Pemanfaatan Algoritma BFS pada Graf Tak Berbobot untuk Mencari Jalur Terpendek

Aswin Juari
Institut Teknologi Bandung
Jl. Ganesha 10 Bandung Jawa Barat – Indonesia
E-mail: if15076@students.if.itb.ac.id

ABSTRAK

Makalah ini membahas tentang aplikasi Algoritma BFS untuk mencari jalur terpendek pada graf yang tidak berbobot. Maksud dari graf tak berbobot adalah graf yang jarak antar simpulnya sama. Dengan kata lain, ini adalah graf yang semua sisinya bernilai sama. Pada makalah ini, hanya akan dibahas cara mecari jalur terpendek pada graf yang tak berarah saja.

Pemanfaatan algoritma ini dalam mencari jalur terpendek pada graf tak berbobot ini banyak diaplikasikan pada permainan – permainan di mana musuh mampu mengejar pemain sehingga musuh tak 'bodoh' dalam melakukan pengejaran. Permainan komputer yang menggunakan aplikasi jalur terpendek antara lain : *Rodent Revenger, War Craft,* dan *Pac-Man*. Titik awal pengejaran akan menjadi simpul asal dan yang mejadi titik akhir pengejaran adalah posisi pemain(dalam permainan tersebut). Walaupun demikian, algoritma ini masih memiliki kelemahan jika daerah titik awal pengejaran dan titik akhir pengejaran terlalu jauh.

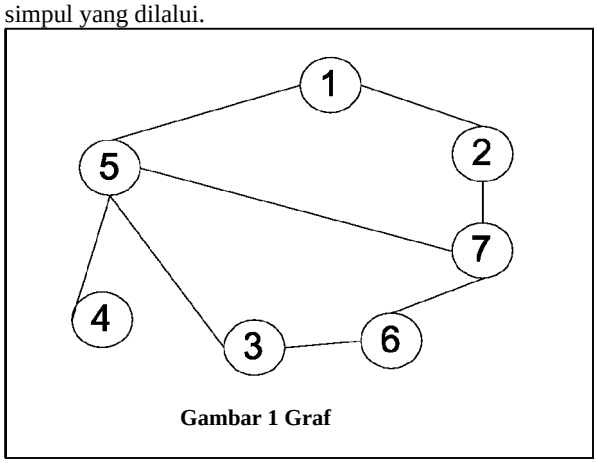
Algoritma BFS(*Breadth First Search*) ini merupakan salah satu strategi pemecahan masalah jalur terpendek pada graf tak berbobot. Algoritma ini mengimplementasikan struktur data *queue*(antrian) dan *linkedlist*(list berkait). Dengan algoritma ini pula kita bisa menghitung jalur terpendek.

Kata Kunci: *linkedlist*, *queue*, matriks, sel, sisi, dan simpul.

1. PENDAHULUAN

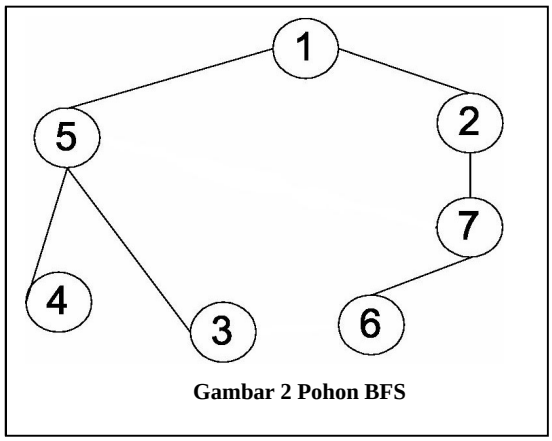
Algoritma BFS(*Breadth First Search*) merupakan algoritma traversal dengan melakukan pencarian melebar pada graf. Setelah ditentukan sebuah simpul awal, strategi dari Breadth First Search adalah secara sistematis melakukan traversal seluruh simpul yang dapat dicapai oleh simpul awal. Pohon BFS yang dimulai dengan simpul akar akan maju satu demi satu level.

Graf tak berbobot adalah graf yang semua sisinya sama, sehingga jalur terpendek hanya ditentukan dari jumlah simpul yang dilalui.



