Penerapan Algoritma Runut-balik untuk Menemukan Sirkuit Hamilton pada Graf

Isa Mujahid Darussalam 13517002
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13517002@std.stei.itb.ac.id

***Abstrak*—Dalam teori graf, sirkuit Hamilton adalah suatu lintasan yang melalui setiap simpul dalam graf tepat satu kali dan kembali ke tempat awal. Umumnya, semakin kompleks dan rumit bentuk suatu graf, maka semakin rumit pula menemukan sirkuit Hamiltonnya secara coba-coba. Oleh karena itu, diperlukan suatu algoritma untuk menemukan sirkuit Hamilton pada suatu graf. Pada makalah ini, digunakan algoritma runut-balik untuk menemukan sirkuit Hamilton dari suatu graf. Proses runut-balik sendiri menggunakan konsep fungsi rekursif, yaitu suatu fungsi yang didefinisikan mengacu kepada dirinya sendiri.**

***Kata kunci*—runut-balik, rekursif, sirkuit, Hamilton, graf.**

I. PENDAHULUAN

Salah satu ilmu dari Matematika Diskrit adalah teori tentang graf. Aplikasi dari graf banyak digunakan di berbagai bidang, seperti kimia, kedokteran, sosiologi, ekonomi, ilmu komputer, dan masih banyak lainnya. Salah satu yang menarik dari graf adalah sirkuit Hamilton.

Sirkuit Hamilton diterapkan dalam beberapa permasalahan, contohnya *travelling salesperson problem* dan *knight tour*. *Travelling salesperson problem* (TSP) adalah suatu permasalahan di mana terdapat sejumlah kota dengan jarak yang diketahui satu dengan yang lain. Tiap kota harus disinggahi satu kali kemudian kembali ke kota awal dengan rute terpendek. Untuk menyelesaikan permasalahan ini, kita perlu menemukan sirkuit Hamilton berbobot minimum. Sedangkan, *knight tour* adalah permasalahan untuk menemukan sirkuit Hamilton dari bidak kuda pada papan catur berukuran N × N. Kuda digerakkan dan harus menempati setiap kotak pada catur satu kali dan kembali ke posisi awal.

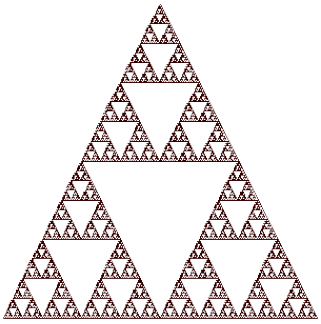
Untuk menemukan sirkuit Hamilton pada suatu graf, terdapat berbagai macam cara. Cara paling naif adalah dengan menelusuri satu persatu kemungkinan yang ada (*brute force*). Pertama, pilihlah simpul awal. Kemudian telusuri setiap lintasan yang ada dengan melewati setiap simpul tepat satu kali hingga kembali lagi ke simpul awal. Namun, cara seperti ini tentu tidak efisien. Pada makalah ini, diterapkan algoritma runut-balik (*backtracking*) untuk menemukan sirkuit Hamilton pada graf.

Algoritma runut-balik memanfaatkan konsep dari fungsi rekursif. Fungsi rekursif adalah suatu fungsi yang mengacu kepada dirinya sendiri. Fungsi rekursif banyak diimplementasikan dalam berbagai permasalahan yang menuntut runut-balik. Selain itu, fungsi rekursif dapat menggantikan iterasi dalam pemrograman prosedural meskipun tidak semua kasus cocok menggunakan rekursi. Dalam konsep rekursif, dikenal basis dan rekurens yang akan dibahas lebih lanjut pada poin selanjutnya

II. LANDASAN TEORI

A. Rekursif

Rekursif (*recursive)* adalah suatu fungsi yang didefinisikan mengacu kepada dirinya sendiri. Proses rekursif disebut sebagai rekursi (*recursion).* Berikut adalah contoh-contoh objek rekursif :



Gambar 2.1 : Segitiga dalam segitiga
(Sumber : http://www.cs.cmu.edu/~ref/pgss/lecture/7/sier.gif, diakses 8 Desember 2018 pukul 16:36 GMT+7).

