MODUL 9 TUGAS BESAR HAMILTONIAN PATH

Kelompok C4

Gilbert Ng (13220032) **Yusuf Hafizh Sofwan (13220040)** **Jota Baret Tata (13220048)**

Bostang Palaguna (13220055) **Michael Tiroi (13220062)**

Asisten: Muhammad Morteza Mudrick (13219061)

EL2208 – Praktikum Pemecahan Masalah dengan C

Laboratorium Dasar Teknik Elektro - Sekolah Teknik Elektro dan Informatika ITB

Abstrak

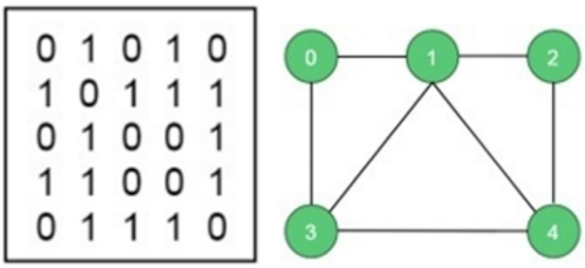
Graf adalah struktur diskret yang terdiri dari kumpulan vertex dan edges yang menghubungkan vertex tersebut[1]. Path adalah sekuens dari edges yang bermulai dari pada sebuah vertex pada graph dan menjelajahi dari satu vertex ke vertex lain di sepanjang edge pada graph. Hamiltonian path adalah path yang melewati setiap vertex tepat sekali. Pada modul ini praktikan akan bekerja dalam kelompok beranggotakan 4-6 orang untuk membuat program yang dapat mencari banyaknya hamiltonian path dan menampilkan semua kemungkinan path pada terminal serta menyimpan ke dalam file eksternal dari sebuah graf yang disimpan di sebuah file eksternal dalam representasi adjacency matrix.

Kata Kunci : hamiltonian path, adjacency matrix, hamiltonian cycle, kompleksitas waktu dan ruang.

1. Pendahuluan

1.1 Deskripsi Permasalahan

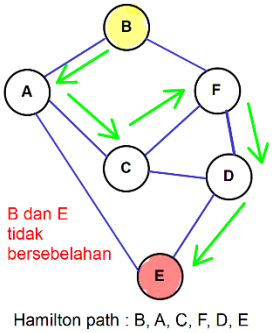
Praktikan akan diberikan data berupa *graph* yang direpresentasikan dalam bentuk *adjacency matrix*, yaitu suatu matriks berukuran jumlah titik x jumlah titik dengan 0 menandakan tidak ada jalur antartitik dan 1 menandakan ada jalur antartitik.



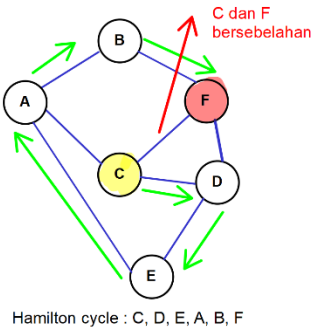
Gambar 1.1 : Contoh representasi *graph* dalam bentuk *adjacent matrix*

Praktikan diminta untuk membuat program yang dapat mencari semua kemungkinan *Hamilton Path* (jalur yang melewati semua titik tepat satu kali) dari *graph* yang diberikan.

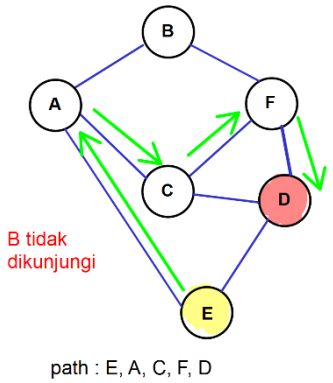
Hamiltonian path adalah sekuens vertex (*graph path*) pada graph yang mengunjungi setiap vertex tepat sekali. Kalau vertex terakhir dan vertex pertama saling bersebelahan (*adjacent*), maka path ini disebut dengan Hamilton cycle[5]. Berikut adalah contoh path yang merupakan Hamilton path, Hamilton cycle, dan bukan Hamilton path dari suatu graf.



Gambar 1.2 : Contoh *Hamilton Path yang bukan merupakan Hamilton cycle dari suatu graph*



Gambar 1.3 : Contoh *Hamilton Path yang sekaligus merupakan Hamilton cycle dari suatu graph*



Gambar 1.4 : Contoh *Path yang bukan merupakan hamilton path*

Adapun program diharapkan akan memiliki sifat-sifat berikut.

1. Memberi solusi valid ketika ada
2. Memberi semua kemungkinan solusi
3. *Time complexity* serendah mungkin
4. *Space complexity* serendah mungkin

Ketentuan input dari program adalah program menerima input berupa nama file eksternal yang isinya banyak titik dan *adjacency matrix* dari *graph*. Contoh isi file dan contoh input adalah sebagai berikut.

