

LAPORAN PRAKTIKUM 6

Worksheet 6

Analysis Algorithm

Dasar Analisis Algoritma Graf

Studi Kasus : Breadth First Search and Depth First Search



Disusun oleh :

**Mohammad Dhikri
140810180075**

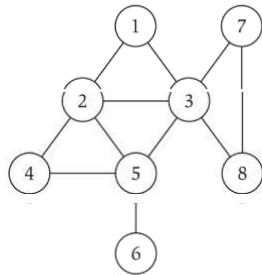
**PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS PADJADJARAN
2020**

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: $n = |V|$, $m = |E|$



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

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

$n = 8$

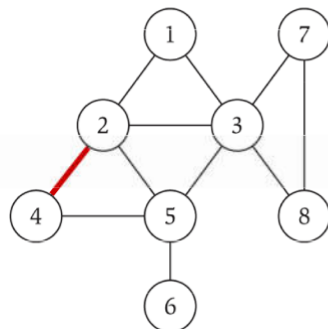
$M=11$

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
- Memeriksa apakah (u, v) edge membutuhkan waktu $\Theta(1)$
- Mengidentifikasi semua tepi membutuhkan $\Theta(n^2)$ waktu

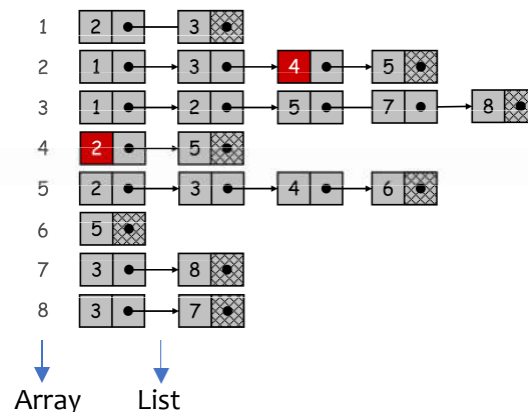
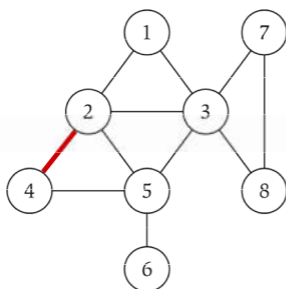


	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	1	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

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 $\Theta(m + n)$.