Gambar 2.2 : Proses pembuatan pohon dengan rekursif
(Sumber : https://introcs.cs.princeton.edu/java/23recursion/images/recursive-trees.png, diakses 8 Desember 2018 pukul 16:40 GMT+7).

Di dalam fungsi rekursif terdapat dua komponen antara lain :

1. Basis
   Basis adalah bagian dari fungsi rekursif yang didefinisikan secara eksplisit (tidak mengacu kepada dirinya sendiri). Bagian ini berfungsi untuk menghentikan fungsi rekursif dan memberikan nilai yang terdefinisi.
2. Rekurens
   Rekurens adalah bagian yang mengacu kepada terminologi fungsinya sendiri. Setiap rekurens harus dapat mencapai basis yang dituju.

Salah satu permasalahan yang dapat ditinjau secara rekursif adalah faktorial. Berikut adalah definisi faktorial secara rekursif :

a. Basis: $n! = 1$, untuk $n = 0$.
b. Rekurens: $n! = (n-1)! \times n$, untuk $n > 0$.

Perhatikan kasus di atas. Nilai dari basis terdefinisi secara eksplisit saat $n = 0$. Kemudian untuk nilai yang lebih besar dari nol, rekurens mengacu kepada terminologi fungsi faktorial itu sendiri. Sekarang, akan didemonstrasikan langkah menghitung secara rekursif. Dari definisi di atas, maka 4! dapat dihitung dengan langkah sebagai berikut :

[1] $4! = 3! \times 4$
[2] $4! = 2! \times 3 \times 4$
[3] $4! = 1! \times 2 \times 3 \times 4$
[4] $4! = 0! \times 1 \times 2 \times 3 \times 4$
[5] $4! = 1 \times 1 \times 2 \times 3 \times 4$
[6] $4! = 24$

Didapat nilai dari 4! = 24.

Pemahaman tentang rekursif sangat diperlukan dalam bidang matematika dan ilmu komputer. Beberapa masalah dapat dipecahkan lebih mudah dengan rekursif. Contohnya untuk membentuk *abstract data type* (ADT) graf dan pohon. Proses runut-balik diperlukan dan proses ini lebih mudah diimplementasikan secara rekursif.

B. Algoritma runut-balik

Algoritma runut-balik adalah perbaikan atau versi lebih baik daripada algoritma *brute-force*. Algoritma ini digunakan untuk menemukan solusi dari suatu permasalahan dengan mengeksplorasi tiap-tiap kandidat yang mengarah ke solusi dan mengabaikan kandidat (*backtracking*) yang ternyata tidak mungkin memenuhi kondisi dari solusi.[2][3]

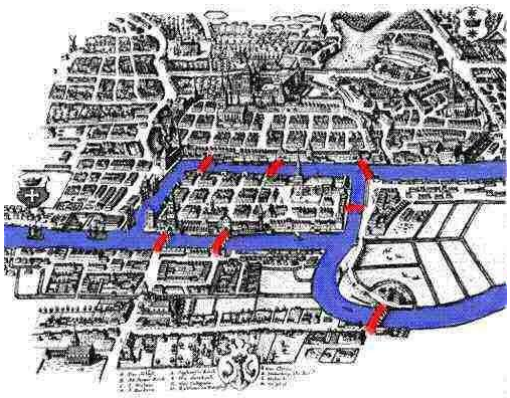
Solusi dari algoritma runut-balik dapat dipandang sebagai vektor (v_1, ..., v_m). Kondisi awal vektor kosong. Di setiap langkah, vektor diisi dengan nilai baru yang memenuhi solusi parsial. Apabila vektor parsial (v_1, ..., v_i) tidak dapat memenuhi solusi parsial, akan dilakukan runut-balik/mundur dengan menghapus nilai vektor terakhir dan mencoba nilai lain yang memenuhi solusi.[2]

Algoritma runut-balik dapat diterapkan untuk menyelesaikan permasalahan seperti *N-Queens*, *knight tour*, sudoku, *scrabble*, pewarnaan graf, parsing, dan banyak lainnya.