Gambar 1 Graf

Pada contoh graf di atas, misalnya jika kita mulai dari simpul 1 maka urutan simpul yang dikunjungi adalah 1,2,5,7,3,4,6. Algoritma BFS lebih jelasnya sebagai berikut: Kunjungi simpul asal, kemudian kunjungi simpul yang bertetangga dengan simpul asal. Kemudian kunjungi simpul-simpul yang bertetangga dengan simpul yang tadi dikunjungi dan seterusnya. traversal dari algoritma ini tak bersifat unik. Karena traversal inilah, algoritma ini dapat digunakan sebagai algoritma pencarian jalur terpendek. Jika dibuat pohon dengan 1 sebagai akarnya adalah sebagai berikut:



Gambar 2 Pohon BFS

2. APLIKASI ALGORITMA BFS DALAM MASALAH MENCARI JALUR TERPENDEK

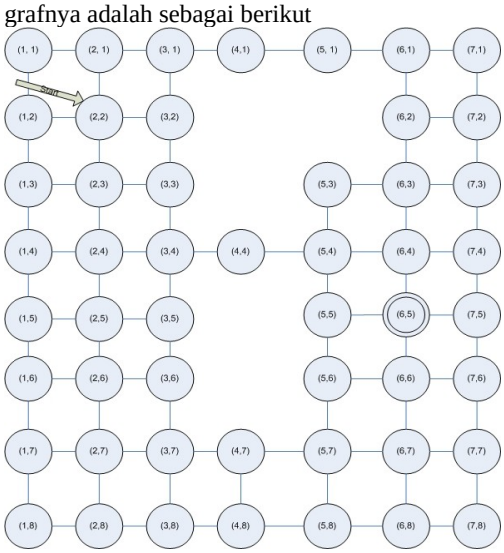
Agar lebih mudah dalam memahami algoritma ini, kita nyatakan dalam matriks karena graf bisa direpresentasikan dalam matriks dan lebih mudah dalam penentuan posisi dan ketetanggaan dalam realisasi algoritma. Diberikan sebuah matriks ukuran n x m:

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	0	0	0	0	0	0	0
2	0	2	0	1	1	0	0
3	0	0	0	1	0	0	0
4	0	0	0	0	0	0	0
5	0	0	0	1	0	3	0
6	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0

m adalah banyaknya kolom pada matriks. n adalah banyaknya baris pada matriks. (x,y) menyatakan koordinat pada matriks. Misalkan m=7 dan n=8. Status 0 menggambarkan bahwa jalan tersebut dapat dilalui, status 0 menggambarkan jalan tersebut tak dapat dilalui, status 2 menggambarkan sel awal, dan status 3 menggambarkan sel yang harus dituju.

Kita ingin mencari jalan terpendek dari (2,2) ke (6,5) tetapi tidak boleh mengambil jalur diagonal. Hal ini mirip dengan graf yang tak berarah di mana sel-sel yang dapat dikunjungi itu sebagai simpul dan jalur yang dapat dilalui sebagai sisi(yang semua nilainya sama). Sel mula-mula juga dapat kita analogikan sebagai simpul awal dan sel tujuan dapat kita analogikan sebagai simpul tujuan.

Jika kita ingin merepresentasikannya dalam graf, maka grafnya adalah sebagai berikut



2.1. Sketsa Penyelesaian (Informal)

Untuk menyelesaikan masalah jalur terpendek dengan menggunakan BFS dapat dilakukan langkah-langkah berikut:

1. Letakkan simpul awal pada antrian
2. Ambil antrian pertama lalu periksa:
   a. Jika simpul awal sudah sama dengan simpul akhir, maka akhiri pencarian dan kembalikan solusi.
   b. Jika tidak, masukkan semua suksesor dari simpul tersebut pada akhir antrian, jika ada.
3. Jika antrian simpul kosong dan semua simpul sudah dicek, akhiri pencarian dan solusi tidak ditemukan.
4. Ulangi mulai langkah 2.

