  Edge[I,J];
     EdgeE2_S2[I,J] :=  Edge[I,J];
     Arc_In[I] := Arc_In[I] + (EdgeE1_S1[I,J] +
                  EdgeE2_S1[I,J]);
     Arc_Out[I] := Arc_Out[I] + (EdgeE1_S2[I,J] +
                   EdgeE2_S2[I,J]);
     Etotal :=  Arc_In[I] + Arc_Out[I]
```

2. Kemudian ditentukan vertek-vertek yang *indegree*nya tidak sama dengan *outdegree*. Adapun caranya adalah dengan mengisikan b_1 dengan jumlah arc pada A yang menuju vertek dikurangi dengan jumlah arc pada A yang meninggalkan vertek.

```
for J := 1 to Vertice do
  begin
```

```
    if Arc[I, J] > 0 then
      Arc_Out[I] := Arc_Out[I] + Arc[I,J]
    else if Arc[I, J] < 0 then
      Arc_In[I] := Arc_In[I] + (-1 * Arc[I,J])
  end;
VertexB1[I] := Arc_In[I] - Arc_Out[I];
VertexB2[I] := Arc_In[I] - Arc_Out[I]
```

VertexB1 mewakili b_I , jika VertexB1 bernilai positif ini berarti jumlah *indegree* lebih banyak dari pada jumlah *outdegree*. Maka agar sama arc yang meninggalkan vertek ditambahkan sebanyak selisihnya.

3. Dengan prosedur Shortestpath dicari semua path terpendek yang menghubungkan vertek yang VertexB1 positif dengan vertek yang VertekB2 negatif. Kemudian dibuatkan *copy* tambahan arc sabanyak selisihnya. Tapi sebelumnya identifikasikan U kosong, dan M sama dengan A. Setiap *copy* tambahan dimasukkan ke dalam M. Seperti terlihat pada potongan program di bawah ini.

```
EdgeU[I, J] := 0;
ArcM[I, J] := Arc[I, J];
ArcM[Start,Rute1[I]]:=ArcM[Start,Rute1[I]]+VertexB1[Start];
ArcM[Rute1[I],Start]:=ArcM[Rute1[I],Start]-VertexB1[Start];
```

4. Untuk mengisi U adalah dengan melihat sisa edge yang tidak berubah menjadi arc. Seperti potongan program di bawah ini.

```
if (MatrixG[I, J] <> 0) and (Edge[I, J] <> 0) then
  EdgeU[I, J] := Edge[I, J]
```

Output yang dihasilkan dari prosedur Inoutdegree ini adalah berupa mixed graf $G' = (V, E', A')$, yang nantinya menjadi input dari prosedur Evenparity dan prosedur Largecycle.

4.4.1.3. Evenparity

Prosedur Inoutdegree akan mengasilkan graf yang *indegree* sama dengan *outdegre*, efek dari penambahan arc untuk menyamakan *in-outdegree*nya tidak bisa mempertahankan degree genap pada vertek yang diberi tambahan arc tersebut.

Prosedur Evenparity akan mengubah *parity* dari vertek ganjil, prosedur Evenparity tidak akan mengubah sifat bahwa *indegree* sama dengan *outdegree* pada setiap vertek.

1. Mula-mula adalah mengidentifikasi vertek yang mempunyai *degree* ganjil ke dalam V',

```
if Temp[I] <> 0 then
  begin
    OddG[I] := 1;
    V1 := V1 + 1
  end
```

dan isikan M dengan M', dan U dengan U'.

2. Dimisalkan $M'' \subseteq M$ yang merupakan tambahan dari arc dan edge yang sudah berarah yang dihasilkan dari prosedur Inoutdegree.

```
ArcM2[Start, Rute1[I]] := VertexB1[Start];
ArcM2[Rute1[I], Start] := -VertexB1[Start];
```

3. Tahap berikutnya adalah memanggil prosedur Adjustcycle, prosedur ini mengidentifikasi siklus yang terdiri dari lintasa alternatif pada M'' dan U, dengan masing-masing lintasan berakhir pada setiap ujung dari sebuah vertek ganjil. Lintasan-lintasan pada M'' akan dibentuk tanpa memperhitungkan arah arc dan

edge yang berarah. Karena siklus tersebut akan diduplikasi atau didelete, dan edge pada siklus akan menjadi terarah. Hal ini diperoleh karena parity dari vertek ganjil telah berubah, sedangkan indegree sama dengan outdegree untuk setiap vertek. Di bawah ini akan sisajikan prosedur Adjustcycle.

```
while V1 <> 0 do
  begin
    I := 0;
    repeat
      I := I + 1;
      V := I;
      Vs := V;
    until OddV1[I] <> 0;
    while OddV1[V] <> 0 do
      begin
        V1 := V1 - 1;
        repeat
          for J := 1 to Vertice do
            begin
              if V <> J then
                begin
                  If ArcM2[V, J] > 0 then
                    begin
                      ArcM1[V, J] := ArcM1[V, J] + 1;
                      ArcM1[J, V] := ArcM1[J, V] - 1;
                      OddV1[V] := 0;
                      V := J
                    end;
                  if ArcM2[V, J] < 0 then
                    begin
                      ArcM1[V, J] := ArcM1[V, J] - 1;
                      ArcM1[J, V] := ArcM1[J, V] + 1;
                      OddV1[V] := 0;
                      V := J
                    end
```

```
            end
        end;
    until OddV1[V] <> 0;
    V1 := V1 - 1;
    repeat
      for J := 1 to Vertice do
        begin
          if V <> J then
            begin
              if EdgeU1[V, J] <> 0 then
                begin
                  ArcM1[V, J] := -1;
                  ArcM1[J, V] := 1;
                  EdgeU1[V, J] := 0;
                  EdgeU1[J, V] := 0;
                  OddV1[V] := 0;
                  V := J
                end
            end
        end
    until (OddV1[V] <> 0) or (V = Vs)
  end
end
```

Output dari prosedur Evenparity merupakan hasil akhir dari program postman problem dengan algoritma Edmonds and Johnson, yaitu merupakan masukan M' dan U' yang memenuhi kriteria yang sama dengan masukkan M dan U dari prosedur Inoutdegree, sehingga vertek-vertek dari (V, U', M') merupakan degree genap.

4.4.2. Algoritma Kedua Untuk Mixed Postman

Suatu pendekatan alternatif terhadap mixed *postman problem* merupakan kebalikan dari pendekatan Edmonds and Johnson. Pertama *indegree* dan *outdegree*

dari setiap vertek dibuat sama. Kemudian *degree* dari setiap vertek dibuat genap, sambil mempertahankan karakteristik untuk setiap vertek *indegree* sama dengan outdegree.

4.4.2.1 Inoutdegree

Untuk membuat *indegree* sama dengan *outdegree* digunakan prosedur Inoutdegree seperti yang sudah dijelaskan pada algoritma Edmonds and Johnson di atas. Karena itu, tidak akan dijelaskan lagi lebih lanjut.

4.4.2.2. Largecycle

Prosedur Largecycle mengambil input dari hasil prosedur Inoutdegree

1. Pertama-tama, identifiakasi vertek yang ber*degree* ganjil pada graf $G' = (V, U)$. Dengan menggunakan program untuk mengidentifikasi vertek ganjil seperti di bawah ini.

```
Temp[I] := Odd[I] mod 2;
if Temp[I] <> 0 then
  begin
    OddG[I] := 1;
    V1 := V1 + 1
  end
else
     OddG[I] := 0
```

   Bila OddG[I] bernilai 1 berarti vertek mempunyai *degree* ganjil, begitu sebaliknya.

2. Dengan prosedur Shortestpath dicari semua lintasan terpendek antara vertek ber*degree* ganjil pada graf $G' = (V, E)$.