C. Graf

Graf adalah pasangan himpunan $G = (V,E)$ dengan V adalah himpunan tak kosong dari simpul, dan E adalah himpunan dari sisi yang menghubungkan sepasang simpul. Singkatnya, graf merupakan himpunan dari objek-objek (simpul) tidak kosong yang dihubungkan oleh garis (sisi).[1]

Penggunaan graf pertamakali menurut sejarah adalah jembatan Königsberg. Di kota Königsberg, Jerman (sekarang menjadi kota Kaliningrad, Russia), terdapat sungai Pregal yang mengalir mengitari pulau Kneiphof lalu bercabang menjadi dua buah anak sungai. Ada tujuh buah jembatan yang menghubungkan daratan yang dibelah oleh sungai tersebut.



Gambar 2.3 : Jembatan Königsberg
(Sumber : https://www.maa.org/sites/default/files/images/cms_upload/Konigsberg_colour37936.jpg, diakses 8 Desember 2018 pukul 20:05 GMT+7).

Masalah dari jembatan Königsberg adalah, dapatkah kita melalui setiap jembatan tepat satu kali dan kembali ke tempat semula? Sebagian orang setuju bahwa jawabannya adalah tidak mungkin, tetapi mereka tidak mampu membuktikan jawabannya. Tahun 1736, barulah Leonhard Euler menjadi orang pertama yang mampu membuktikan masalah jembatan Königsberg. Euler memodelkan jembatan Königsberg dalam graf dengan tiap daratan sebagai simpul dan jembatan sebagai sisi graf. Dalam graf tersebut semua simpul berderajat ganjil sehingga tidak mungkin melalui tiap sisi satu kali dan kembali ke tempat semula. Diperlukan semua simpul dengan derajat genap supaya dapat melalui tiap sisi tepat satu kali dan kembali ke simpul awal.[1]

Graf dapat dibagi menjadi beberapa jenis. Berdasarkan ada tidaknya gelang atau sisi ganda pada graf, maka graf digolongkan menjadi dua jenis:

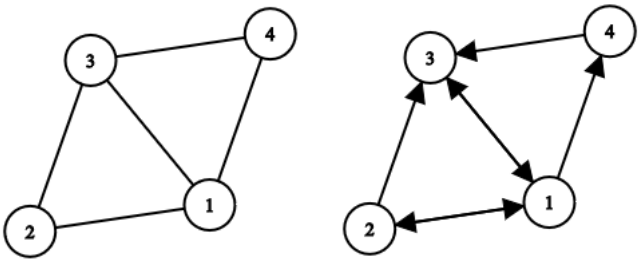
1. Graf sederhana
   Graf sederhana adalah graf yang tidak mengandung gelang maupun sisi ganda.

2. Graf tak-sederhana
   Graf tak-sederhana adalah graf yang mengandung sisi ganda atau gelang. Terdapat dua macam graf tak-sederhana yaitu graf ganda (multigraph) dan graf semu (*pseudograph*). Graf ganda adalah graf yang memiliki sisi ganda, sedangkan graf semu adalah graf yang mengandung gelang (*loop*).

Berdasarkan orientasi arah pada sisi, graf dibagi menjadi dua jenis:

1. Graf tak-berarah
   Graf tak-berarah adalah graf yang sisinya tidak mempunyai orientasi arah.

2. Graf berarah
   Graf berarah adalah graf yang tiap sisinya memiliki orientasi arah. Sisi berarah disebut sebagai busur. Pada graf berarah (u,v), simpul u disebut sebagai simpul asal (*initial vertex*) dan simpul v disebut sebagai simpul tujuan/terminal (*terminal vertex*).



Gambar 2.4 : Graf tak-berarah (kiri) dan graf berarah (kanan).

