

LAPORAN PRAKTIKUM 6

ANALISIS ALGORITMA



Disusun oleh :

Alvin

140810180013

Kelas A

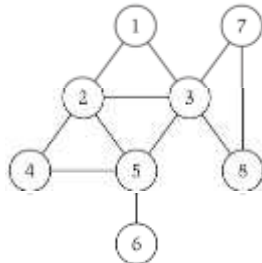
Program Studi S-1 Teknik Informatika
Departemen Ilmu Komputer
Fakultas Matematika dan Ilmu Pengetahuan
Alam
Universitas Padjadjaran
2020

Pendahuluan

Graf Tak Berarah (Undirected Graf)

(Undirected) graph: $G=(V,E)$

- V = sekumpulan node (vertex, simpul, titik, sudut)
- E = sekumpulan edge (garis, tepi)
- Menangkap hubungan berpasangan antar objek.
- Parameter ukuran Graf: $n = |V|$, $m=|E|$



$V = \{1,2,3,4,5,6,7,8\}$

$E = \{(1,2), (1,3), (2,3), (2,4), (2,5), (3,5), (3,7), (3,8), (4,5), (5,6), (7,8)\}$

$n = 8$

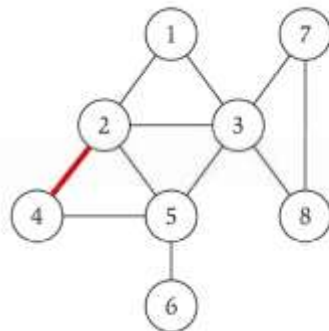
$M=11$

Dalam pemrograman, Graf dapat direpresentasikan dengan **adjacency matrix** dan **adjacency list**

Representasi Graf dengan Adjacency Matrix

Adjacency Matrix: n-ke-n matriks dengan

- Dua representasi dari setiap sisi
- Ruang berukuran sebesar
- Memeriksa apakah (u, v) edge membutuhkan waktu $\Theta(1)$
- Mengidentifikasi semua tepi membutuhkan $\Theta(n^2)$ waktu

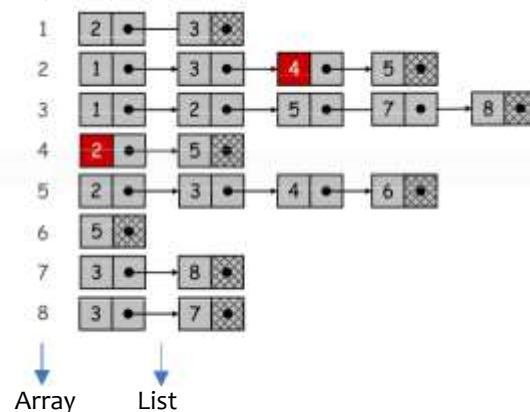
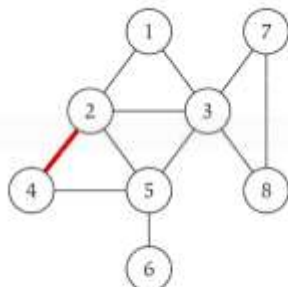


	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	1	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

Representasi Graf dengan Adjacency List

Adjacency List: node diindeks sebagai list

- Dua representasi untuk setiap sisi
- Ukuran ruang $m + n$
- Memeriksa apakah (u, v) edge membutuhkan $O(\deg(u))$. Degree = jumlah tetangga u.
- Mengidentifikasi semua tepi membutuhkan $\Theta(m + n)$.

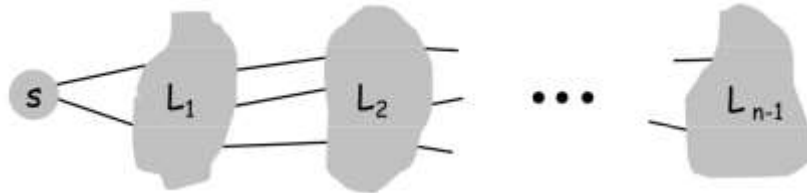


Breadth First Search

Intuisi BFS. Menjelajahi alur keluar dari s ke semua arah yang mungkin, tambahkan node satu "layer" sekaligus.