```
Repeat
  VFlag[Start] := True;
  For i := 1 To Vertice Do
```

```
      Begin
        If EdgeOrArc Then isEdge := (MatrixG[Start,I] <> 0)
        Else isEdge:=((MatrixG[Start,I]<>0)and(Arc[Start,I]>=
                    0));
        If isEdge Then
          Begin
            If i = Finish Then
              Begin
                Rute1[l] := i;
                Tambah(Awal,Akhir,Bantu,Rute1,Jarak1 +
                       MatrixG[Start,i],l,Finish);
                If Bantu^.Berikut <> Nil Then
                  Begin
                    Repeat
                      Hapus(Awal,Akhir);
                    Until Akhir = Bantu;
                  End
              End
            Else
              Begin
                If VFlag[i] <> True Then
                  Begin
                    Rute1[l] := i;
                    Tambah(Awal,Akhir,Bantu,Rute1,
                    Jarak1 + MatrixG[Start,i],l,Finish);
                  End
              End
          End
      End;
    Jarak1 := Awal^.Jarak;
    L := Awal^.Vertex;
    Rute1 := Awal^.Rute;
    Start := Awal^.Rute[L];
    I := L + 1;
    Hapus_Depan(Awal,Akhir);
  until Awal = Nil;
```

3. Lakukan *minimum-cost matching* dari vertek ganjil dengan menggunakan jarak lintasan terpendek. Masukkan edge yang digunakan dalam *matching* ke dalam *U*. Kemudian lakukan traversal dari arc dan edge berarah pada *M* dan edge tidak berarah pada *U*. Di bawah ini adalah potongan program untuk menjelaskan hal di atas.

```
repeat
  Inc(I);
  If EdgeU[Start,Rute1[I]] <> 0 then
    begin
      EdgeU[Start,Rute1[I]] := EdgeU[Start,Rute1[I]] + 1;
      EdgeU[Rute1[I],Start] := EdgeU[Rute1[I],Start] + 1
    end;
  Start := Rute1[I];
until Start = Finish;
```

Program di atas mengakhiri prosedur Largecycle, dan hasilnya merupakan *output* dari algoritma Kedua untuk Mixed Postman.

BAB V

ANALISA HASIL UJI COBA DAN PEMBAHASAN

Setelah sistem perangkat lunak dikembangkan dan diuji hasilnya perlu dievaluasi kembali. Pengevaluasian meliputi evaluasi pada algoritma, evaluasi prosedur, dan evaluasi terhadap kelebihan, analisa hasil dan kekurangan sistem yang dikembangkan.

5.1. Pengoperasian Perangkat Lunak

Perangkat lunak postman problem ini dibuat dengan bahasa pemrograman **Turbo Pascal** , dan untuk menunjang tampilan agar bisa dijalankan pada sistem operasi **windows**, input dan outputnya dipadukan dengan bahasa pemrograman **Borland Delphi.**

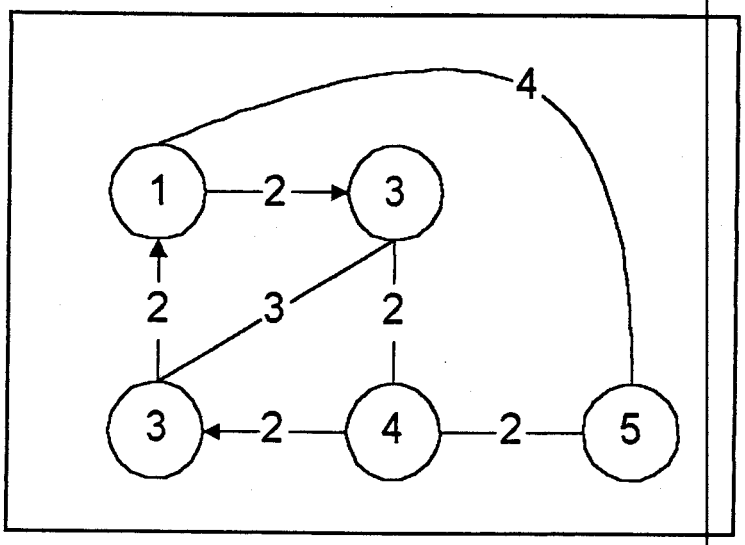
5.1.1. Input Graf

Input dari perangkat lunak ini adalah berupa graf yang diinputkan oleh user. Tahap awal adalah menginputkan vertek-verteknya, kemudian vertek-vertek tersebut dihubungkan sesuai dengan peta postman problem.

Contoh *input* yang ditampilkan di bawah ini akan dapat menerangkan kelebihan dan kekurangan dari algoritma yang dipakai dalam Tugas Akhir ini. Oleh sebab itu kami menggunakan dua algoritma, yaitu algoritma Edmonds and Johnson dan

algoritma lain (dalam pembahasan pada bab sebelumnya disebut dengan algoritma Kedua untuk *Mixed Postman*) yang prinsip kerjanya merupakan kebalikannya. Sehingga kekurangan dari algoritma yang satu ditutupi oleh algoritma yang lain.

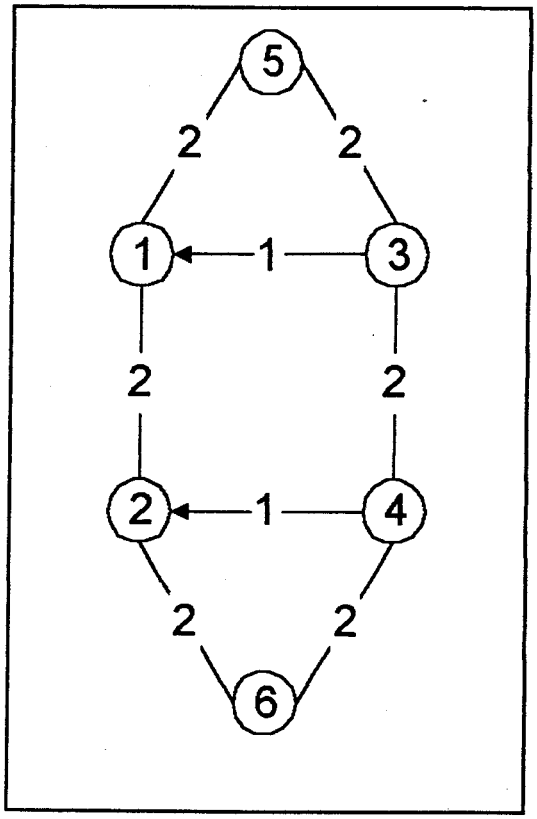
Seperti contoh pertama di bawah ini :



gambar 5.1

Input graf Mixed Postman 1

untuk algoritma Kedua merupakan kasus buruk, karena menghasilkan total jarak yang tidak optimum. Sedangkan untuk contoh kedua di bawah ini :



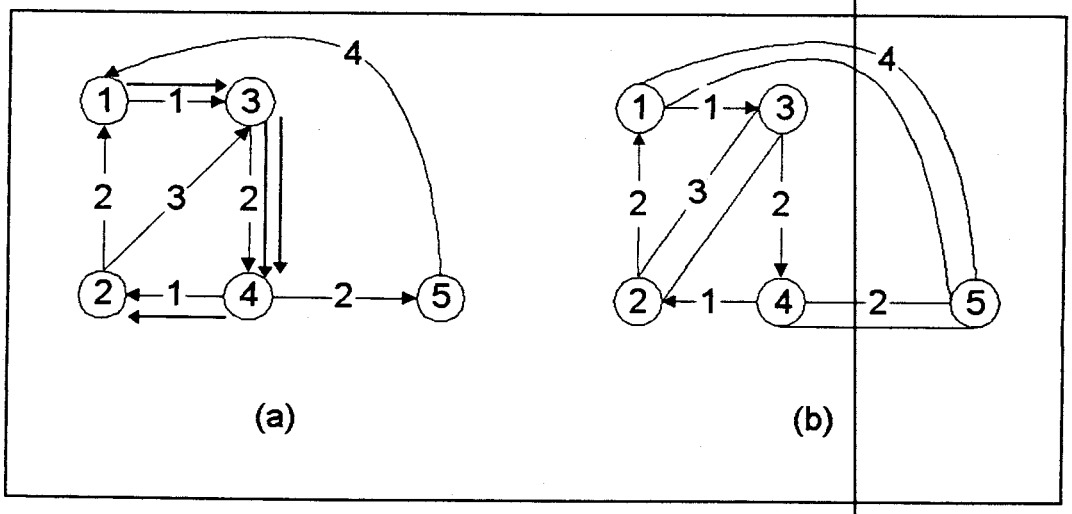
gambar 5.2

Input graf Mixed Postman 2

merupakan kasus terburuk algoritma Edmonds and Johnson, sedang dengan algoritma Kedua hasilnya lebih mendekati optimum.