Graf memiliki beberapa terminologi dasar. Terminologi dari graf antara lain :

1. Ketetanggaan (*Adjacent*)
   Dua buah graf disebut tetangga apabila dua simpul terhubung secara langsung oleh sisi. Contohnya pada Gambar 2.4, pada graf tak-berarah di sisi kiri, simpul 1 bertetangga dengan simpul 2, 3, dan 4.
2. Bersisian (*Incidency*)
   Ambil sembarang sisi pada graf, misalkan sisi $u = (v_i , v_j)$. Maka sisi u dikatakan bersisian dengan simpul v_i dan v_j.
3. Simpul terpencil (*isolated vertex*)
   Simpul terpencil adalah simpul dalam graf yang tidak mempunyai sisi yang bersisian dengannya.
4. Graf kosong (*null graph*)
   Graf kosong ialah graf yang tidak memiliki sisi atau himpunan sisinya kosong.
5. Derajat (*Degree*)
   Derajat dalam simpul ialah jumlah sisi yang bersisian dengan simpul tersebut. Contohnya pada Gambar 2.4, graf tak-berarah di sisi kiri dengan simpul 1 berderajat 3, dan graf berarah di sisi kanan dengan simpul 1 memiliki derajat masuk 2 serta derajat keluar 3.
6. Lintasan (*path*)
   Lintasan adalah barisan berselang-seling simpul dan sisi yang berbentuk v_0, e_1, v_1, e_2, ... , v_{n-1}, e_n, v_n dari simpul awal v_0 menuju simpul akhir v_n sehingga $e_1=(v_0,v_1)$, $e_2=(v_1,v_2)$, ... , $e_n = (v_{n-1}, v_n)$ adalah sisi-sisi dari graf.
7. Sirkuit (*cycle*)
   Sirkuit adalah lintasan yang berawal dan berakhir pada simpul yang sama.
8. Terhubung (*connected*)
   Pada graf tak berarah, graf disebut terhubung apabila terdapat lintasan yang menghubungkan setiap pasangan simpul v_i dan v_j yang terdapat dalam graf tersebut. Apabila tidak, maka graf tersebut disebut graf tak-terhubung (*disconnected graph*).

   Pada graf berarah, graf disebut terhubung apabila graf tidak berarahnya terhubung. Graf tidak berarah didapat dengan menghilangkan arah dari tiap busur pada graf berarah.
9. Upagraf (*subgraf*) dan komplemen upagraf
   Misalkan terdapat graf $G = (V, E)$ dan graf $G_1 = (V_1, E_1)$. Graf G_1 merupakan upagraf dari G jika $V_1 \subseteq V$ dan $E_1 \subseteq E$. Sedangkan, komplemen dari graf G_1 terhadap graf G adalah graf $G_2 = (V_2, E_2)$ dengan $E_2 = E - E_1$ dan V_2 adalah himpunan simpul yang bersisian dengan anggota E_2.
10. Upagraf rentang (*spanning subgraf*)
    Apabila $G_1 = (V_1, E_1)$ adalah upagraf dari $G = (V, E)$ dan $V_1 = V$ (G_1 mengandung semua simpul G), maka G_1 dikatakan upagraf rentang dari G.
11. *Cut-Set*
    Cut-set dari graf terhubung adalah himpunan sisi yang apabila dibuang menyebabkan graf tersebut menjadi tidak terhubung. Jadi, *cut-set* selalu menghasilkan dua buah komponen terhubung.
12. Graf berbobot (*Weighted graph*)
    Graf berbobot adalah graf yang setiap sisinya diberi harga/bobot.

Graf dapat direpresentasikan dalam berbagai macam bentuk. Di sini hanya diberikan representasi graf dalam bentuk matriks ketetanggaan. Suatu graf $G = (V,E)$ dapat direpresentasikan dalam matriks berukuran $n \times n$. Bila representasi graf tersebut dinamakan sebagai matriks $A = [a_{ij}]$, maka a_{ij} bernilai nol apabila simpul i dan j bertetangga atau a_{ij} bernilai satu apabila simpul i dan j tidak bertetangga.

Sebagai contoh digunakan Gambar 2.4 graf tak-berarah pada bagian kiri. Graf tersebut akan direpresentasikan dalam bentuk matriks ketetanggaan. Dengan meninjau tiap simpul pada graf dan tetangganya, didapat representasi graf dalam matriks A.