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

Gambar 1.5 : Contoh isi file eksternal sebagai input program. Baris pertama adalah bilangan bulat *n* yang menandakan banyaknya vertex pada graf, sedangkan *n* buah baris berikutnya berisi representasi graf dalam adjacency matrix.

Masukkan Graphs: file1.txt

Ketentuan output dari program terbagi menjadi beberapa macam. Yang pertama adalah program menampilkan jumlah *Hamilton Path* yang ada dan rute-rute *Hamilton Path* yang ada (apabila 0 akan ditampilkan 0 dan rutenya kosong). Yang kedua apabila input nama file salah, program akan menampilkan Error: file invalid!. Yang ketiga apabila file kosong, program akan menampilkan Error: file empty!.

Contoh tampilan program apabila dijalankan.

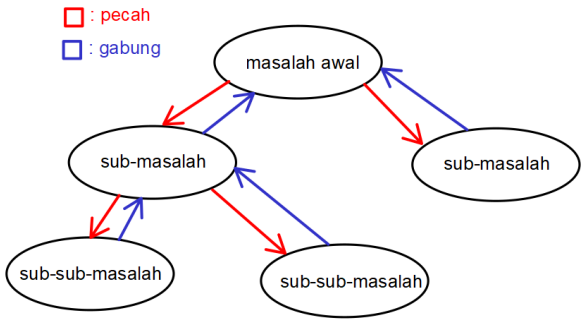
```
Masukkan Graphs: file1.txt

Banyak Hamilton Path: 20
Hamilton Path:
3-4-2-1-0
2-4-3-1-0
4-2-1-3-0
2-4-1-3-0
2-1-4-3-0
1-2-4-3-0
2-4-3-0-1
0-3-4-2-1
```

Gambar 1.6 : Contoh tampilan program ketika dijalankan (*Hamilton Path* yang ditampilkan tidak semua karena alasan *formatting* pada laporan)

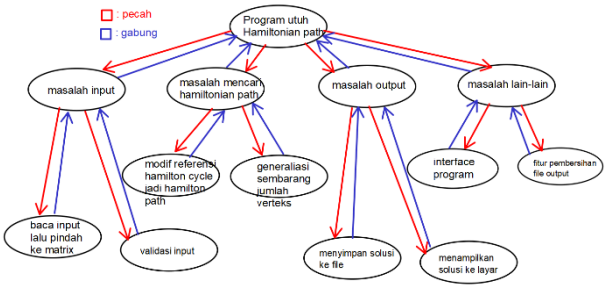
1.2 Analisis Permasalahan

Dalam menyelesaikan permasalahan ini, dilakukan metode *programming* secara modular, yaitu membagi program menjadi beberapa fungsi dan/atau prosedur untuk menyelesaikan permasalahan-permasalahan kecil dan kemudian digabung menjadi satu solusi untuk permasalahannya secara keseluruhan.



Gambar 1.7 : ilustrasi pendekatan programming secara modular

Jika disesuaikan dengan permasalahan ini, berikut adalah ilustrasi pemecahan-mecahan masalahnya:



Gambar 1.8 : pendekatan programming secara modular untuk program hamilton path

Pertama, terkait dengan input program adalah masalah untuk memvalidasi dan menyimpan data *graph* dalam bentuk *adjacency matrix* dari file eksternal. Langsung pada main program akan dibuat suatu tahapan yang memvalidasi kemudian membaca isi dari file eksternal apabila ada, kemudian menyimpannya ke dalam matriks berukuran V x V dengan V jumlah vertex/node pada graph.

Kedua, terkait dengan implementasi algoritma Hamiltonnya. Kami tidak melakukan pemrograman dari nol terkait dengan implementasi algoritma, tetapi mengembangkan dan memodifikasi referensi [2] serta [3] agar mampu memenuhi spesifikasi tugas besar yang diberikan. Pada referensi [2], disajikan kode program untuk mencari hamilton cycle dari suatu graph yang tersedia dalam bentuk adjacency matrix. Kami harus mampu mengubah kapabilitas kode untuk mampu mencari hamilton path dan

juga generalisasi untuk jumlah vertex sembarang. Untuk mampu mengurusi adjacency matrix dengan jumlah matrix sembarang, kami menggunakan matrix of integer yang di-alokasi secara dinamis menggunakan malloc.

Terakhir, terkait output program. Solusi-solusi *Hamilton Path* yang diperoleh akan ditampilkan ke layar dan juga disimpan ke file eksternal. Untuk itu akan dibuat suatu fungsi yang menuliskan semua jalur yang ditemukan ke dalam satu file eksternal baru.

Selain permasalahan input, proses, dan output, terdapat juga masalah lain seperti pemrosesan file output dan juga interface program.