5.1.2. Output Program

Untuk *output* dari contoh pertama adalah sebagai berikut :



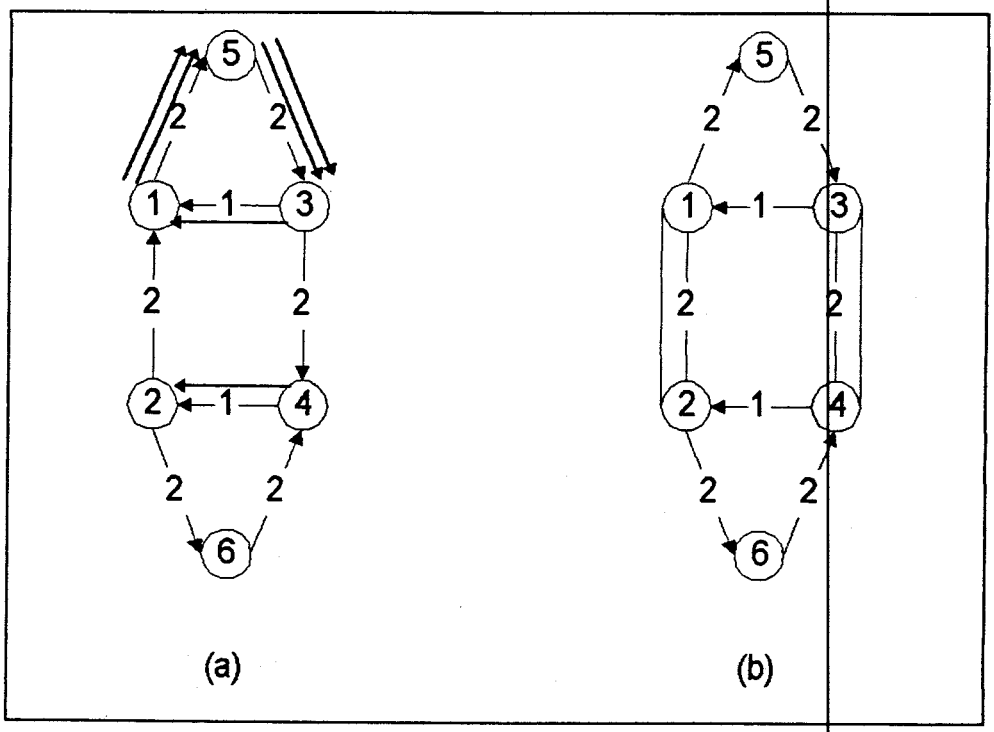
gambar 5.3

(a) Hasil dengan algoritma Edmonds and Johnson

(b) Hasil dengan algoritma Kedua untuk Mixed Postman

jelas terlihat perbedaan total jarak yang ditempuh oleh kedua algoritma di atas, total jarak yang dihasilkan dari algoritma Edmonds and Johnson yaitu 21 lebih kecil dari hasil algoritma Kedua yaitu 24.

Untuk contoh kedua adalah sebagai berikut :



gambar 5.4

(a) Hasil dengan algoritma Edmonds and Johnson

(b) Hasil dengan algoritma Kedua untuk Mixed Postman

total jarak yang dihasilkan algoritma Edmonds and Johnson yaitu 24 lebih besar dari hasil algoritma Kedua yaitu 18.

Dilihat dari kedua contoh di atas sebuah algoritma belum menjamin menghasilkan hasil yang diinginkan, maka dalam Tugas Akhir ini digunakan dua algoritma. Dari *output* yang didapat dipilih total jarak yang terpendek.

5.2. Analisa Hasil

Algoritma yang digunakan dalam merancang sistem perangkat lunak telah dijelaskan pada bab 3. Algoritma tersebut merupakan algoritma yang menurut kami paling efisien dan efektif. Hal ini didasarkan pada pengalaman dan pengamatan selama perancangan dan pengembangan sistem ini. Algoritma-algoritma ini dijelaskan pada sub bab berikut.

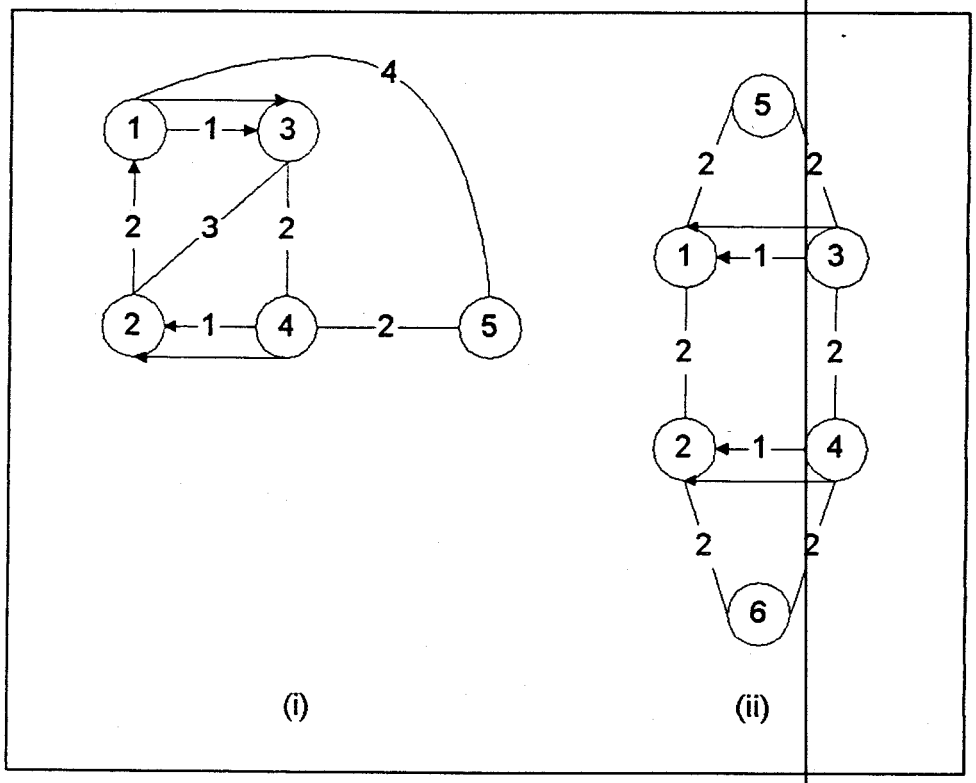
5.2.1. Algoritma Edmonds dan Johnson untuk Mixed Postman

Pada algoritma ini terdapat tiga tahap pengerjaan. Yang pertama adalah menjadikan vertek-vertek yang mempunyai degree ganjil menjadi genap, tahap pertama ini diselesaikan dengan algoritma Evendegree. Yang kedua adalah dengan algoritma Inoutdegree, menjadikan *indegree* sama dengan *outdegree*. Dan yang ketiga dengan algoritma Evenparity, membuat degree pada setiap vertek genap dengan tetap mempertahankan *indegree* sama dengan *outdegree*.