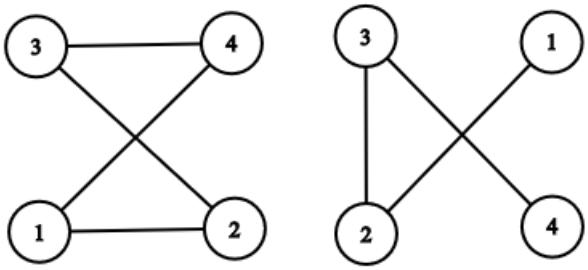
$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

Matriks A di atas adalah representasi graf dalam bentuk matriks ketetanggaan. Baris dan kolom disimbolkan sebagai tiap simpul yang terdapat pada graf.

Selanjutnya, akan dijelaskan apa itu lintasan dan sirkuit Hamilton. Konsep ini penting untuk dipahami karena makalah ini mengulas tentang algoritma menemukan sirkuit Hamilton.

Lintasan Hamilton adalah lintasan yang melalui tiap simpul di dalam graf tepat satu kali. Apabila lintasan itu kembali ke simpul asal dan membentuk lintasan tertutup (*cycle*), maka lintasan tertutup itu disebut sirkuit Hamilton. Jadi, sirkuit Hamilton adalah sirkuit yang melalui setiap simpul tepat sekali, kecuali simpul awal sekaligus akhir yang dilalui dua kali.

Graf yang memiliki sirkuit Hamilton dapat disebut sebagai graf Hamilton, sedangkan graf yang hanya memiliki lintasan Hamilton disebut sebagai graf semi-Hamilton.



Gambar 2.5 : Graf Hamilton (kiri) dan graf semi-Hamilton (kanan).

Ambil contoh pada Gambar 2.5. Graf di bagian kiri adalah graf Hamilton karena memiliki sirkuit Hamilton. Sirkuit Hamilton dari graf bagian kiri adalah 1, 2, 3, 4, 1. Sedangkan graf di bagian kanan adalah graf semi-Hamilton, karena memiliki lintasan Hamilton, namun tidak memiliki sirkuit Hamilton. Lintasan Hamilton dari graf bagian kanan adalah 1, 2, 3, 4.

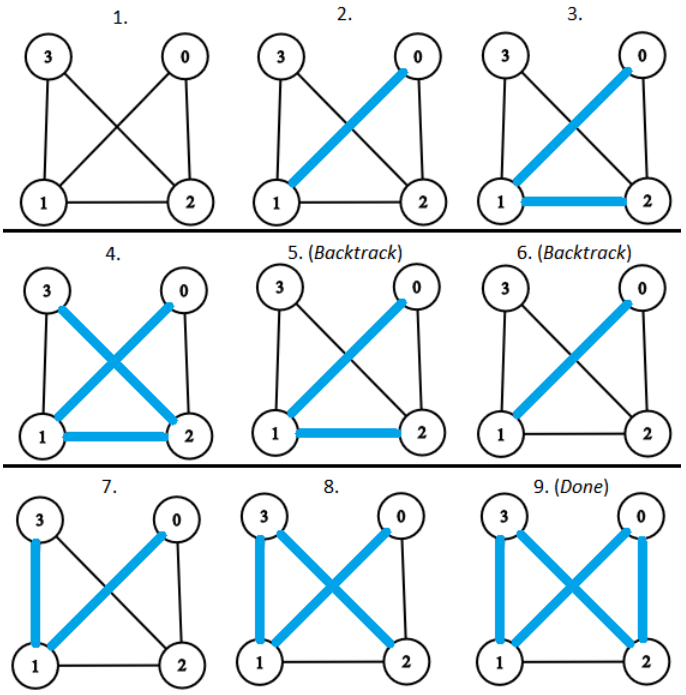
III. Algoritma Runut-balik untuk Menemukan Sirkuit Hamilton

Untuk menemukan sirkuit Hamilton, sebenarnya bisa saja dengan mencoba seluruh kemungkinan yang ada (*brute force*). Namun, cara ini tentunya tidak mangkus dan memiliki kompleksitas algoritma yang sangat besar (O(*n*!)). Untuk itu, diperlukan algoritma lain yang lebih baik. Pada kasus ini, algoritma runut-balik dapat diterapkan. Meskipun algoritma ini memiliki dimensi waktu (*Big O*) yang sama dengan algoritma *brute force* yaitu O(*n*!), namun nilai dari *n* pada algoritma runut-balik ini jauh lebih kecil sehingga mampu mempercepat waktu eksekusi.