Algoritma BFS

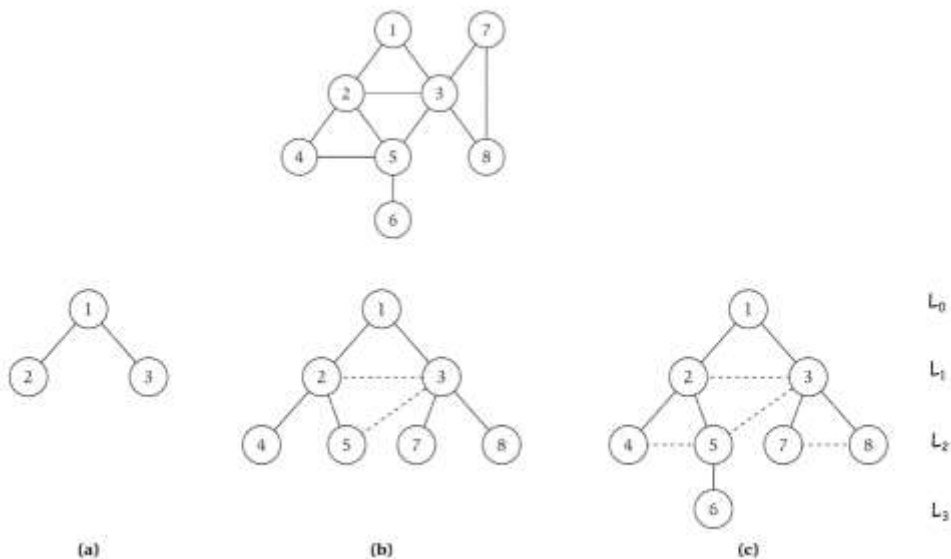
- $L_0 = \{s\}$
- $L_1 =$ semua tetangga dari L_0
- $L_2 =$ semua node yang tidak termasuk ke dalam L_0 atau L_1 , dan yang mempunyai edge ke sebuah node di L_1
- $L_i + 1 =$ semua node yang bukan milik layer sebelumnya, dan yang memiliki edge ke node di L_i



Gambar 1. Ilustrasi algoritma BFS

Teorema 1.0

Untuk setiap t , terdiri dari semua node pada jarak tepat t dari s . Ada path dari s ke t jika t muncul di beberapa layer.



Gambar 2. Ilustrasi pembentukan tree BFS dari undirected Graf

Implementasi BFS dalam Koding Program

- Adjacency list adalah representasi struktur data paling ideal untuk BFS
- Algoritma memeriksa setiap ujung yang meninggalkan node satu per satu. Ketika kita memindai edge yang meninggalkan u dan mencapai $edge(u, v)$, kita perlu tahu apakah node v telah ditemukan sebelumnya oleh pencarian.
- Untuk menyederhanakan ini, kita maintain array yang ditemukan dengan panjang n dan mengatur $Discovered[v] = \text{true}$ segera setelah pencarian kita pertama kali melihat v . Algoritma BFS membangun lapisan node L_1, L_2, \dots , di mana L_i adalah set node pada jarak i dari sumber s .
- Untuk mengelola node dalam layer L_i , kami memiliki daftar $list_i$ untuk setiap $i = 0, 1, 2, \dots$

Depth First Search

Algoritma BFS muncul, khususnya, sebagai cara tertentu mengurutkan node yang kita kunjungi — dalam lapisan berurutan, berdasarkan pada jarak node lain dari s . Metode alami lain untuk menemukan node yang dapat dijangkau dari s adalah pendekatan yang mungkin Anda lakukan jika grafik G benar-benar sebuah labirin dari kamar yang saling berhubungan dan kita berjalan-jalan di dalamnya.