Semua algoritma yang mendukung algoritma Edmonds and Johnson akan dijelaskan lebih lanjut di bawah ini.

5.2.1.1. Algoritma Evendegree

Pada contoh input pertama vertek 1, 2, 3, dan 4 diidentifikasi ber*degree* ganjil, dan pada contoh input kedua vertek 1, 2, 3, dan 4 juga ber*degree* ganjil. Dengan algoritma Evendegree vertek-vertek yang ber*degree* ganjil akan menjadi genap. Seperti pada gambar 5.5 di bawah.



gambar 5.5

Hasil dari algoritma Evendegree

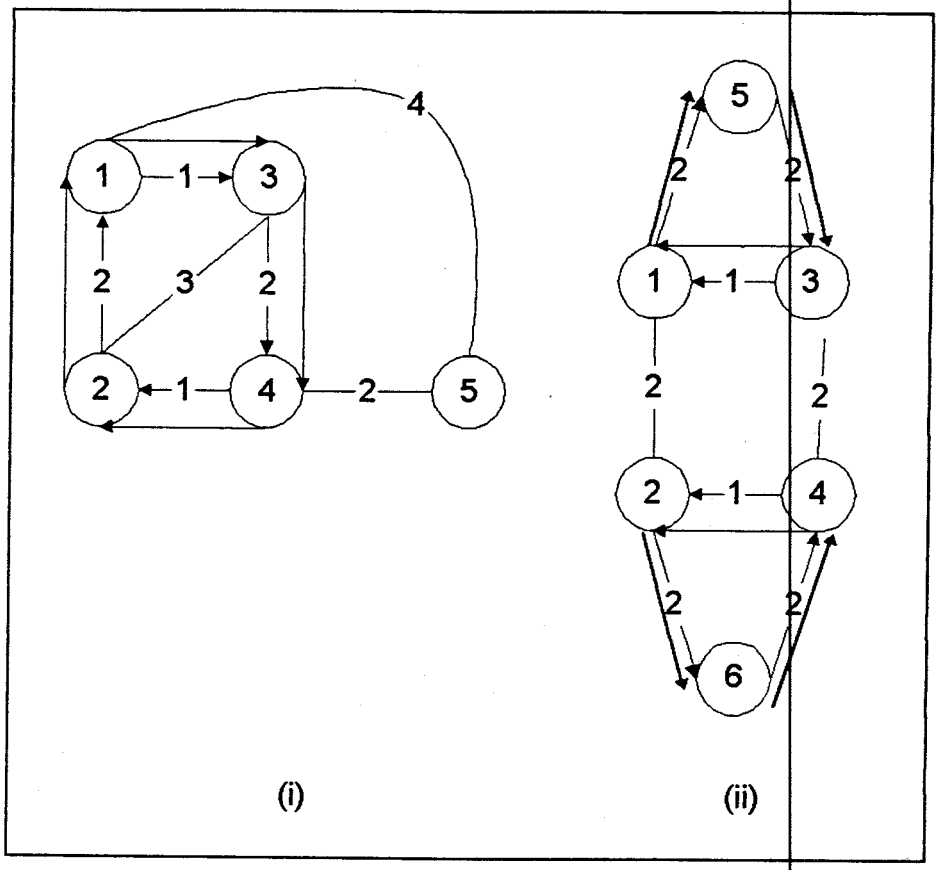
Semua verteknya sudah mempunyai *degree* ganjil, tapi kekurangannya adalah bahwa *indegree* dan *outdegree*nya belum sama. Hal ini akan diproses pada algoritma berikutnya.

5.2.1.2. Algoritma Inoutdegree

Algoritma Inoutdegree menghasilkan *output* yang *indegree* sama dengan *outdegree*. Dengan mengidentifikasi vertek *input* yang dihasilkan dari algoritma Evendegree, jumlah arc yang masuk dikurangi jumlah arc yang keluar dapat diketahui

berapa jumlah arc yang harus ditambahkan sehingga didapat *indegree* sama dengan *outdegree*. Tetapi dengan adanya tambahan arc ini dapat mempengaruhi *degree* dari vertek menjadi ganjil. Maka arc yang menjadi tambahan perlu disimpan dalam suatu variabel tertentu, dalam program ini variabel ini adalah ArcM2. Ini akan sangat menentukan untuk proses selanjutnya pada algoritma Evenparity.

Pada algoritma ini prosedur Shortestpath juga sangat berperan di dalam menentukan pasangan vertek yang mempunyai *indegree* yang lebih besar dengan vertek yang mempunyai *outdegree* yang lebih besar. Arah arc adalah dari vertek yang mempunyai *indegree* lebih besar ke vertek yang mempunyai *outdegree* yang lebih besar. Hasil algoritma ini dapat dilihat pada gambar 5.6.



gambar 5.6

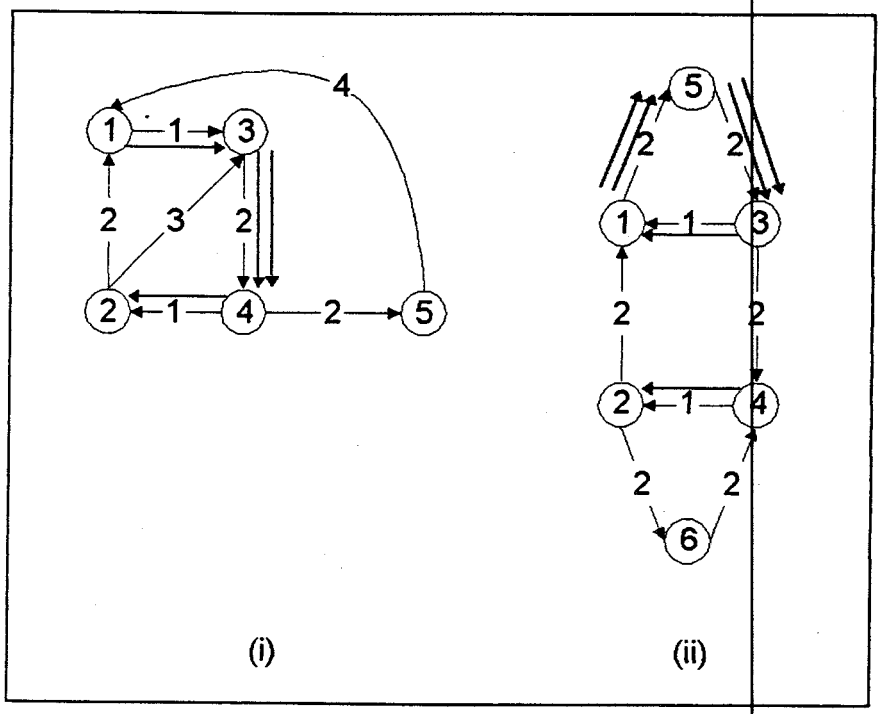
Hasil algoritma Inoutdegree

5.2.1.3. Algoritma Evenparity

Output Inoutdegree akan menyababkan kembali beberapa vertek mempunyai *degree* ganjil. Dari arc tambahan pada variabel ArcM2 akan diolah pada algoritma Evenparity agar semua vertek mempunyai *degree* genap dengan tetap mempertahankan *indegree* sama dengan *outdegree*.

Dimulai dari vertek ganjil pertama, apakah variabel ArcM2 mengandung arc masuk atau arc keluar. Jika arc masuk maka arc yang menjadi tambahan di algoritma

Inoutdegree didelete, kalau sebaliknya maka arc tersebut akan ditambahkan satu. Demikian juga dengan edge yang berada pada jalur ini akan berubah menjadi edge yang berarah. Seperti pada gambar 5.7 merupakan hasil dari algoritma Evenparity.



gambar5.7

Hasil algoritma Evenparity

Hasil akhir dari algoritma Evenparity adalah merupakan hasil akhir dari algoritma Edmonds and Johnson secara keseluruhan.