Berikut langkah-langkah dari algoritma runut-balik untuk menemukan sirkuit Hamilton :

1. Representasikan graf dalam bentuk matriks ketetanggaan.
2. Mulai dari simpul ke-0. Masukkan simpul ke-0 sebagai solusi sementara.
3. Apabila semua simpul telah diperiksa dan simpul yang terakhir dengan simpul awal bertetangga maka solusi sirkuit Hamilton ditemukan, apabila tidak maka tidak ada sirkuit Hamilton (basis dari kasus).
4. Lakukan pengulangan sebanyak tiap simpul yang ada dalam graf. Pada setiap kalangnya :
   a. Periksa apakah simpul yang sedang diperiksa valid sebagai solusi sementara. Simpul valid apabila simpul tersebut bertetangga dengan simpul sebelumnya dan belum dimasukkan ke dalam solusi sementara. Apabila simpul valid, tambahkan simpul tersebut ke dalam solusi sementara.
   b. Jika simpul valid telah dimasukkan ke dalam solusi sementara, periksa juga apakah simpul tersebut valid untuk simpul-simpul setelahnya dengan memanggil fungsi itu sendiri (rekurens). Apabila tetap valid, kembalikan *true*, jika tidak lakukan runut-balik (*backtrack*) dengan menghapus solusi sementara tersebut dan mencoba kemungkinan simpul-simpul valid yang lain dengan kembali melakukan perulangan tiap simpul.
   c. Apabila setiap simpul telah diperiksa dan telah dimasukkan ke dalam solusi sementara artinya basis telah tercapai. Solusi sementara menjadi solusi dari sirkuit Hamilton graf.
   d. Apabila dari setiap simpul yang telah diperiksa dalam solusi sementara tidak lagi mempunyai simpul yang valid sebagai lintasan selanjutnya pada sirkuit Hamilton, maka solusi sementara dibatalkan. Artinya graf tidak mempunyai sirkuit Hamilton.

Berikut adalah ilustrasi langkah demi langkah untuk menemukan sirkuit Hamilton.



Gambar 3.1 : Ilustrasi langkah demi langkah algoritma runut-balik dalam menemukan sirkuit Hamilton dari graf.

Dari Gambar 3.1, terlihat jelas langkah demi langkah mencari sirkuit Hamilton dengan algoritma runut-balik. Sisi yang berwarna biru adalah sirkuit Hamiltonnya. Perhatikan langkah ke 4 dan 5, tidak ada simpul selanjutnya yang valid dari posisi simpul yang sekarang. Oleh karena itu, dilakukan proses runut-balik pada langkah berikutnya.