2.2. *Pseudo-Code*

```
Inisialisasi Matriks Boolean yang diset false
Inisialisasi Sebuah Antrian Kosong
Buat Simpul Asal
Masukkan Simpul Asal ke dalam Antrian
Buat Simpul Tujuan
Boolean ketemu=false
int jarak = 1
While AntrianTidakKosong and not ketemu do
        Ambil simpul n dari antrian
        if n=simpul_tujuan then
                ketemu=true
                output(jarak)
                return solusi
        else
                increment (level)
                Bangkitkan suksesor n' dari n
                for setiap suksesor n' dari n
                    if(dikunjungi[posisi x dari
                    n'][posisi y dari n']=false)
                    then
                      Set parent dari n' menjadi n
                      Masukkan n' ke dalam antrian
                endfor
endwhile
```

2.3. Kode Program

Untuk dapat lebih mengerti realisasi algoritma ini, lebih baik jika kita melihat kode programnya secara langsung. Dalam makalah ini, kode program ditulis dalam bahasa java. Representasi dalam matriks.

```
import java.util.*;

//Kelas ini untuk pencarian
class Simpul{
    private int x;//posisi x pada matriks
    private int y;//posisi y pada matriks
    private Simpul parent;//menyatakan parent
    //dari elemen

    //Konstruktor
    public Simpul(int _x,int _y){
         x=_x; y=_y; parent=null;
    }
    public Simpul(int _x,int _y, Simpul p){
         x=_x; y=_y; parent = p;
    }
    //getter
    public int getX(){return x;}
    public int getY(){return y;}
    public int getParent(){return parent;}

    //setter
    public void setX(int _x){x=_x;}
    public void setY(int _y){Y=_y;}
    public void setParent(Simpul p)
                         {parent=n;}

}

//Kelas ini untuk membuat sel pada matriks
class Grid{
    private int x; //posisi x
    private int y; //posisi y
    private int status; //status pada grid
    //0. bisa dikunjungi 1. tak bisa
    //dikunjugi
                         //1. tak bisa
                         //dikunjungi
    //Konstruktor
    public Grid(int _x,int _y){
        x=_x; y=_y;
    }

    //Konstruktor
    public Grid(int _x,int _y){
        x=_x; y=_y;
    }

    //getter
    public int getX(){return x;}
    public int getY(){return y;}
    public int getStatus(){return status;}

    //setter
    public void setX(int _x){x=_x;}
    public void setY(int _y){Y=_y;}
    public void setStatus(int s){status=s;}
}
```

```
//Kelas Pencarian dengan BFS untuk masalah
//di atas
public class BFS{
    public static void main(String args[]){
        //Inisialisasi Matriks
        int n=7;
        int m=8;
        Grid Matriks[][]=new Grid[m+1][n+1];
        for (int i=1;i<m+1;i++){
            for (int j=1;j<n+1;j++){
                Matriks[i][j]=new Grid(i,j);
                Matriks[i][j].setStatus(0);
            }
        }
        Matriks[2][2].setStatus(2);
        Matriks[6][5].setStatus(3);
        Matriks[5][2].setStatus(1);
        Matriks[4][2].setStatus(1);
        Matriks[4][3].setStatus(1);
        Matriks[4][5].setStatus(1);
        Matriks[4][6].setStatus(1);

        //Inisialisasi Matriks boolean yang
        //diset false
        boolean Matbol[][]=new
        boolean[m+1][n+1];
        for (int i=1;i<m+1;i++){
            for (int j=1;j<n+1;j++){
                Matbol[i][j]=false;
            }
        }

        //Inisialisasi Sebuah Antrian Kosong
        LinkedList<Simpul> Antrian=new
        LinkedList<Simpul>();

        //Buat Simpul Asal
        Simpul SimpulAsal=new Simpul(2,2);
        //Masukkan Simpul Asal ke dalam
        //Antrian
        Antrian.addFirst(SimpulAsal);
        //Buat Simpul Tujuan
        Simpul SimpulTujuan=new Simpul(6,5);

        boolean ketemu=false;
        int jarak=1;
        Simpul S;
        while(!(Antrian.size()=0) &&
        (!ketemu)){
            S=Antrian.remove();
            Matbol[S.getX()][S.getY()]=true;

            if((S.getX()==SimpulTujuan.getX(
            )) &&
            (S.getY()==SimpulTujuan.getY()))
            {
                ketemu=true;
                System.out.println("Jarak
                ="+jarak);
                //Solusi telah ditemukan
                While(S!=SimpulAsal){
                    //tandai jalan
                    Matriks[S.getX()]
                    [S.getY()].
                    setStatus(5);
                    S=S.getParent();
                }
            }
```

```
            else{
                jarak++;
                //Bangkitkan suksesor dan
                //masukkan ke antrian
                Simpul S_suk[]=new Simpul[4];
                //generate Kiri
                if((S.getX()-1)>0)) {
                  if ((MatBol[S.getX()-1]
                [S.getY()]==false) &&
                (Matriks[S.getX()-1]
                [S.getY()].getStatus()==0))
                   {
                 S_suk[0]=new Simpul(S.getX()-
                1,S.getY(),S);
                Antrian.addLast(S_suk[0]);
                   }
                }
                //generate kanan
                if((S.getX()+1)<=m)) {
                  if ((MatBol[S.getX()+1]
                [S.getY()]==false) &&
                (Matriks[S.getX()+1]
                [S.getY()].getStatus()==0))
                   {
                 S_suk[1]=new
                Simpul(S.getX()+1,S.getY(),S);
                Antrian.addLast(S_suk[1]);
                   }
                }

               //generate atas
                if((S.getY()-1)>0)) {
                   if ((MatBol[S.getX()]
                [S.getY()-1]==false) &&
                (Matriks[S.getX()] [S.getY()-
                1].getStatus()==0))
                   {
                 S_suk[2]=new
                Simpul(S.getX(),S.getY()-1,S);
                Antrian.addLast(S_suk[2]);
                   }
                }
                //generate bawah
                if((S.getY()+1)<=n)) {
                  if ((MatBol[S.getX()]
                [S.getY()+1]==false) &&
                (Matriks[S.getX()]
                [S.getY()+1].getStatus()==0))
                   {
                 S_suk[3]=new
                Simpul(S.getX(),S.getY()+1,S);
                Antrian.addLast(S_suk[3]);
                    }
                }
            }
        }
    }
}
```