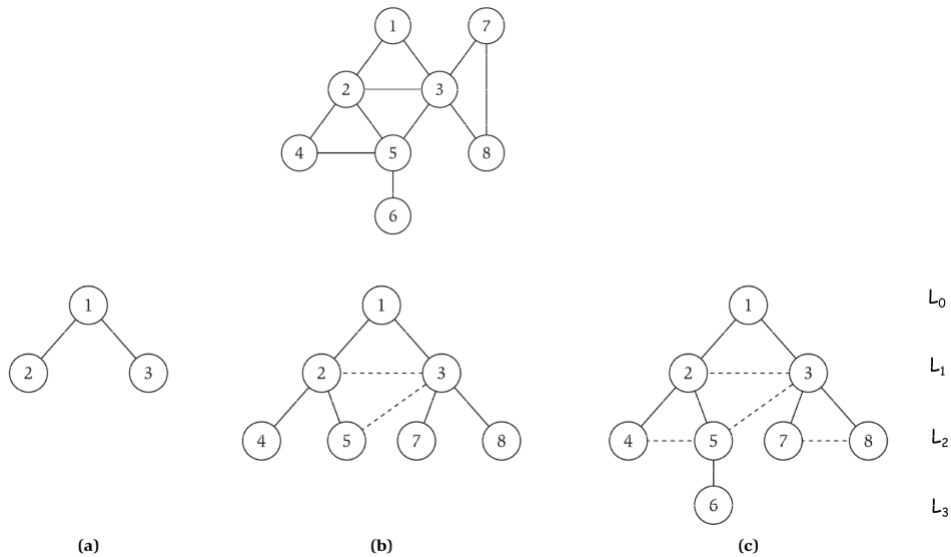


Breadth First Search

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

Teorema 1.0

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



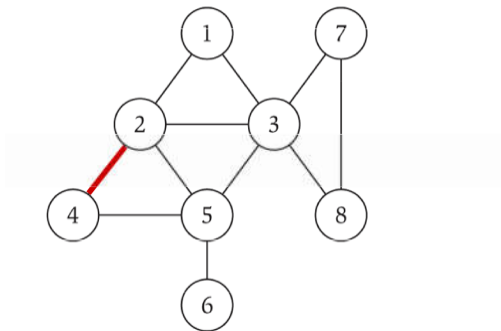
Gambar 2. Ilustrasi pembentukan tree BFS dari undirected Graf

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 $Discovered[v] = true$ 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 dari sumber s .
- Untuk mengelola node dalam layer, kami memiliki daftar $[]$ untuk setiap $i = 0, 1, 2, \dots$.

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

Program :

```
#include <iostream>
#include <cstdlib>
using namespace std;
#define MAX 20

class AdjacencyMatrix{
private:
    int n;
    int **adj;
    bool *visited;
public:
    AdjacencyMatrix(int n){
        this->n = n;
        visited = new bool [n];
        adj = new int* [n];
        for (int i = 0; i < n; i++){
            adj[i] = new int [n];
            for(int j = 0; j < n; j++){
                adj[i][j] = 0;
            }
        }
    }
    void add_edge(int origin, int destin){
        if( origin > n || destin > n || origin < 0 || destin < 0){
            cout<<"Invalid edge!\n";
        }
        else{
            adj[origin - 1][destin - 1] = 1;
        }
    }
    void display(){
        int i,j;
        for(i = 0;i < n;i++){
            for(j = 0; j < n; j++)
```

```

        cout<<adj[i][j]<<" ";
        cout<<endl;
    }
}
};
int main()
{
    int nodes, max_edges, origin, destin;
    cout<<"Enter number of nodes: ";
    cin>>nodes;
    AdjacencyMatrix am(nodes);
    max_edges = nodes * (nodes - 1);
    for (int i = 0; i < max_edges; i++){
        cout<<"Enter edge (-1 -1 to exit): ";
        cin>>origin>>destin;
        if((origin == -1) && (destin == -1))
            break;
        am.add_edge(origin, destin);
    }
    am.display();
    return 0;
}

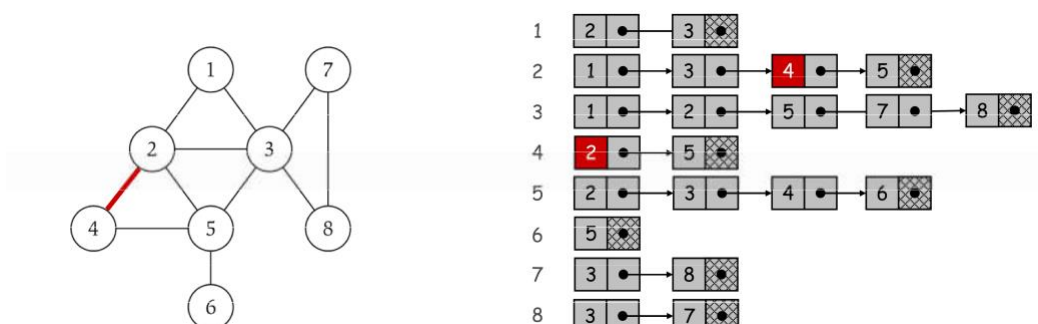
```

```

Enter number of nodes: 3
Enter edge (-1 -1 to exit): 2
3
Enter edge (-1 -1 to exit): 3
4
Invalid edge!
Enter edge (-1 -1 to exit): 5
6
Invalid edge!
Enter edge (-1 -1 to exit): 7
8
Invalid edge!
Enter edge (-1 -1 to exit): 4
6
Invalid edge!
Enter edge (-1 -1 to exit): 3
5
Invalid edge!
0 0 0
0 0 1
0 0 0