Di bawah ini adalah implementasi dari algoritma runut-balik dengan bahasa C. Perlu diperhatikan, graf pada program ini telah direpresentasikan dalam bentuk matriks ketetanggaan dan di-*hardcode* langsung di dalam program utama.

```
#include <stdio.h>
#define boolean unsigned char

/*------Variabel, fungsi, dan prosedur global-------*/
int N;

boolean isValid(boolean graf[N][N], int *solusi, int
CurPos, int ChkPos);

boolean CekSirkuitHamilton(boolean graf[N][N], int
*solusi, int CurPos);

void SolveandPrintHamiltonCycle(boolean graf[N][N]);

/*---------------------Driver-----------------------*/
int main(){
    boolean graf[4][4] = {{0, 1, 1, 0},
                          {1, 0, 1, 1},
                          {1, 1, 0, 1},
                          {0, 1, 1, 0},};
    N = 4;
    // Print the solution
    SolveandPrintHamiltonCycle(graf);
    return 0;
}
```

```
/*---------IMPLEMENTASI FUNGSI DAN PROSEDUR---------*/
boolean isValid(boolean graf[N][N], int *solusi, int
CurPos, int ChkPos)
/* IsValid adalah sebuah fungsi untuk mengecek apakah
simpul yang akan dicek adalah valid untuk membangun
sirkuit Hamilton sejauh ini */
{
    // Cek apakah bertetangga dengan posisi sebelumnya
    if (graf [solusi[CurPos-1]][ChkPos] == 0) {
        return 0;
    }
    else {
        boolean aman = 1;
        int i = 0;
        // Cek apakah simpul telah ditambah di solusi
        while (aman && i<CurPos) {
            if (solusi[i] == ChkPos)
                aman = 0;
            else
                i++;
        }
        return aman;
    }
}

boolean CekSirkuitHamilton(boolean graf[N][N], int
*solusi, int CurPos)
/* CekSirkuitHamilton merupakan fungsi rekursif untuk
mengecek tiap simpul dalam graf apakah dapat dibentuk
menjadi sirkuit Hamilton */
{
    /* Basis : semua simpul telah dicek */
    if (CurPos == N) {
        if (graf[solusi[CurPos-1]] [solusi[0]] == 0)
            return 0;
        else
            return 1;
    }
    else {
        boolean isSafe = 0;
        int i = 1;
        /* kalang untuk mencari jalur yang valid */
        while (!isSafe && i<N){
            if (isValid(graf, solusi,CurPos,i)) {
                solusi[CurPos] = i;
                /* Rekurens: Cek posisi selanjutnya
                   apakah valid */
                if (CekSirkuitHamilton(graf,solusi,
                CurPos+1))
                    isSafe = 1;
                else {
                    solusi[CurPos] = -1;
                    i++;
                }
            }
            else {
                i++;
            }
        }
        return isSafe;
    }
}
```

```
void SolveandPrintHamiltonCycle(boolean graf[N][N]){
/* SolveandPrintHamilton adalah prosedur untuk mencari
sirkuit Hamilton dengan memanggil fungsi
CekSirkuitHamilton */
    int solusi[N+1];
    int i;

    /* Inisialisasi. Solusi pertama adalah simpul 0 */
    /* Sisanya diisi dengan -1 */
    solusi[0] = 0;
    for (i = 1; i<N; i++){
        solusi[i] = -1;
    }
    /* Masukkan juga simpul pertama agar membentuk
    sirkuit Hamilton */
    solusi[N] = solusi[0];
    if (CekSirkuitHamilton(graf,solusi,1)) {
        printf("Sirkuit Hamiltonnya :");
        for (i = 0; i<=N; i++){
            printf(" %d",solusi[i]);
        }
        printf("\n");
    }
    else {
        printf("Sirkuit Hamilton tidak ditemukan\n");
    }
}
```

Driver atau *main* program dari kode sumber di atas telah melakukan *hardcode* dari graf. Dalam kasus di atas, graf yang di-*hardcode* adalah graf pada Gambar 3.1.

IV. Eksperimen

Pada bagian ini, akan dilakukan pengujian program untuk tiga buah graf. Karena representasi matriks dari graf di-*hardcode* dalam program utama, perlu diperoleh representasi matriks ketetanggan dari graf terlebih dahulu.

1. Graf yang pertama adalah graf pada Gambar 3.1. Bentuk Matriks ketetanggaan dari graf tersebut adalah:

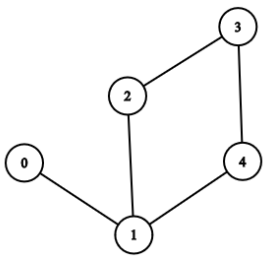
$$A_1 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

Ketika program dikompilasi dan dijalankan, program menghasilkan keluaran:

```
PS C:\Users\acer\Desktop\daftar_logo\Matdis> ./hamilton
Sirkuit Hamiltonnya : 0 1 3 2 0
```

Gambar 4.1 : Pengujian pertama sirkuit Hamilton dari graf dengan matriks A_1.