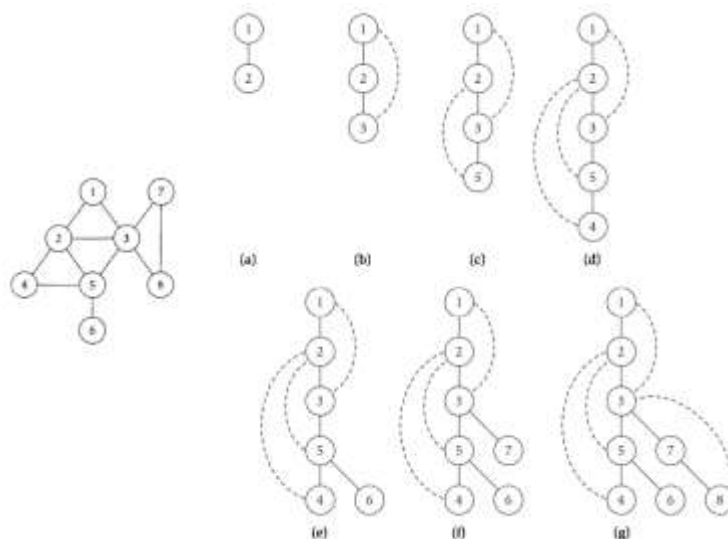
Kita akan mulai dari s dan mencoba edge pertama yang mengarah ke ke node v . Kita kemudian akan mengikuti edge pertama yang mengarah keluar dari v , dan melanjutkan dengan cara ini sampai kita mencapai "jalan buntu" — sebuah node di mana Anda sudah menjelajahi semua tetangganya. Kita kemudian akan mundur sampai kita mencapai node dengan tetangga yang belum dijelajahi, dan melanjutkan dari sana. Kita menyebutnya Depth-first search (DFS), karena ini mengeksplorasi G dengan masuk sedalam mungkin dan hanya mundur jika diperlukan.

DFS juga merupakan implementasi khusus dari algoritma component-growing generik yang dijelaskan sebelumnya. Kita dapat memulai DFS dari titik awal mana pun tetapi mempertahankan pengetahuan global tentang node yang telah dieksplorasi.

```
DFS( $u$ ):  
  Mark  $u$  as "Explored" and add  $u$  to  $R$   
  For each edge  $(u, v)$  incident to  $u$   
    If  $v$  is not marked "Explored" then  
      Recursively invoke DFS( $v$ )  
    Endif  
  Endfor
```

Untuk menerapkan ini pada problem konektivitas s - t , kita cukup mendeklarasikan semua node pada awalnya untuk tidak dieksplorasi, dan memanggil DFS (s).

Ada beberapa kesamaan dan beberapa perbedaan mendasar antara DFS dan BFS. Kesamaan didasarkan pada fakta bahwa mereka berdua membangun komponen terhubung yang mengandung s , dan bahwa mereka mencapai tingkat efisiensi yang serupa secara kualitatif. Sementara DFS akhirnya mengunjungi set node yang sama persis seperti BFS, ia biasanya melakukannya dalam urutan yang sangat berbeda; menyelidiki jalan panjang, berpotensi menjadi sangat jauh dari s , sebelum membuat cadangan untuk mencoba lebih dekat node yang belum dijelajahi.



Gambar 3. Ilustrasi pembentukan tree DFS dari undirected graph

Implementasi BFS dalam Koding Program

Implementasi DFS paling ideal adalah dengan menggunakan stack. Adapun algoritma DFS dengan stack adalah sebagai berikut:

```
DFS(s):
  Initialize S to be a stack with one element s
  While S is not empty
    Take a node u from S
    If Explored[u] = false then
      Set Explored[u] = true
      For each edge (u,v) incident to u
        Add v to the stack S
      Endfor
    Endif
  Endwhile
```

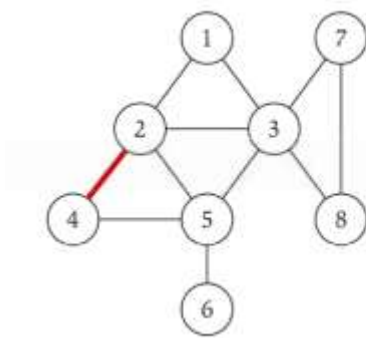
Nama : Alvin

NPM : 140810180013

TUGAS-6

Tugas Anda

1. Dengan menggunakan *undirected graph* dan *adjacency matrix* berikut, buatlah koding programnya menggunakan bahasa C++.