2. ALGORITMA

2.1 Daftar Variabel dan Fungsi/Prosedur

Berikut adalah daftar variabel beserta deskripsinya yang terdapat dalam program kami:

Tabel 2.1 : Daftar variabel pada main.c beserta deskripsinya

Nama Variabel	Tipe Data	Deskripsi
aksi	character	Pilihan aksi yang ingin dilakukan user
isProgramFinished	boolean	Flag yang menyatakan apakah program sudah ingin terminasi atau belum
V	integer	Banyaknya vertex pada graph
filename	string	Nama file input yang berisi representasi adjacency matrix dari graph
temp	string (pointer to chracter)	String untuk mengambil token dari filename
temp2	string (array of character)	String untuk memodifikasi filename (setelah pemeriksaan apakah tidak kosong dan sesuai format), string filename akan dikembalikan
n_solusi	integer	Jumlah solusi dari operasi mencari hamiltonian path terakhir
stream	pointer to file	Pointer ke file dalam skema pemindahan input dari file ke matrix of integer
line	array of chracter (string)	Baris yang sedang dibaca pada file dalam skema pemindahan input dari file ke matrix of integer
token	pointer to character	Substring dari line (diperoleh dari penggunaan strtok())
graph	matrix [V ×V] of integer	(dialokasi secara dinamis menggunakan malloc)
hasil_sebelumnya	Array [0..V] of integer	Array yang menyimpan hamiltonian path yang terakhir kali diperoleh (dari iterasi sebelumnya)

n_solusi_hapus	String (akan dikonversi menjadi integer)	Jumlah file solusi_i.txt yang ingin dihapus. Apabila nilainya "a", maka semua file solusi akan dihapus.

Berikut adalah daftar fungsi/prosedur bentukan yang berkaitan dengan algoritma pencarian hamiltonian path beserta deskripsinya yang terdapat dalam file func-proc.c/func-proc.h:

Tabel 2.2 : Daftar Fungsi/Prosedur bentukan pada func-proc.c/func-proc.h beserta deskripsinya

Fungsi/Prosedur	Deskripsi
void hamiltonianPath(int** graph, int hasil_sebelumnya[], int V, int *nSolusi)	Sebagai prosedur utama mencari hamilton cycle dari suatu graph. I.S. graph yang ingin dicari hamilton pathnya telah terdefinisi. F.S. semua kemungkinan hamilton path dari graph telah diperoleh dan dicetak ke layar
void cetakSolusi(int solusi[],int n)	Mencetak hamilton path yang telah diperoleh dari posedur cariSolusi() ke layar. I.S. path hamilton telah ditemukan (sekuens vertexnya disimpan dalam bentuk array). F.S. path hamilton tercetak ke layar
void cariSolusi(int** graph, int visited[],int hasil[],int node,int start, int counter, int hasil_sebelumnya[], int *no_solusi, int V)	Prosedur rekursif yang mencari semua kemungkinan hamiltonian path dari suatu graf yang diberikan. I.S. graf yang ingin dicari hamiltonian pathnya telah terdefinisi sebagai sebuah matrix of integer. F.S. semua hamiltonian path akan dicari (serta ditampilkan ke layar dan disimpan ke file)
void setDefault(int visited[], int V)	Menandai semua vertex belum dikunjungi. I.S. array visited telah terdefinisi. F.S. semua elemen array visited di-inisiasi dengan nilai -1.
void simpanKeFile(int solusi[], int n, int noFile);	Menyimpan solusi hamiltonian path ke dalam file. I.S. hamiltonian path telah dicari oleh prosedur cariSolusi. F.S. hamiltonian path akan disimpan ke dalam file solusi_n.txt dengan n menyatakan path ke-n yang ditemukan oleh prosedur cariSolusi()
void hapusFile(int nFile);	Menghapus file solusi solusi_1.txt sampai solusi_nFile.txt. I.S. folder output terdefinisi. F.S. jika ada file solusi_n.txt untuk n dari 1 sampai n_File, maka akan dihapus, bila tidak, akan ditampilkan pesan error.
void validasi_file(char filename[]);	Memvalidasi nama dan format file eksternal. I.S. -. F.S. nama file ada dan format file benar (.txt).