5.2.2. Algoritma Kedua untuk Mixed Postman

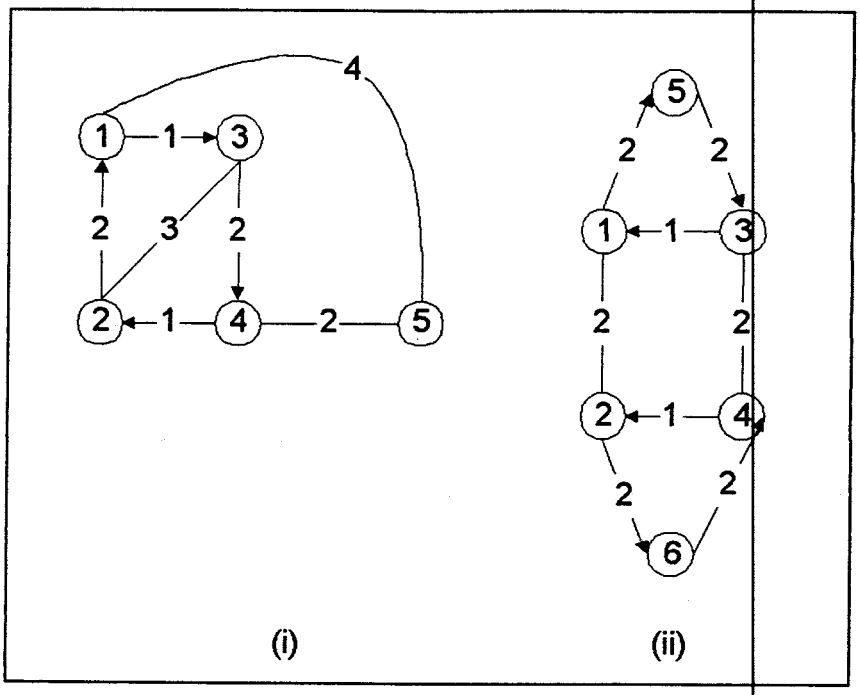
Sistem kerja dari algoritma Kedua ini marupakan kebalikan dari algoritma Edmonds and Johnson. Dimana proses pertama adalah menjadikan *indegree* sama dengan *outdegree*, setelah itu baru menggenapkan setiap vertek yang mempunyai degree ganjil, dengan tetap mempertahankan *indegree* sama dengan *outdegree*nya.

Di bawah ini akan dijelaskan kedua algoritma yang mendukung algoritma Kedua untuk *Mixed Postman.*

5.2.2.1. Algoritma Inoutdegree

Sistem dari algoritma Inoutdegree disini sama dengan sistem algoritma Inoutdegree pada algoritma Edmonds and Johnson di atas. Perbedaannya adalah pada inputnya, yaitu langsung dari user sedangkan kalau pada algoritma Edmonds and Johnson, *input* dari algoritma Inoutdegreenya adalah dari *output* Evendegree.

Adapun hasil dari algoritma ini adalah seperti terlihat pada gambar 5.8.



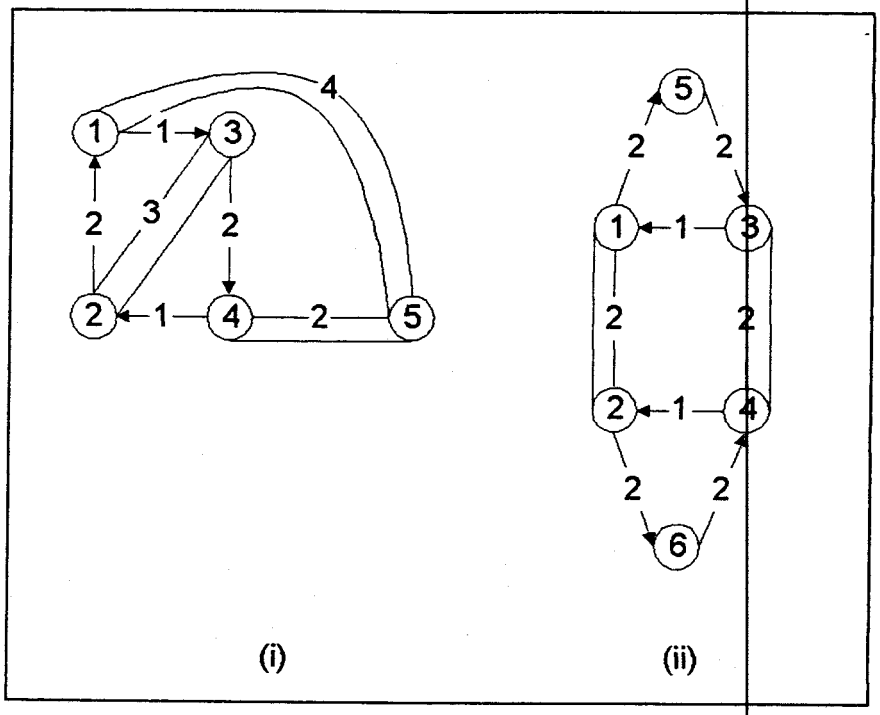
gambar 5.8

Hasil algoritma Inoutdegree pada algoritma Kedua

*Output*nya akan menjadi *input* untuk algoritma selanjutnya yaitu algoritma Largecycle.

5.2.2.2. Algoritma Largecycle

Algoritma Largecycle ini akan menduplikatkan setiap edge yang berapa pada pesangan vertek yang ber*degree* ganjil, sehingga setiap vertek akan mempunyai *degree* genap. Karena yang mengalami perubahan hanya edge saja maka *indegree* sama dengan *outdegree* dari hasil algoritma Inoutdegree akan tetap tanpa mengalami perubahan.



gambar 5.9

Hasil algoritma Largecycle

Gambar 5.9 merupakan hasil akhir dari algoritma Largecycle sekaligus merupakan hasil dari algoritma Kedua untuk Mixed Postman.

BAB VI

PENUTUP

6.1. Kesimpulan

Setelah sistem direalisasikan diuji dan diamati dapat diperoleh beberapa kesimpulan :

1. Program *postman problem* sangat membantu dalam menentukan lintasan yang mesti dilalui oleh tukang pos dalam melakukan tugasnya, sehingga dapat lebih mengefisienkan waktu perjalanan, karena lintasan yang dihasilkan merupakan pendekatan hasil yang optimum.
2. Dalam kasus tertentu algoritma Edmonds and Johnson menghasilkan rute lintasan yang buruk dalam artian liantasan yang dihasilkan tidak merupakan hasil yang optimum. Ini merupakan kelemahan algoritma Edmonds and Johnson, tapi dengan tambahan algoritma lain yang sistem kerjanya kebalikan dari algoritma Edmonds and Johnson hal ini dapat diatasi. Algoritma ini disebut algoritma Kedua untuk Mixed Postman.
3. Untuk dua vertek yang sama, mempunyai lintasan langsung yang lebih dari satu tidak dapat diinputkan langsung karena dengan menggunakan array hanya tercatat satu lintasan, untuk menutupi hal ini antara dua vertek ini diberi tambahan vertek

lagi. Begitu juga halnya bila terdapat *loop* pada satu vertek, harus ditambah satu vertek lagi, agar tidak terjadi *loop*.

4. Algoritma Evendegree menyebabkan graph adalah graph Euler. Semua verteknya mempunyai degree genap yang merupakan syarat graph Euler, tapi belum tentu merupakan digraph Euler, karena *indegree* belum sama dengan *outdegree*.
5. Algoritma Inoutdegree mengasilkan *indegree* sama dengan *outdegree*, tapi menyebabkan degree pada verteknya menjadi ganjil.
6. Algoritma Evenparity dan Largecycle dapat menyelesaikan permasalahan di atas yaitu menghasilkan graph dan digraph Euler. Karena setelah algoritma Inoutdegree, algoritma Evenparity dan Largecycle menjadikan vertek-verteknya mempunyai *degree* genap dengan tetap mempertahankan *indegree* sama dengan *outdegree*.