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

• Code

```
/*      Nama      : Alvin
      NPM       : 140810180013
      Kelas    : A
      Program : Representasi Graph dengan Matrix
*/
#include<iostream>
using namespace std;

int main(){
  int m[8][8] = {
    {0,1,1,0,0,0,0,0},
    {1,0,1,1,1,0,0,0},
    {1,0,1,1,1,0,0,0},
    {1,1,0,0,1,0,1,1},
    {0,1,0,1,1,0,0,0},
  }
```

```

{0,1,1,1,0,1,0,0},
{0,0,0,0,1,0,0,0},
{0,0,1,0,0,0,0,1},
{0,0,1,0,0,0,1,0}
};
cout<<"MATRIX-ADJACENCY\n"<<endl;
for(int i=0;i<8;i++){
    for(int j=0; j<8;j++){
        cout<<m[i][j]<<" ";
    }
    cout<<endl;
}
}

```

- Screenshot

```

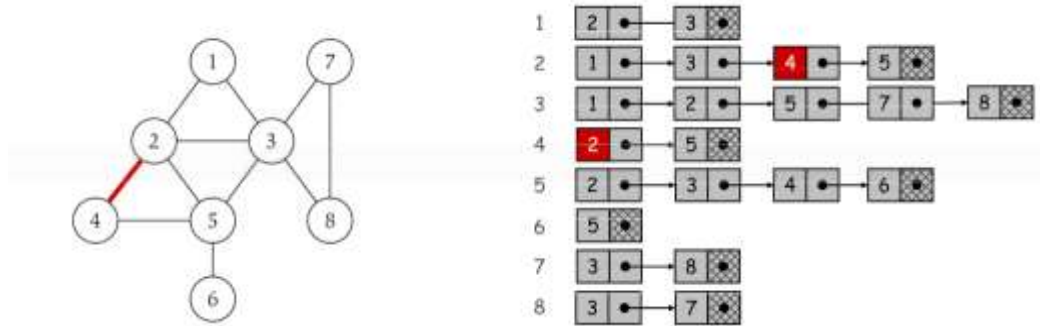
D:\SEMESTER-4\ANALGO\PRAKTIKUM\AnalgoKu\AnalgoKu6\1.Adjacency-Matrix.exe
MATRIX-ADJACENCY

0 1 1 0 0 0 0 0
1 0 1 1 1 0 0 0
1 1 0 0 1 0 1 1
0 1 0 1 1 0 0 0
0 1 1 1 0 1 0 0
0 0 0 0 1 0 0 0
0 0 1 0 0 0 0 1
0 0 1 0 0 0 1 0

-----
Process exited after 0.1129 seconds with return value 0
Press any key to continue . . .

```

2. Dengan menggunakan *undirected graph* dan representasi *adjacency list*, buatlah koding programnya menggunakan bahasa C++.



• Code

```

/*      Nama      : Alvin
      NPM       : 140810180013
      Kelas    : A
      Program   : Representasi Graph dengan List
*/
#include<iostream>
#include<windows.h>
using namespace std;

struct adjacent{
    int nodeAdj;
    adjacent* nextAdj;
};

struct elemen{
    int node;
    elemen* next;
    adjacent* firstAdj;
};

typedef elemen* pointerNode;
typedef adjacent* pointerAdj;
typedef pointerNode list;

void createListNode(list& first){
    first = NULL;
}

void createNode(pointerNode& pBaru,int vertex){
    pBaru = new elemen;
    pBaru->node = vertex;
    pBaru->next = NULL;
    pBaru->firstAdj = NULL;
}

void createAdjacent(pointerAdj& pBaru,int vertex){
    pBaru = new adjacent;
    pBaru->nodeAdj = vertex;
    pBaru->nextAdj = NULL;
}

void insertAdjacent(pointerNode& curNode,pointerAdj pBaruAdj){
    pointerAdj last;
    if(curNode->firstAdj == NULL){
        curNode->firstAdj = pBaruAdj;
    }else{
        last = curNode->firstAdj;
        while(last->nextAdj != NULL){