Berikut adalah daftar fungsi/prosedur bentukan yang berkaitan dengan interface program yang terdapat dalam file interface.h :

Tabel 2.3 : Daftar Fungsi/Prosedur bentukan pada interface.h beserta deskripsinya

Fungsi/Prosedur	deskripsi
void cetakBumperOp()	Mencetak bumper pembuka berupa ASCII art 'Hamiltonian'
void cetakPengarang()	Mencetak nama pengarang program
void help()	menampilkan bantuan opsi aksi yang bisa dilakukan

void opening()	Memanggil prosedur cetakBumper() dan cetakPengarang() di awal program
void cetakBumperCls()	Mencetak bumper penutup berupa ASCII art 'Thank you!'
void closing()	Memanggil prosedur cetakBumperCls() di akhir program

2.2 Alur Program

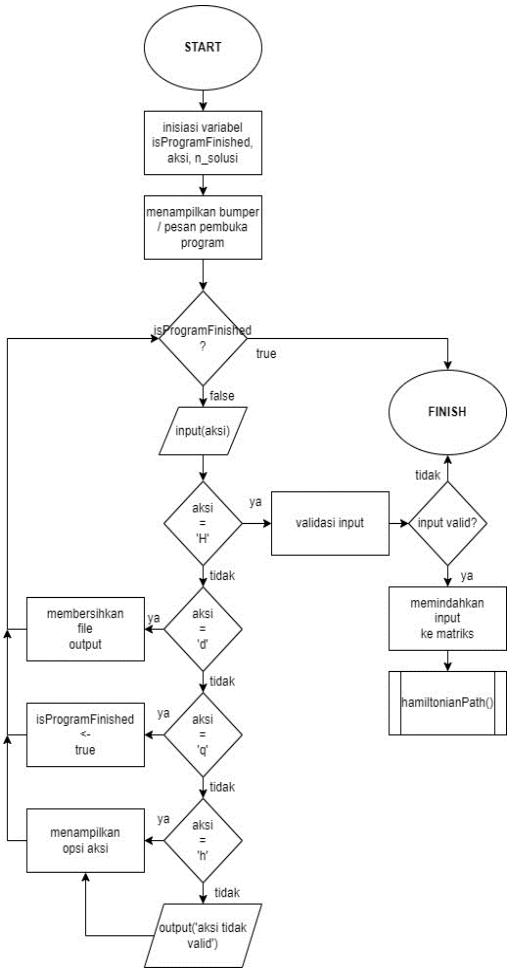
Agar lebih nyaman dalam menjelasakan alur program, kita akan bahas dari flowchart, bukan sourcecode.

Flowchart program penyelesaian sementara yang kami usulkan adalah sebagai berikut.

Main

Untuk main program, pertama akan diinisiasi variabel isProgramFinished sebagai false, n_solusi nol, serta aksi. Kemudian ser akan terus diminta melakukan aksi tertentu sampai variabel isProgramFinsihed bernilai true.

Apa yang terjadi pada suatu iterasi tergantung dari nilai variabel aksi. Bilai aksi bernilai 'h', maka akan ditampilkan pesan opsi yang dapat dilakukan dengan pemanggilan prosedur help(). Bila aksi bernilai 'q', maka program akan terminasi. Bila aksi bernilai 'd', maka akan dilakukan skema penghapusan file solusi. Bila aksi bernilai 'H', maka akan dilakukan validasi input, pemindahan input dari file ke matrix, dan penjalanan algoritma utama yaitu pencarian hamiltonian path dari graph.



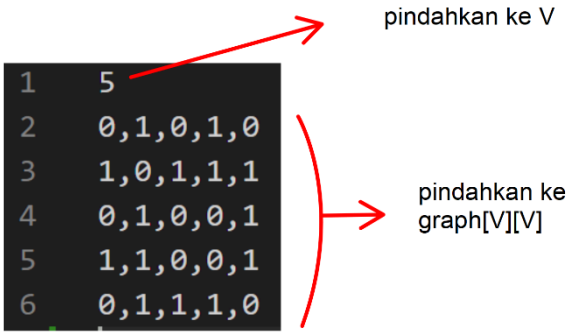
Gambar 2.1 : flowchart fungsi main (driver code)

Menyimpan File ke dalam Matriks

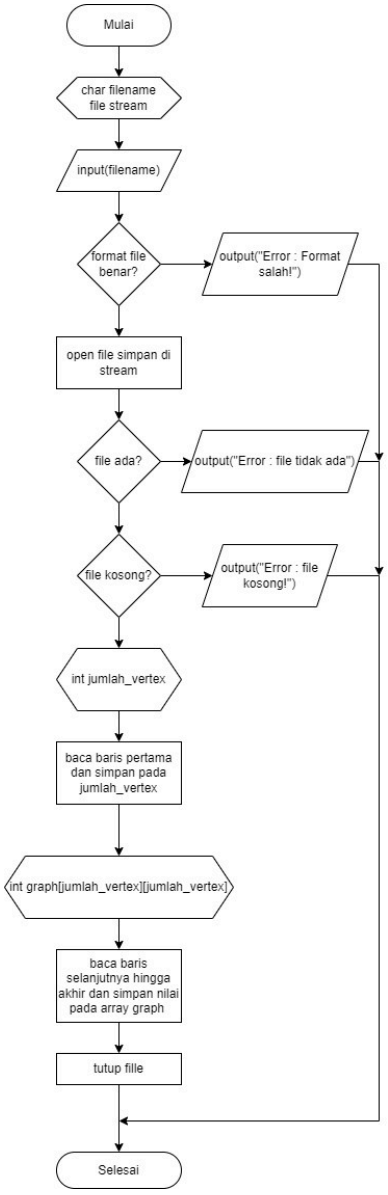
Pertama-tama kita validasi terlebih dahulu apakah file input ada, sudah sesuai format, dan tidak kosong. Setelah valid, kita lanjut ke skema pemindahan file.