2. Graf yang kedua memiliki bentuk sebagai berikut:



Gambar 4.2 : Graf untuk pengujian kedua.

Dari graf pada Gambar 4.2, didapat matriks ketetanggaannya sebagai berikut:

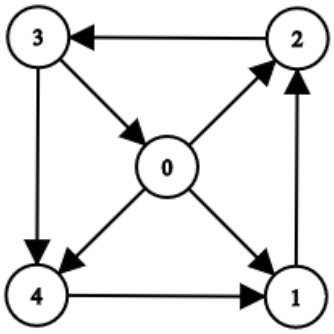
$$A_2 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

Dengan melakukan *hardcode* matriks A_2 di atas, didapat keluaran dari program:

```
PS C:\Users\acer\Desktop\daftar_logo\Matdis> ./hamilton
Sirkuit Hamilton tidak ditemukan
```

Gambar 4.3 : Hasil pengujian graf dengan representasi matriks A_2.

3. Pada pengujian yang ketiga, digunakan graf berarah sebagai berikut :



Gambar 4.4 : Graf untuk pengujian ketiga.

Matriks ketetanggan dari graf tersebut adalah :

$$A_3 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Ketika program dijalankan, keluaran yang ditampilkan adalah :

```
PS C:\Users\acer\Desktop\daftar_logo\Matdis> ./hamilton
Sirkuit Hamiltonnya : 0 4 1 2 3 0
```

Gambar 4.5 : Hasil pengujian untuk graf dengan matriks A_3.

V. KESIMPULAN

Ada banyak algoritma untuk menemukan sirkuit Hamilton pada suatu graf. Salah satu algoritma tersebut adalah algoritma runut-balik. Dalam makalah ini diterapkan algoritma runut-balik untuk mencari sirkuit Hamilton dari suatu graf yang telah direpresentasikan dalam bentuk matriks ketetanggaan. Perlu diingat bahwa dalam suatu graf mungkin saja memiliki lebih dari satu sirkuit Hamilton. Solusi yang didapat dari algoritma runut-balik hanya salah satu dari semua kemungkinan solusi.

VI. PENUTUP

Pertama, penulis mengucap puji syukur kepada Tuhan Yang Maha Esa atas limpahan rahmat dan karunia yang diberikan sehingga penulis dapat menyelesaikan makalah ini. Selanjutnya, penulis ingin mengucapkan terima kasih kepada para dosen mata kuliah Matematika Diskrit, bapak Rinaldi Munir, bapak Judhi Santoso, dan khususnya untuk ibu Harlili selaku dosen K02 selama satu semester yang begitu sabar dalam mengajar dan membagikan ilmu. Tidak lupa, penulis juga berterima kasih atas dukungan seluruh keluarga dan teman yang tak hentinya memotivasi dan menginspirasi penulis. Terakhir, penulis juga ingin mengucapkan terima kasih terhadap bantuan seluruh pihak yang telah berkontribusi dalam penyelesaian makalah ini baik secara langsung maupun secara tidak langsung.

REFERENCES

[1] Munir, Rinaldi. *Matematika Diskrit*. Bandung: Informatika, 2010, edisi ketiga.

[2] Gurari, Eitan. *CIS 680: DATA STRUCTURES: Chapter 19: Backtracking Algorithms.* 1999. Diakses dari : https://web.archive.org/web/20070317015632/http:/www.cse.ohio-state.edu/~gurari/course/cis680/cis680Ch19.html, diakses 8 Desember 17:10 GMT+7.

[3] Donald E. Knuth. *The Art of Computer Programming*. 1968. Addison-Wesley.

[4] https://www.geeksforgeeks.org/hamiltonian-cycle-backtracking-6/, diakses 8 Desember pukul 15:13 GMT+7.

[5] http://www.cs.cmu.edu/~ref/pgss/lecture/7/index.html, diakses 8 Desember pukul 16:20 GMT+7.

[6] http://scanftree.com/Graph-Theory/, diakses 8 Desember pukul 20:03 GMT+7.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 10 Desember 2018

Isa Mujahid Darussalam
13517002