```

```

        last = last->nextAdj;
    }
    last->nextAdj = pBaruAdj;
}

void insertElement(list& first, pointerNode pBaruNode, int size){
    pointerNode last;
    pointerAdj pBaruAdj;
    if(first == NULL){
        first = pBaruNode;
    }else{
        last = first;
        while(last->next != NULL){
            last = last->next;
        }
        last->next = pBaruNode;
    }
    if(size>0){
        cout<<"Masukan node yang berhubungan dengan "<<pBaruNode->node<<" : "<<endl;
    }
    for(int i = 0; i < size; i++){
        int vertex;
        cin>>vertex;
        createAdjacent(pBaruAdj,vertex);
        insertAdjacent(pBaruNode,pBaruAdj);
    }
}

void output(list first){
    pointerNode pOut;
    pointerAdj pOutAdj;
    if(first == NULL){
        cout<<"Tidak ada Node"<<endl;
    }else{
        pOut = first;

        while(pOut != NULL){
            cout<<"Parent = "<<pOut->node<<endl;
            if(pOut->firstAdj == NULL){
                cout<<"Tidak ada adjacency"<<endl;
            }else{
                pOutAdj = pOut->firstAdj;
                cout<<"Child = ";
                while(pOutAdj != NULL){
                    cout<<pOutAdj->nodeAdj<<" ";
                    pOutAdj = pOutAdj->nextAdj;
                }
            }
            cout<<endl;
            pOut = pOut->next;
        }
    }
}

int main(){
    list first;
    pointerNode node;
    cout<<"==== MEMBUAT NODE =====\n"<<endl;
    createListNode(first);

    createNode(node,1);
    insertElement(first,node,2);
    createNode(node,2);
    insertElement(first,node,4);
    createNode(node,3);
}

```



```

insertElement(first,node,5);
createNode(node,4);
insertElement(first,node,2);
createNode(node,5);
insertElement(first,node,4);
createNode(node,6);
insertElement(first,node,1);
createNode(node,7);
insertElement(first,node,2);
createNode(node,8);
insertElement(first,node,2);
cout<<"==== HASIL ====\n"<<endl;
output(first);
system("pause");
}

```

- **Screenshot**

```

D:\SEMESTER-4\ANALGO\PRAKTIKUM\AnalgoKu\AnalgoKu6\2.Adjacency-List.exe
===== MEMBUAT NODE =====
Masukan node yang berhubungan dengan 1 :
2
3
Masukan node yang berhubungan dengan 2 :
1
3
4
5
Masukan node yang berhubungan dengan 3 :
1
2
5
7
8
Masukan node yang berhubungan dengan 4 :
2
5
Masukan node yang berhubungan dengan 5 :
2
3
4
6
Masukan node yang berhubungan dengan 6 :
5
Masukan node yang berhubungan dengan 7 :
3
8
Masukan node yang berhubungan dengan 8 :
3
7

```

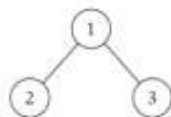
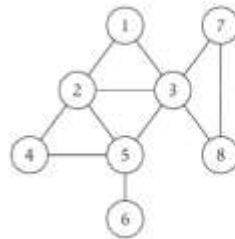
```

===== HASIL =====

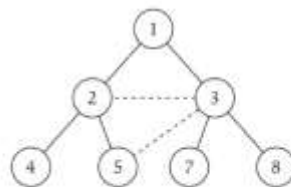
Parent = 1
Child = 2 3
Parent = 2
Child = 1 3 4 5
Parent = 3
Child = 1 2 5 7 8
Parent = 4
Child = 2 5
Parent = 5
Child = 2 3 4 6
Parent = 6
Child = 5
Parent = 7
Child = 3 8
Parent = 8
Child = 3 7
Press any key to continue . . .