```

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



Program :

```
#include <iostream>
#include <cstdlib>
using namespace std;
struct AdjListNode{
    int data;
    struct AdjListNode* next;
};
struct AdjList {
    struct AdjListNode *head;
};
class Graph{
private:
    int V;
    AdjList* array;
public:
    Graph(int V){
        this->V = V;
        array = new AdjList [V];
        for (int i = 0; i < V; ++i)
            array[i].head = NULL;
    }
    void addEdge(int src, int dest){
        AdjListNode* newNode = new AdjListNode;
        newNode->data = dest;
        newNode->next = NULL;          // 1----->NULL

        newNode->next = array[src].head;
        array[src].head = newNode;      // 0-->1-->2

        newNode = new AdjListNode;
        newNode->data = src;
        newNode->next = NULL;           // 0--->NULL

        newNode->next = array[dest].head; // 0---->NULL (bcuz.. 1--
>NULL)
        array[dest].head = newNode;     // 1----->0
    }
    void printGraph(){
        int v;
        for (v = 0; v < V; ++v){
            AdjListNode* tmp = array[v].head;
            cout<<"\n Adjacency list of vertex "<<v<<"\n head ";
            while (tmp){
                cout<<"-> "<<tmp->data;
                tmp = tmp->next;
            }
            cout<<endl;
        }
    }
}
```

```

    }
};
int main()
{
    Graph gh(5);
    gh.addEdge(0, 1);
    gh.addEdge(0, 4);
    gh.addEdge(1, 2);
    gh.addEdge(1, 3);
    gh.addEdge(1, 4);
    gh.addEdge(2, 3);
    gh.addEdge(3, 4);

    gh.printGraph();
    return 0;
}

Adjacency list of vertex 0
head -> 4-> 1

Adjacency list of vertex 1
head -> 4-> 3-> 2-> 0

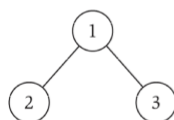
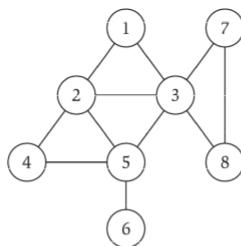
Adjacency list of vertex 2
head -> 3-> 1

Adjacency list of vertex 3
head -> 4-> 2-> 1

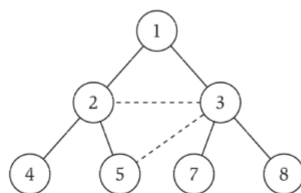
Adjacency list of vertex 4
head -> 3-> 1-> 0

```

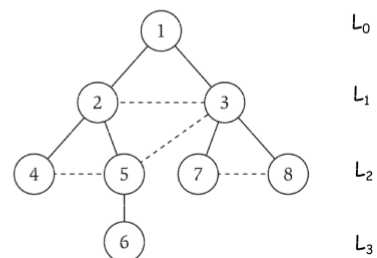
3. Buatlah program Breadth First Search dari algoritma BFS yang telah diberikan. Kemudian uji coba program Anda dengan menginputkan *undirected graph* sehingga menghasilkan tree BFS. Hitung dan berikan secara asimtotik berapa kompleksitas waktunya dalam Big- Θ !