Untuk skema pemindahan file ke dalam variabel program sebenarnya mirip-mirip untuk program apapun. Kita akan melakukan pembacaan isi file menggunkaan fread() secara baris per baris dan memindahkannya ke variabel string line, lalu kita akan memecah-mecah atau mengambil substring dari line menggunakan strtok() untuk dipindahkan ke variabel pointer to character bernama token. Kemudian konten token akan dipindahkan ke matrix dengan terlebih dahulu dikonversi menjadi integer menggunakan atoi().

Hal yang perlu diperhatikan adalah setelah pembacaan baris pertama berupa banyaknya vertex dan disimpan ke variabel V, kita langsung alokasi secara dinamis matrix[V×V] of integer bernama graph menggunkaan malloc().



Gambar 2.2 : Ilustrasi isi file input dipindahkan ke variabel program apa



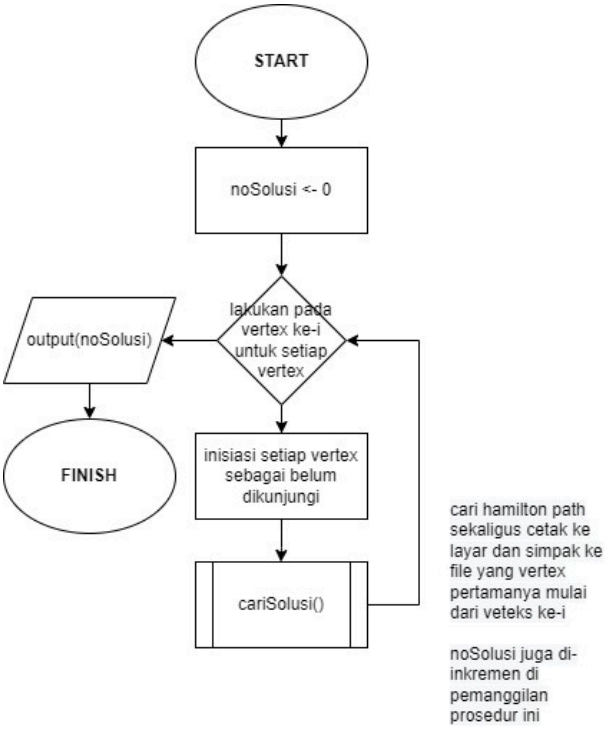
Gambar 2.3 : Flowchart bagian program saat memindahkan konten file ke dalam matriks

HAMILTONIAN PATH

Hamiltonian path adalah interface antara fungsi main() dengan algoritma utama pencarian hamiltonian path yaitu cariSolusi().

Pada prosedur hamiltonianPath(), jumlah solusi akan di-inisiasi dengan nol lalu akan dilakukan pemanggilan prosedur cariSolusi() secara berulang dengan vertex awal adalah vertex ke-i untuk setiap vertex yang ada di graph. Artinya akan dilakukan pencarian hamiltonian path mulai dari vertex ke-i. Setelah pencarian hamiltonian path selesai, maka banyaknya solusi noSolusi akan ditampilkan ke layar.

Sebagai catatan : pencetakan path dan penyimpanan ke file eksternal terjadi di cariSolusi() dan dilakukan sesaat setelah suatu path tertentu ditemukan.



Gambar 2.4 : flowchart prosedur hamiltonianPath() yang merupakan interface fungsi main dengan cariSolusi()