```

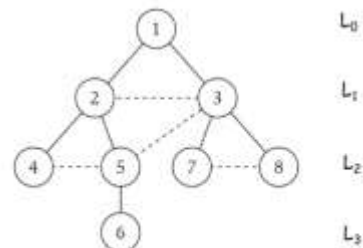
3. Buatlah program Breadth First Search dari algoritma BFS yang telah diberikan. Kemudian uji coba program Anda dengan menginputkan *undirected graph* sehingga menghasilkan tree BFS. Hitung dan berikan secara asimptotik berapa kompleksitas waktunya dalam Big- Θ !



(a)



(b)



(c)

• Code

```

/*      Nama      : Alvin
      NPM       : 140810180013
      Kelas    : A
      Program : Breadth First Search
*/
#include<iostream>
using namespace std;

int main(){
    int vertexSize = 8;
    int adjacency[8][8] = {
        {0,1,1,0,0,0,0,0}, //representasi graph dalam matrix adjacency
        {1,0,1,1,1,0,0,0},
        {1,1,0,0,1,0,1,1},

```

```

        {0,1,0,0,1,0,0,0},
        {0,1,1,1,0,1,0,0},
        {0,0,0,0,1,0,0,0},
        {0,0,1,0,0,0,0,1},
        {0,0,1,0,0,0,1,0}
    };

    bool discovered[vertexSize];
    for(int i = 0; i < vertexSize; i++){
        discovered[i] = false;
    }
    int output[vertexSize];

    //inisialisasi start
    discovered[0] = true;
    output[0] = 1;

    int counter = 1;
    for(int i = 0; i < vertexSize; i++){
        for(int j = 0; j < vertexSize; j++){
            if((adjacency[i][j] == 1)&&(discovered[j] == false)){
                output[counter] = j+1;
                discovered[j] = true;
                counter++;
            }
        }
    }

    cout<<"=== HASIL DARI BREADTH FIRST SEARCH === "<<endl;
    for(int i = 0; i < vertexSize; i++){
        cout<<output[i]<<" ";
    }
}

```

- **Screenshot**

```

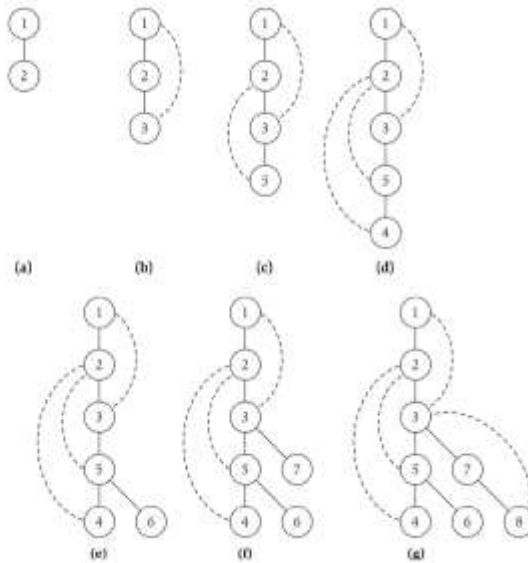
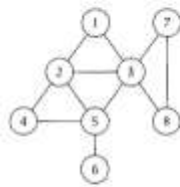
D:\SEMESTER-4\ANALGO\PRAKTIKUM\AnalgoKu\AnalgoKu6\3.Breadth-First-Search.exe
=== HASIL DARI BREADTH FIRST SEARCH ===
1 2 3 4 5 7 8 6
-----
Process exited after 0.2025 seconds with return value 0
Press any key to continue . . .