7.2. Saran

1. Pengembangan lebih lanjut algoritma untuk *Mixed Postman* akan menjadi hal yang menarik, algoritma yang ada masih memiliki kelemahan-kelemahan sehingga dalam implementasi *postman problem* dibutuhkan lebih dari satu algoritma agar bisa saling menutupi kelemahan.
2. Penerapan algoritma Edmonds and Johnson dapat diperluas dengan menampilkan simulasi perjalanan dari lintasan awal ke setiap lintasan yang dilalui. Agar dapat diketahui urutan rute perjalannya.

DAFTAR PUSTAKA

- V. K. Balakrishnan, *Introductory Discrete Mathematics*, Prentice Hall, Englewood Cliffs, New Jersey, 1989.
- C. L. Liu, *Elements of Discrete Mathematics*, Second Edition, McGraw-Hill International Edition, 1986.
- Greg N. Frederickson, *Approximation Algorithms for Some Postman Problem*, Journal of the Association for Computing Machinery, 1979.
- Edmonds, J., *The Chinese Postman Problem*, Suppl., 1965.
- Edmonds, J. and Johnson, E. L., *Matching, Euler Tours and The Chinese Postman*, Math. Programming, 1973.
- Floyd, R., *Shortest Path*, Comm. ACM, 1962.
- Theresia M. H., *Pengantar Graf*, University Press IKIP Surabaya, 1992.
- Aaron M. Tenenbaum , Moshe J. Augenstein, *Data Structures Using Pascal*, Prentice-Hall, Inc., Englewood Cliff, New Jersey, 1981.
- Ir. P. Insap Santosa, Msc., *Struktur Data menggunakan Turbo Pascal 6.0*, Edisi Pertama, Andi Offset Yogyakarta, 1992.
- Ir. P. Insap Santosa, Msc., *Pemrograman Pascal tingkat lanjut*, Edisi Pertama, Andi Offset Yogyakarta, 1989.
- Wozniewicz dan Shammas, *Teach Yourself Delphi in 21 Days*, First Edition, Sams Publishing, Indianapolis, 1995.

- Joko Pramono, *Belajar Sendiri Delphi '95*, PT Elex Media Komputindo, Jakarta, 1996.
- Joko Pramono, *Contoh Penggunaan Rutin-rutin Borland Delphi*, PT Elex Media Komputindo, Jakarta, 1996.

A. PENGOPERASIAN PERANGKAT LUNAK

Dalam pengoperasian perangkat lunak terdapat beberapa hal yang harus diperhatikan di antaranya :

1. Kebutuhan Perangkat Keras dan Lunak

Perangkat lunak dikembangkan dalam sistem operasi Windows yang berbasis grafis sehingga membutuhkan perangkat keras dengan kriteria sebagai berikut :

- IBM PC-AT 386-DX atau *clone*-nya dan versi yang lebih baru
- *Monitor* VGA dengan kartu antarmuka ber-*memory* minimal 1 mega byte
- *Memory on Board* minimal 4 mega byte
- *Space Harddisk* minimal 40 mega byte

dan kebutuhan perangkat lunaknya :

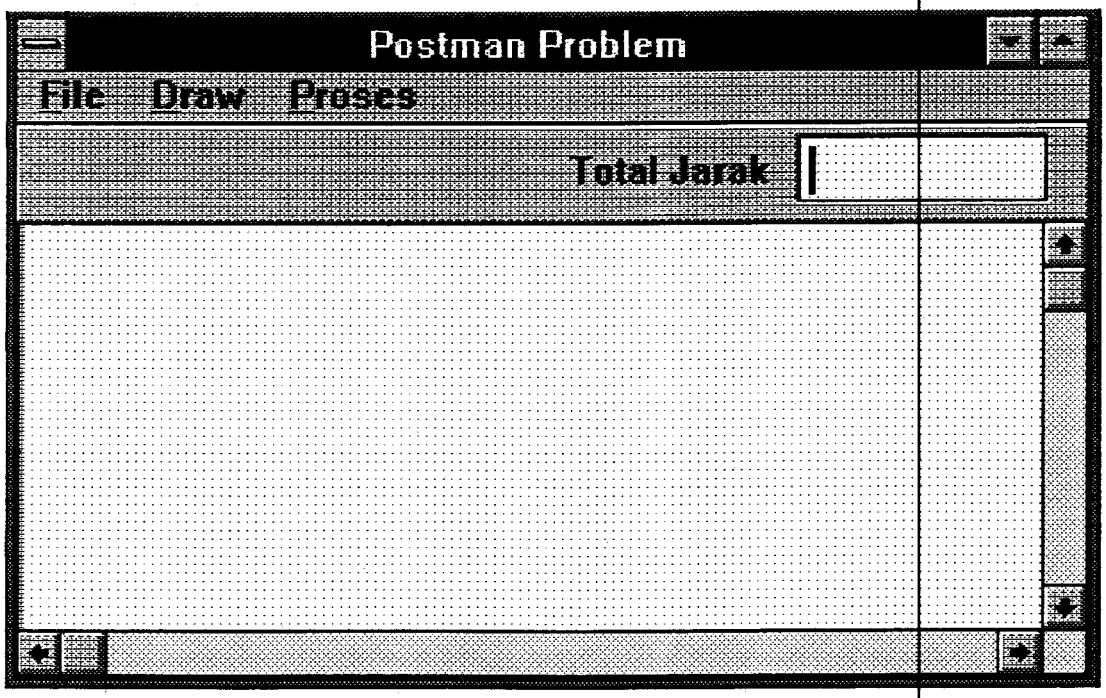
- Sistem Operasi *Windows 3.1* atau versi yang lebih baru
- File sistem perangkat lunak **POSTMAN.EXE**

Resister sistem ini pada Windows dilakukan dengan membuat *group* tersendiri dengan nama yang boleh ditentukan secara bebas. Tambahkan *program item* dengan mengisikan file aplikasi-nya dengan nama file dan working direktory-nya. Misalnya: file-file sistem perangkat lunak disimpan pada direktori **C:\TPOST**, maka nama file yang dipilih **C:\TPOST\POSTMAN.EXE** dan *working directory* diisi dengan **C:\TPOPST**.

1. Cara Pengoperasian Perangkat Lunak

Jalankan perangkat lunak dengan menekan *icon*-nya pada daftar-daftar aplikasi dalam *File Manager Wndows* atau dengan mengeksekusinya melalui pilihan menu *File-Run-Browse.*

Tampilan dari perangkat lunak POSTMAN.EXE akan terlihat seperti gambar A.1 di bawah ini.