(a)



(b)



(c)

L_0

L_1

L_2

L_3

Program :

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std;
int cost[10][10],i,j,k,n,qu[10],front,rare,v,visit[10],visited[10];
int main()
{
    int m;
    //clrscr();
    cout <<"Enter no of vertices:";
    cin >> n;
    cout <<"Enter no of edges:";
    cin >> m;
    cout <<"\nEDGES \n";
    for(k=1; k<=m; k++)
    {
        cin >>i>>j;
        cost[i][j]=1;
    }
    cout <<"Enter initial vertex to traverse from:";
    cin >>v;
    cout <<"Visited vertices:";
    cout <<v<<" ";
    visited[v]=1;
    k=1;
    while(k<n)
    {
        for(j=1; j<=n; j++)
            if(cost[v][j]!=0 && visited[j]!=1 && visit[j]!=1)
            {
                visit[j]=1;
                qu[rare++]=j;
            }
        v=qu[front++];
        cout<<v <<" ";
        k++;
        visit[v]=0;
        visited[v]=1;
    }
    getch();
    return 0;
}
```

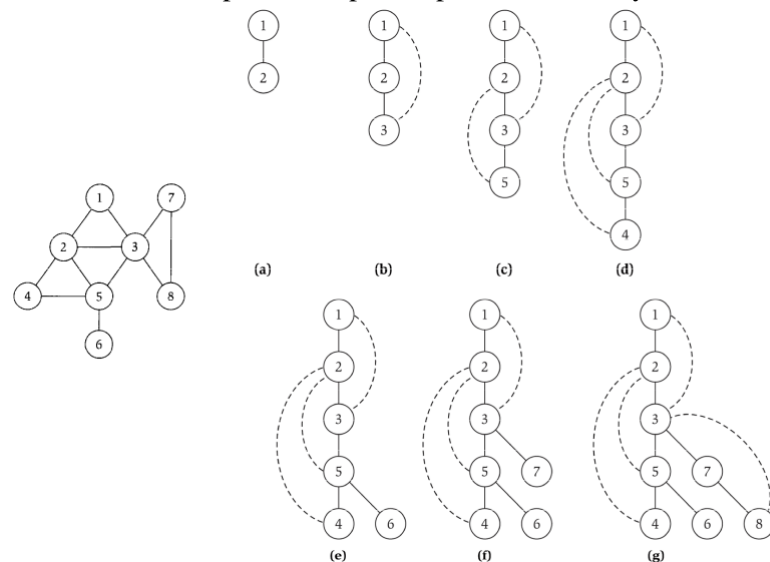


```

Enter no of vertices:4
Enter no of edges:4
EDGES
1 2
3 4
5 6
7 8
Enter initial vertex to traverse from:3
Visited vertices:3 4 0 0
-----
Process exited after 33.05 seconds with return value 0
Press any key to continue . . .

```

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 !



Program :

```

#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std;
int cost[10][10],i,j,k,n,stk[10],top,v,visit[10],visited[10];
int main()
{
    int m;
    //clrscr();
    cout <<"Enter no of vertices:";
    cin >> n;
    cout <<"Enter no of edges:";
    cin >> m;
    cout <<"\nEDGES \n";
    for(k=1; k<=m; k++)
    {
        cin >>i>>j;
    }
}

```

```

        cost[i][j]=1;
    }
    cout <<"Enter initial vertex to traverse from:";
    cin >>v;
    cout <<"DFS ORDER OF VISITED VERTICES:";
    cout << v <<" ";
    visited[v]=1;
    k=1;
    while(k<n)
    {
        for(j=n; j>=1; j--)
            if(cost[v][j]!=0 && visited[j]!=1 && visit[j]!=1)
            {
                visit[j]=1;
                stk[top]=j;
                top++;
            }
        v=stk[--top];
        cout<<v << " ";
        k++;
        visit[v]=0;
        visited[v]=1;
    }
    getch();
    return 0;
}

```

```

Enter no of vertices:4
Enter no of edges:4
space std;
EDGES ],i,j,k,n,stk[10],top,v,visit[10],visited[10];
1 2
3 4
5 6 );
7 8
Enter no of vertices:";
Enter initial vertex to traverse from:4
DFS ORDER OF VISITED VERTICES:4 4 2 0
=====
Process exited after 27.92 seconds with return value 0
Press any key to continue . . .

```