3. ANALISIS

Metode BFS dapat menjamin ditemukannya suatu solusi jika memang terdapat solusi. Akan tetapi kompleksitas waktu algoritma ini kurang begitu baik sehingga jika daerah pencariannya luas maka akan membutuhkan waktu komputasi yang cukup lama.

Lebih jelasnya, pada level ke-1 dari pohon pencarian BFS akan simpul akar membangkitkan n buah simpul baru. Pada level ke-2 jika setiap simpul tersebut menghasilkan kembali simpul yang baru maka akan menghasilkan n^2 simpul. Begitu pula pada level 3 dan seterusnya. Jika dirumuskan secara matematik, jumlah simpul yang dibangkitkan:

$$1 + n + n^2 + n^3 + \ldots n^x = (n^{x+1}-1)/(n-1)$$

Jadi, kompleksitas waktunya : $O(n^x)$.

Oleh karena itu, pencarian dengan BFS tidak efektif jika daerah pencariannya terlalu besar karena akan memakan waktu(membuat program terlihat '*hang*'). Ada algoritma yang lebih baik daripada BFS untuk mencari jalan terpendek pada graf tak berarah karena algoritma tersebut menggunakan prinsip algoritma *Branch and Bound*, yaitu menggunakan prinsip ongkos pencarian.

4. Kesimpulan

1. Untuk mencari jalur terpendek pada graf tek berbobot dapat digunakan algoritma BFS walaupun algoritma ini bukan merupakan solusi yang terbaik.

2. Prinsip pencarian jalur terpendek pada matriks pada dasarnya sama dengan graf tak berbobot dan juga tak berarah.

REFERENSI

[1] Munir, Rinaldi. "Diktat Kuliah IF2251 Strategi Algoritmik". Bandung: Program Studi Teknik Informatika ITB. 2007.

[2]http://en.wikipedia.org/wiki/breadth-first_search. Tanggal akses: 21 Mei 2007 pukul 17.00.