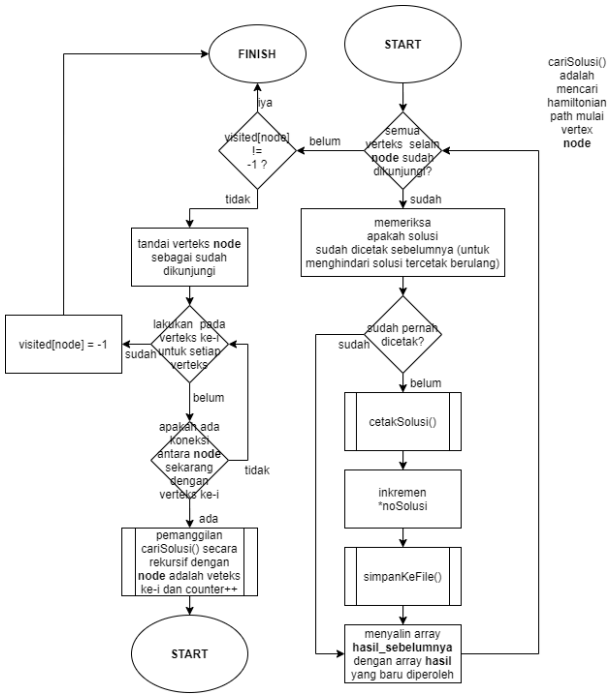
CARI SOLUSI

cariSolusi() merupakan *core* dari program kita karena merupakan implementasi dari algoritma pencarian hamiltonian path dari suatu graph. Prosedur ini juga merupakan prosedur yang paling sukar untuk dipahami karena merupakan prosedur rekursif.

Secara sederhana dijelaskan, selama semua vertex selain node belum dikunjungi (belum terbentuk hamilton path), maka akan dilakukan pemanggilan kembali prosedur cariSolusi() secara rekursif dengan parameter node baru adalah verteks ke-i yang sedang ditinjau. Selama

peninjauan verteks ke-i tersebut, array hasil juga sambil diisi. Sebelum verteks ke-i ditinjau, akan diperiksa dulu apakah terdapat koneksi antara node dengan verteks ke-i pada graf.

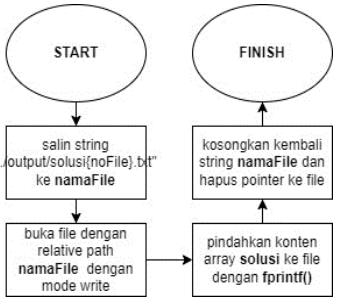
Bila array hasil telah penuh (semua veteks telah dikunjungi), maka akan dilakukan pemeriksaan apakah array hasil sama dengan array hasil_sebelumnya atau tidak. Bila tidak sama, maka akan dilakukan pencetakan solusi, inkremen jumlah solusi, penyimpanan solusi ke array, dan juga pembaruan isi variabel array hasil_sebelumnya.



Gambar 2.5 : Flowchart prosedur cariSolusi() yang merupakan core dari implementasi algoritma pencarian hamiltonian path

Simpan ke File

Penyimpanan ke file sebenarnya sama saja dengan pencetakan ke layar, tetapi alih-alih kita menggunakan printf(), kita menggunakan fprintf(). Sebelum itu, kita buat dulu string namaFile dan melakukan referencing pointer fp ke file solusi{noFile}.txt yang terletak di sebuah folder khusus bernama 'output'.



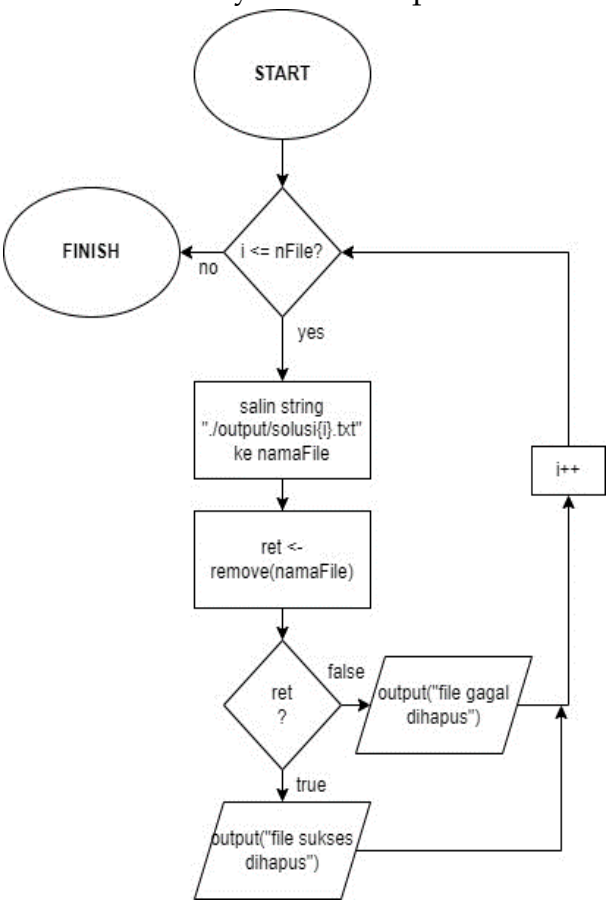
Gambar 2.6 : Flowchart prosedur simpanKeFile() untuk menyimpan hamiltonian path ke file eksternal

Hapus File

Untuk menghapus ke file, kita akan menggunakan

diisi. Sebelum verteks ke-i ditinjau, akan diperiksa dulu apakah terdapat koneksi antara node dengan verteks ke-i pada graf.