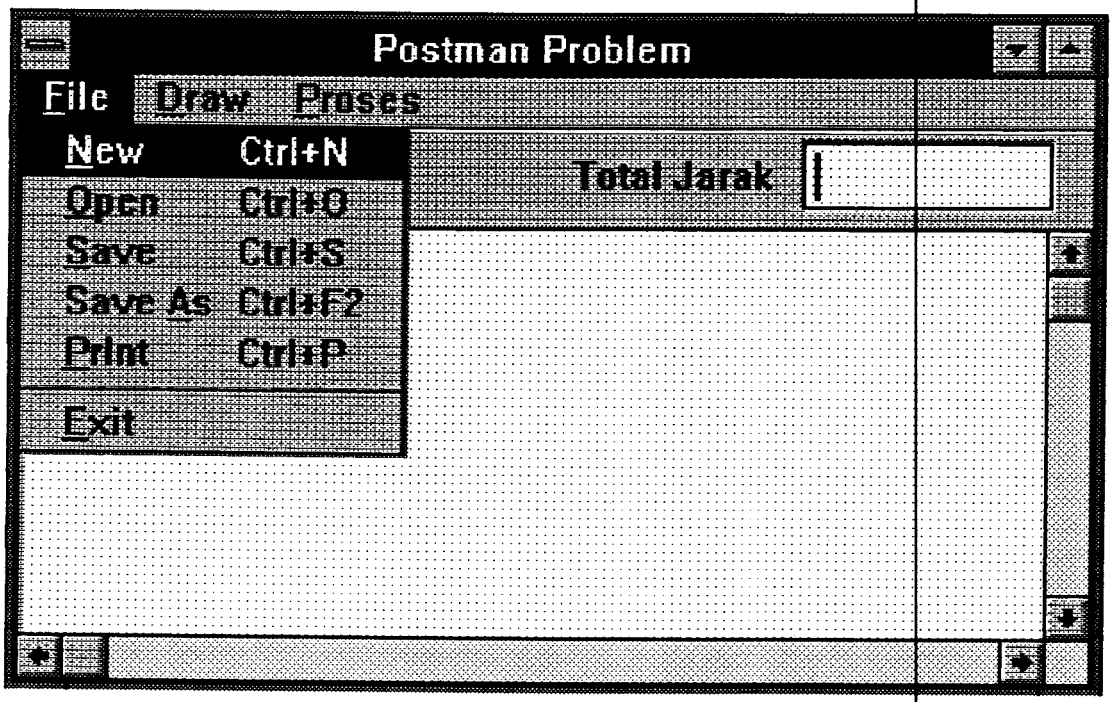


Gambar A.1

Tampilan Windows Postman Problem

3. Mempelajari Menu

3.1. Menu File



Gambar A.2

Menu File

Pilihan-pilihan dalam menu **File** antara lain untuk membuat gambar peta perjalanan tukang pos baru, memanggil gambar peta (berupa file) yang sudah ada yang tersimpan dalam *disk*, menyimpan gambar peta input dan mencetak hasil ke printer. Jika memanggil sebuah file gambar peta secara otomatis gambar peta tersebut akan ditempatkan dalam editor. Untuk menyimpan bisa ditempatkan disembarang tempat

dengan sembarang nama. Gambar A.2 menunjukkan pilihan-pilihan yang ada pada menu **File.**

Pilihan New (Ctrl-N)

Gambar peta yang ada di editor akan dihapus dan gambar peta yang akan digambar dianggap sebagai gambar peta yang baru. Nama akan dituliskan pada saat menyimpan dengan nama sembarang dan diikuti dengan .POS.

Pilihan Open (Ctrl-O)

Pilihan ini digunakan untuk memanggil file gambar dari dalam disk ke dalam editor. Nama file juga bisa berisi nama disk atau direktori di mana gambar itu tersimpan.

Pilihan Save (Ctrl-S)

Pilihan ini digunakan untuk menyimpan gambar peta yang ada di dalam editor ke dalam *disk*. Jika gambar peta tersebut belum pernah tersimpan, maka akan ditanyakan nama apa yang akan diberikan pada gambar yang akan disimpan. Nama file akan diikuti dengan .POS.

Pilihan Save As (Ctrl-F2)

Digunakan untuk menyimpan gambar peta menjadi file yang baru atau mengganti nama file yang sudah ada dalam *disk*. Jika nama yang dipilih sudah ada dalam *disk*, akan muncul pertanyaan verifikasi penggantian nama file.

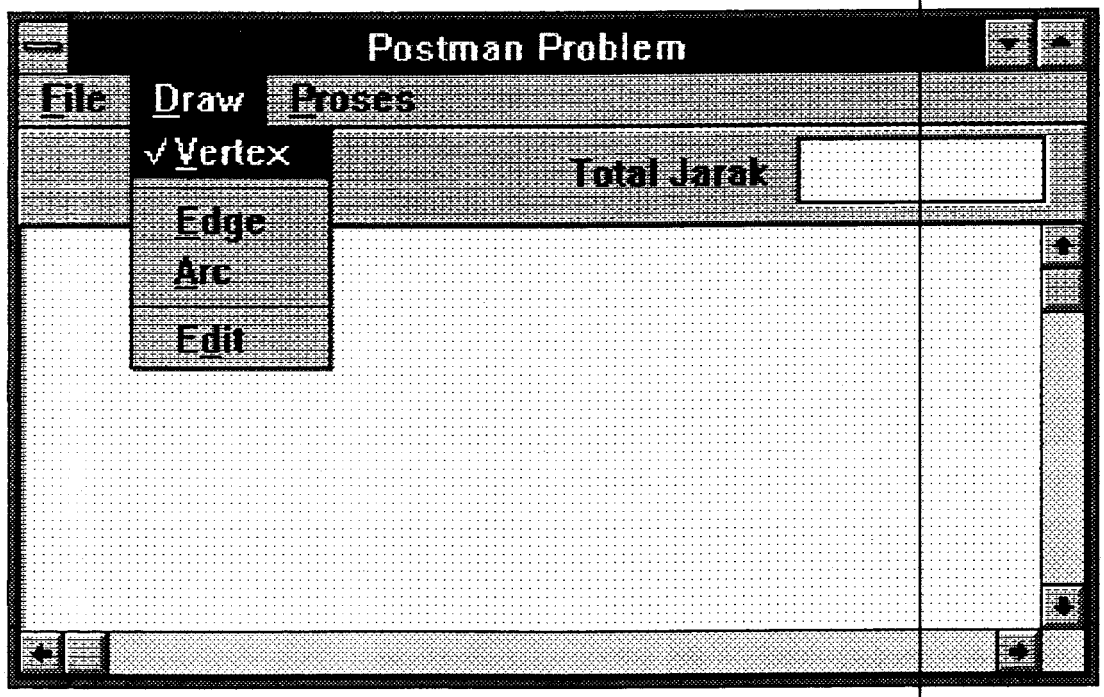
Pilihan Print (Ctrl-P)

Digunakan untuk mencetak gambar peta yang dihasilkan dari proses **Mixed1** atau **Mixed2** ke printer.

Pilihan Exit

Pilihan untuk ke luar dari program Postman Problem.

3.2. Menu Draw



Gambar A.3

Menu Draw

Pilihan dalam menu **Draw** digunakan untuk menggambar vertek, edge (garis tak berarah), dan arc (garis yang berarah). Serta berisi pilihan untuk mengganti arc menjadi edge atau sebaliknya.

Pilihan Vertex

Digunakan untuk membuat titik-titik yang dalam *postman problem* adalah merupakan persimpangan atau kantor pos. Dengan menggunakan mouse tentukan letak titiknya kemudian diklik sekali.

Pilihan Edge

Pilihan ini digunakan untuk menggambar garis yang menghubungkan dua titik, dengan meletakkan kusor dari mouse pada salah satu titik kemudian klik tombol sebelah kiri mouse sambil terus ditekan sampai garis terhubung dengan titik yang di tuju baru tombol mousenya dilepas.