```

- **Analisa**

BFS merupakan metode pencarian secara melebar sehingga mengunjungi node dari kiri ke kanan di level yang sama. Apabila semua node pada suatu level sudah dikunjungi semua, maka akan berpindah ke level selanjutnya. Dalam worst case BFS harus mempertimbangkan semua jalur (path) untuk semua node yang mungkin, maka nilai kompleksitas waktu dari BFS adalah $O(|V| + |E|)$.

4. Buatlah program Depth First Search dari algoritma DFS yang telah diberikan. Kemudian uji coba program Anda dengan menginputkan *undirected graph* sehingga menghasilkan tree DFS. Hitung dan berikan secara asimptotik berapa kompleksitas waktunya dalam Big-Θ!



• Code

```

/*      Nama      : Alvin
      NPM       : 140810180013
      Kelas    : A
      Program : Depth First Search
*/
#include<bits/stdc++.h>
using namespace std;

class Graph
{
    int V;

    list<int> *adj;

    void DFSUtil(int v, bool visited[]);
public:
    Graph(int V);

    void addEdge(int v, int w);

    void DFS(int v);
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w);
}

void Graph::DFSUtil(int v, bool visited[])
{
    visited[v] = true;
    cout << v << " ";
}

```

```

        list<int>::iterator i;
        for (i = adj[v].begin(); i != adj[v].end(); ++i)
            if (!visited[*i])
                DFSUtil(*i, visited);
    }

void Graph::DFS(int v)
{
    bool *visited = new bool[V];
    for (int i = 0; i < V; i++)
        visited[i] = false;

    DFSUtil(v, visited);
}

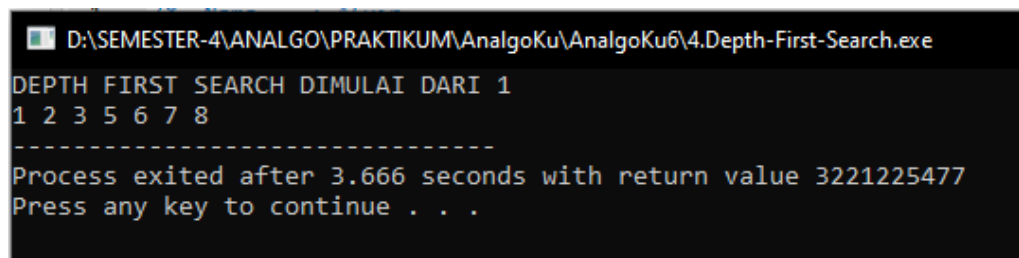
int main()
{
    Graph g(8); //jumlah node
    g.addEdge(1, 2);
    g.addEdge(1, 3);
    g.addEdge(2, 3);
    g.addEdge(2, 4);
    g.addEdge(2, 5);
    g.addEdge(3, 5);
    g.addEdge(3, 7);
    g.addEdge(3, 8);
    g.addEdge(4, 5);
    g.addEdge(5, 6);
    g.addEdge(7, 8);

    cout << "DEPTH FIRST SEARCH DIMULAI DARI 1 \n";
    g.DFS(1);

    return 0;
}

```

- Screenshot



```

D:\SEMESTER-4\ANALGO\PRAKTIKUM\AnalgoKu\AnalgoKu6\4.Depth-First-Search.exe
DEPTH FIRST SEARCH DIMULAI DARI 1
1 2 3 5 6 7 8
-----
Process exited after 3.666 seconds with return value 3221225477
Press any key to continue . . .

```

- Analisa

DFS merupakan metode pencarian mendalam, yang mengunjungi semua node dari yang terkiri lalu geser ke kanan hingga semua node dikunjungi. Kompleksitas ruang algoritma DFS adalah $O(bm)$, karena kita hanya perlu menyimpan satu buah lintasan tunggal dari akar sampai daun, ditambah dengan simpul-simpul saudara kandungnya yang belum dikembangkan.