Bila array hasil telah penu

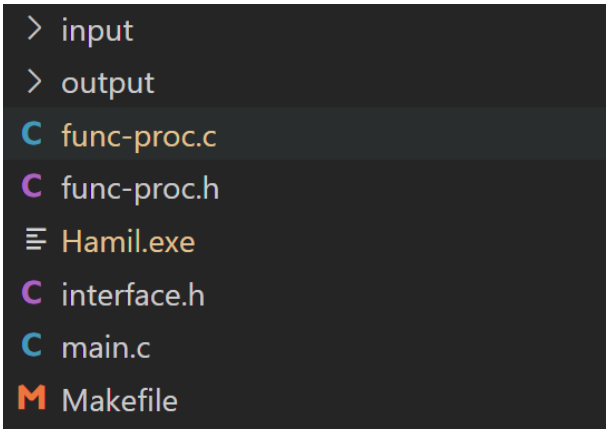


2.3 Dataflow Diagram (DFD)

2.4 Analisis Kompleksitas

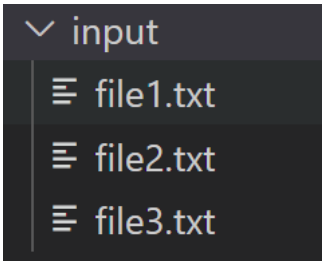
3. Struktur File

Berikut adalah struktur program yang kami buat untuk program pencarian hamiltonian path dari graf:



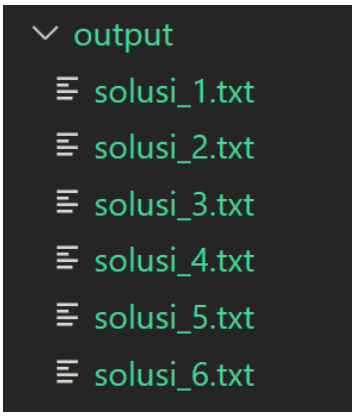
Gambar 3.1 : Struktur file program hamiltonian path

Pertama, terdapat dua folder bernama input dan output. Seperti bisa diduga, folder input adalah tempat kita meletakkan file eksternal .txt yang berisi representasi adjacency matrix dari graph.



Gambar 3.2 : Gambaran isi folder input

Sedangkan folder output berisi kumpulan file .txt dengan nama solusi_n.txt yang menyatakan path ke-*n* yang ditemukan dari algoritma pencarian hamiltonian path.



Gambar 3.3 : Gambaran isi folder output

Kami membuat ketetapan bahwa input dan output harus berada di dalam folder khusus agar struktur file kita lebih rapih. Bayangkan betapa *chaos*-nya struktur file kita bila semuanya (file input, file output, sourcecode) kita gabung di dalam satu folder saja.

Selanjutnya, terlihat bahwa kita pada struktur file kami, terdapat dua buah file sourcecode .c dan 2 buah header file .h.

Pertama, main.c adalah driver code dari program kita. Driver code artinya kode inilah yang mengintegrasikan semua file .c dan .h yang lain. Main.c juga merupakan kode yang menggambarkan periferal / bagaimana user berinteraksi dengan program kita.

```
32  int main()
33  {
34    bool isProgramFinished = false;
35    char aksi;
36    int n_solusi = 0;
37
38    // pesan selamat datang (ala-ala TuBes)
39    opening();
40
41    while (!isProgramFinished){
42      printf("Masukkan aksi:\n>>> ");
43      aksi = getchar();
44      if (aksi == 'H'){···
107
108     else if (aksi == 'd'){···
119
120     else if (aksi == 'q'){···
124
125     else if (aksi == 'h')
126       help();
127
128     else
129       printf("aksi tidak valid!\n");
130
131     fflush(stdin); /* penting untuk membersihkan buffer.
133   }
134   return 0;
135 }
```

Gambar 3.4 : cuplikan isi file main.c yang merupakan driver code

Kedua, func-proc.h adalah file header yang berisi deklarasi konstanta dan deklarasi prototipe fungsi/prosedur bentukan kami.

```
// DEKLARASI KONSTANTA

#define maxSize 63 // untuk ukuran string

// DEKLARASI FUNGSI

void hamiltonianPath(int** graph, int hasil_sebelumnya[], int V, int *nSolusi)
void cetakSolusi(int solusi[],int n);
void cariSolusi(int** graph, int visited[],int hasil[],int node,int start, int
void setDefault(int visited[], int V);
void simpanKeFile(int solusi[], int n, int noFile);
void hapusFile(int nFile);
void help();
void validasi_file(char filename[]);
```

Gambar 3.5 : cuplikan isi file header func-proc.h yang merupakan tempat deklarasi konstanta dan prototipe fungsi/prosedur