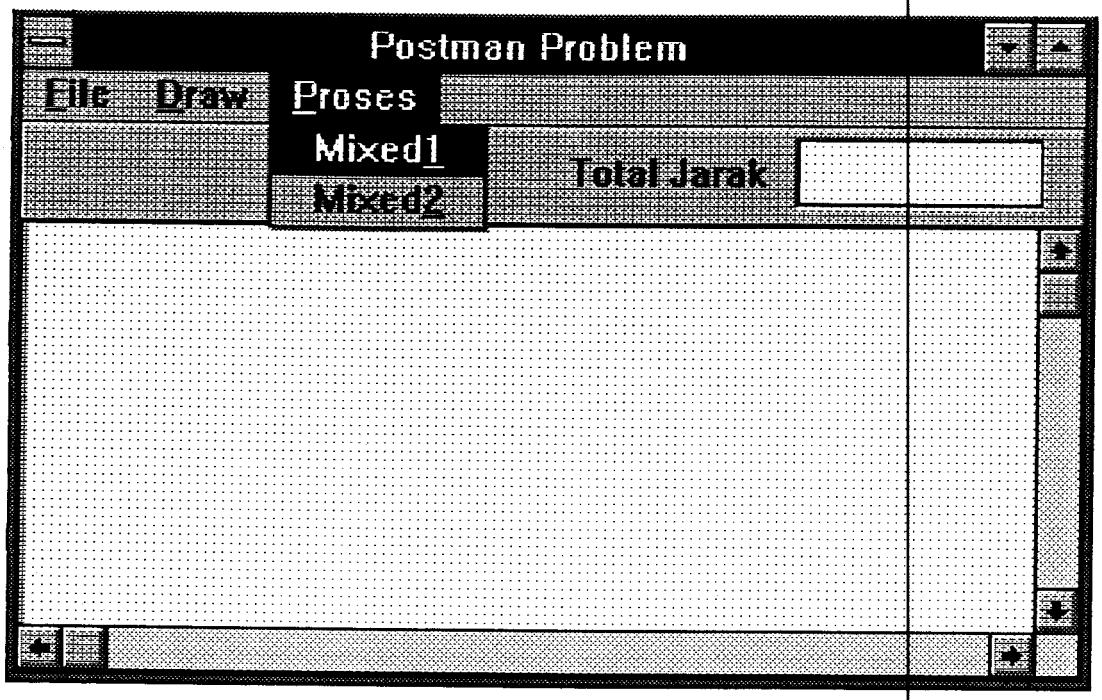
Pilihan Arc

Pilihan ini digunakan untuk membuat garis berarah yang menghubungkan dua titik. Prinsipnya sama dengan pilihan Edge, arahnya adalah dari titik yang awal kusor ke titik akhir kusor.

Pilihan Edit

Untuk mengganti edge dengan arc atau sebaliknya.

3.3. Menu Proses



Gambar A.4

Menu Proses

Pilihan-pilihan pada menu **Proses** adalah digunakan untuk memproses gambar peta *input* gambar peta perjalana tukang pos yang sebenarnya.

Pilihan Mixed1

Adalah pilihan untuk memproses gambar peta *input* dengan algoritma Edmonds dan Johnson yang pertama.

Pilihan Mixed2

Adalah pilihan untuk memproses ganbar peta *input* dengan algoritma Mixed Postman kedua.

USULAN TUGAS AKHIR
CS 1799

Nama Mahasiswa : Agus Ari Pramana
Nrp. : 2902600261
Dosen Pembimbing : DR. IR. Supeno Djanali

JUDUL TUGAS AKHIR :
MENENTUKAN LINTASAN TERPENDEK UNTUK POSTMAN PROBLEM DENGAN ALGORITHMA EDMONDS DAN JOHNSON

ABSTRAK:

Dalam tugas akhir ini akan dibuat perangkat lunak pada salah satu aspek riset operasional yaitu menentukan lintasan terpendek dari *Postman Problem* dengan menggunakan algorithma pendekatan dari perpaduan antara algorithma Edmonds dan Johnson dengan kebalikan dari algorithma tadi.

Algorithma Edmonds dan Johnson menyatakan bahwa setiap verteks dalam sebuah graph harus mempunyai jumlah edge yang genap untuk graph tak berarah. Dan untuk graph yang berarah setiap verteks harus mempunyai edge dan arc yang genap, serta jumlah masukan sama dengan keluaran. Jadi algorithma ini terdiri dari dua langkah yaitu, langkah pertama adalah mengubah graph asal sehingga setiap verteks memiliki derajat genap, dimana setiap arc dianggap sebagai edge. Langkah kedua membentuk masukan dan keluaran setiap verteks berjumlah sama.

Algorithma yang kedua adalah kebalikan dari algorithma diatas, yaitu langkah pertama yang dilakukan adalah langkah kedua dari algorithma Edmonds dan Johnson.

Setelah kedua algorithma di atas digunakan untuk menyelesaikan suatu *Postman Problem*, maka dipilih penyelesaian yang paling baik.

Surabaya, 14 Nopember 1995

Menyetujui,
Dosen Pembimbing

DR. IR. Supeno Djanali
NIP. 130 368 610

USULAN TUGAS AKHIR
CS 1799

AGUS ARI PRAMANA
290 260 0261
TEKNIK KOMPUTER ITS

I. JUDUL :

PENENTUAN LINTASAN TERPENDEK UNTUK POSTMAN PROBLEM DENGAN ALGORITMA EDMONDS DAN JOHNSON

II. LATAR BELAKANG

Riset operasi adalah salah satu bidang pengetahuan yang perkembangannya sangat pesat setelah ditemukannya teknologi komputer. Riset operasi diterapkan pada masalah-masalah mengenai bagaimana melaksanakan dan mengkoordinasi suatu operasi atau kegiatan, baik dalam bisnis, industri, ketentaraan, pemerintahan, dan lain sebagainya.

Salah satu bidang dari riset operasi adalah menentukan lintasan terpendek, yang bertujuan untuk meminimumkan biaya, sehingga sumber-sumber daya yang ada dapat dialokasikan dengan efisien dan menguntungkan. Permasalahan yang akan dibahas di sini adalah mendapatkan lintasan terpendek dari *Postman Problem*. Untuk menentukan

lintasan terpendek maka diperlukan algoritma tertentu sehingga diharapkan penyelesaian yang didapat adalah optimal.

Satu algoritma terkadang tidak dapat menentukan penyelesaian yang optimal dari *Postman Problem*, atau tidak mungkin menentukan hasil yang maksimal dari permasalahan yang ada, sehingga hasil yang didapat adalah merupakan pendekatan dari penyelesaian yang sebenarnya. Untuk itu algoritma-algoritma yang ada sedapat mungkin dipadukan untuk memperoleh penyelesaian yang paling memuaskan. Adapun algoritma tersebut adalah algoritma dari Edmonds dan Johnson yang dipadukan dengan kebalikan algoritma ini.

III. PERMASALAHAN

Untuk menentukan lintasan terpendek dari Postman Problem, efisiensi algoritma yang satu dengan yang lainnya tidak sama terhadap sebuah permasalahan. Maka untuk meningkatkan efisiensinya algoritma-algoritma yang ada dipadukan. Algoritma-algoritma tersebut adalah dari Edmonds dan Johnson dengan algoritma kebalikannya.

IV. TUJUAN

Membuat perangkat lunak untuk mendapatkan lintasan terpendek dari Postman Problem dengan algoritma perpaduan antara algoritma Edmonds dan Johnson dan algoritma kebalikannya.

V. METODOLOGI

1. Studi literatur
2. Analisa data
3. Membuat perangkat lunak
4. Mangadakan evaluasi
5. Dokumentasi dan penulisan naskah

VI. REFERENSI

1. Busacker, R. G., and Saaty, T. L., *Finite Graphs And Networks*, McGraw-Hill, New York, 1965.
2. Ford, L. R. and Fulkerson, D. R., *Flows in Networks*, Princeton U. Press, Princeton, N. J., 1962.
3. Garey, M. R., and Johnson, D. S., *Algorithms and Complexity: Recent Results and New Directions*, J. F. Traub, Ed., Academic Press, New York, 1976.
4. Lawler, E. L., *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart, and Winston, New York, 1976.
5. Hiller, Frederick S., and Lieberman, Gerald J., *Introduction To Operations Reseach*, Fifth Edition, McGraw-Hill Inc., New York, 1990.

VII. RELEVANSI

Perangkat lunak yang dibuat diharapkan dapat menyelesaikan Postman Problem secara efisien. Tentunya perangkat lunak ini nentinya dapat diaplikasikan pada bidang yang memerlukannya.

VIII. JADWAL KEGIATAN

LANGKAH	Bulan Ke					
	I	II	III	IV	V	VI
Studi Literatur						
Analisa Data						
Membuat Perangkat Lunak						
Mangadakan Evaluasi						
Dokumentasi Dan Penulisan Naskah						