Sedangkan func-proc.c berisi realisasi dari masing-masing fungsi/prosedur yang telah kita deklarasikan prototipenya pada func-proc.h

```
#include "func-proc.h"

> void cetakSolusi(int solusi[],int n)···

> void cariSolusi(int** graph, int visited[],int hasil[],int node,int start, int count

> void setDefault(int visited[], int V)···

> void hamiltonianPath(int** graph, int hasil_sebelumnya[], int V, int *nSolusi)···

> void simpanKeFile(int solusi[], int n, int noFile)···

> void hapusFile(int nFile)···

> void validasi_file(char filename[]){···
```

Gambar 3.6 : cuplikan isi file func-proc.c yang merupakan tempat realisasi fungsi/prosedur bentukan

Alasan kami merealisasikan fungsi/prosedur bentukan dalam file terpisah adalah modularitas. Kami tidak ingin sourcecode main.c (driver code) terlalu ramai sehingga alur program sulit dipahami.

Selanjutnya terdapat interface.h yang merupakan tempat kami membuat komponen-komponen periferal/ interaksi dengan user seperti bumper, nama pengarang, menampilkan opsi bantuan, opening dan closing.

```
char bumper_10[127] = "      \\ \\  \\ \\ \\    __  \\ \\    __  \\ \\
char bumper_11[127] = "       \\ \\  \\ \\ \\  \\ \\  \\ \\  \\ \\  \\ \\
char bumper_12[127] = "        \\ \\__\\ \\ \\ \\__\\ \\ \\__\\ \\ \\__\\ \\__
char bumper_13[127] = "         \\\\|__|  \\\\|__|\\\\|__|\\\\|__|\\\\|__|
char bumper_14[127] = "
char bumper_15[127] = "

// DEKLARASI FUNGSI/PROSEDUR
> void cetakBumperOp()···

> void cetakPengarang()···

> void help()···

> void opening()···

> void cetakBumperCls(){···

> void closing(){···
```

Gambar 3.7 : cuplikan isi file interface.h yang merupakan tempat realisasi komponen periferal program

Terakhir, hasil kompilasi semua sourcecode adalah sebuah file exectuable Hamil.exe.

4. Pengujian Program

5. Kesulitan Yang Dihadapi

Dalam merealisasikan program *Hamilton Path* ini, kami menghadapi beberapa kesulitan, antara lain sebagai berikut.

Program tidak dapat menerima input dari file eksternal yang berada dalam suatu folder.

Awalnya tidak dapat menerima input jumlah vertex sembarang (harus di-*define*).

Program belum sempurna dalam menghapus file output jalur-jalur Hamilton yang ditemukan. Karena menggunakan sistem *overwrite*, apabila file sebelumnya memiliki hasil yang lebih banyak dari file terbaru, sebagian solusi dari file lama tetap ada, tidak terhapus.

Program tidak dapat memvalidasi apakah isi file sesuai dengan format, yaitu jumlah *vertex* pada baris pertama, dilanjut dengan *adjacent matrix* yang dipisah oleh koma.

Berkaitan dengan kesulitan di atas, hampir semua ditemukan solusinya. Solusi yang kami peroleh adalah sebagai berikut.

Mengubah string nama file menjadi "./<nama_folder>/<nama_file>.

Menggunakan malloc() untuk menginisiasi matriks

6. Kontribusi Anggota

Fungsi/Prosedur	Berkas/*library* terkait	*Programmer*	*Tester*
Memvalidasi file eksternal	main.h func-proc.c func-proc.h	Gilbert Ng (13220032)	Semua anggota
Membaca dan menyimpan file eksternal	main.h func-proc.c func-proc.h	Jota Baret Tata (13220048)	Semua anggota
Menulis *Hamilton Path* ke dalam file eksternal	main.h func-proc.c func-proc.h	Yusuf Hafizh Sofwan (13220040)	Semua anggota
Mencari *Hamilton Path* dari *graph*	main.h func-proc.c func-proc.h	Bostang Palaguna (13220055)	Semua anggota
Menghapus file eksternal	main.h func-proc.c func-proc.h	Michael Tiroi (13220062)	Semua anggota
Interface program	main.h interface.h	Bostang Palaguna (13220055)	Semua anggota

7. Kesimpulan dan Saran

Daftar Pustaka

[1] Kenneth H. Rosen, *Discrete Mathematics and its application*, McGrawHill, USA, 2007.

[2] Dismas Widyanto, *Naskah Soal Besar Tipe 2*, Lab Dasar Teknik Elektro, Bandung, 2022.

[3] https://kalkicode.com/print-hamiltonian-path-present-graph?msclkid=c0c150b3bf1411ec9c0983650e97158d, diakses 18 April 2022, 20.29.

[4] Dynamic allocation of an unknown matrix in C - Codegrepr, diakses 19 April 2022

[5] Hamiltonian Path -- from Wolfram MathWorld, diakses 22 April 2022