LAPORAN PRAKTIKUM 6

STUDI KASUS: BREADTH FIRST SEARCH DAN DEPTH FIRST SEACRH

MATA KULIAH
ANALISIS ALGORITMA
D10G.4205 & D10K.0400601

PENGAJAR : **(1) MIRA SURYANI, S.Pd., M.Kom**
(2) INO SURYANA, Drs., M.Kom
(3) R. SUDRAJAT, Drs., M.Si
FAKULTAS : **MIPA**
SEMESTER : **IV dan VI**

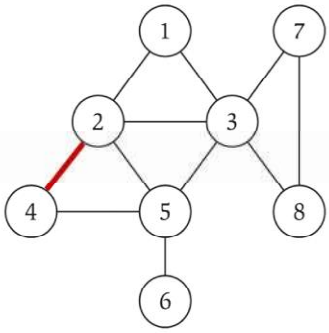
DISUSUN OLEH : **AHMAD FAAIZ A (140810180023)**

PROGRAM STUDI S-1 TEKNIK INFORMATIKA
DEPARTEMEN ILMU KOMPUTER
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS PADJADJARAN

APRIL 2019

Tugas Anda

1. Dengan menggunakan *undirected graph* dan *adjacency matrix* berikut, buatlah koding programmnya 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

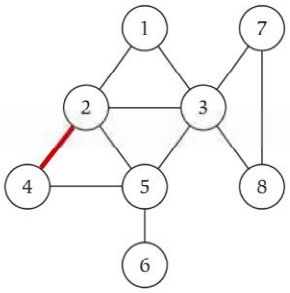
Jawab:

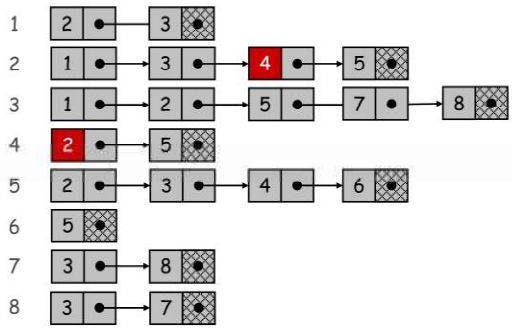
```
#include <iostream>
using namespace std;
int vertArr[10][10];

void displayMatrix(int v)
{
    int i, j;
    for (int i = 1; i <= v; i++)
        i == 1 ? cout << "  " << i : cout << " " << i;
    cout << endl;

    for (i = 1; i <= v; i++)
    {
        cout << i << " ";

        for (j = 1; j <= v; j++)
        {
            cout << vertArr[i][j] << " ";
        }
        cout << endl;
    }
}
void addEdge(int u, int v)
{
    vertArr[u][v] = 1;
    vertArr[v][u] = 1;
}
main(int argc, char *argv[])
{
    int v = 8;

    addEdge(1, 2);
    addEdge(1, 3);

    addEdge(2, 3);
    addEdge(2, 4);
    addEdge(2, 5);

    addEdge(3, 5);
    addEdge(3, 7);
    addEdge(3, 8);

    addEdge(4, 5);
    addEdge(5, 6);
    addEdge(7, 8);

    displayMatrix(v);
}
```

Scrennshot:

```
Prak\Worksheet\06\"matrix-adjacency
  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 0 1 0 0 0
5 0 1 1 1 0 1 0 0
6 0 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
```

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





Jawab :

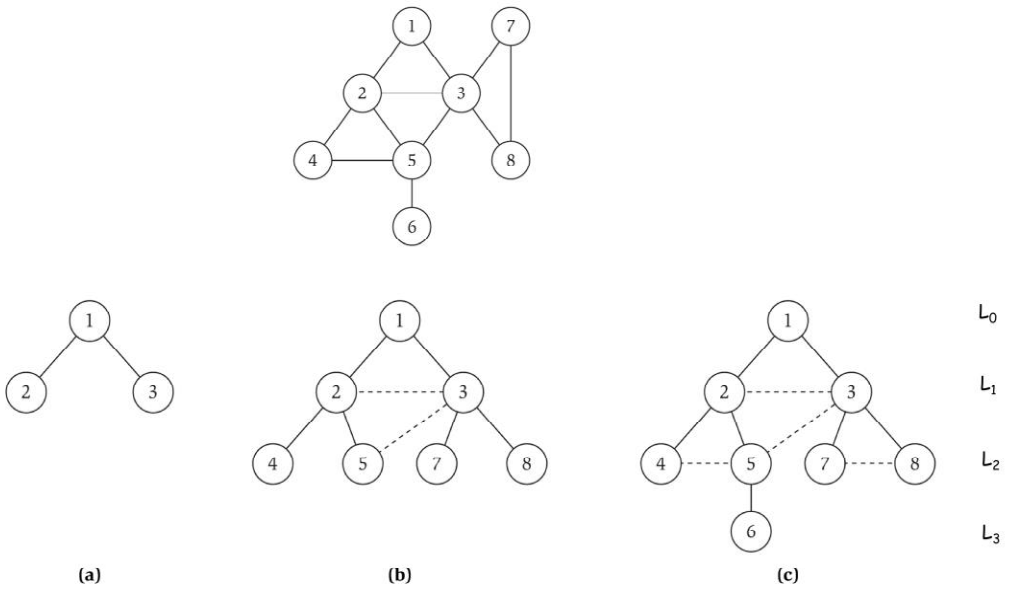
```cpp
#include <bits/stdc++.h>
using namespace std;

void addEdge(vector<int> adj[], int u, int v)
{
    adj[u - 1].push_back(v);
    adj[v - 1].push_back(u);
}

void printList(vector<int> adj[], int V)
{
    for (int v = 0; v < V; ++v)
    {
        cout << v + 1 << ": head ";
        for (auto x : adj[v])
            cout << "-> " << x;
        printf("\n");
    }
}

int main()
{
    int V = 8;
    vector<int> adj[V];

    addEdge(adj, 1, 2);
    addEdge(adj, 1, 3);

    addEdge(adj, 2, 3);
    addEdge(adj, 2, 4);
    addEdge(adj, 2, 5);
```

```
    addEdge(adj, 3, 5);
    addEdge(adj, 3, 7);
    addEdge(adj, 3, 8);

    addEdge(adj, 4, 5);

    addEdge(adj, 5, 6);

    addEdge(adj, 7, 8);

    printList(adj, V);
    return 0;
}
```

Screenshot:

```
\Worksheet\06\"adjacency-list
1: head -> 2-> 3
2: head -> 1-> 3-> 4-> 5
3: head -> 1-> 2-> 5-> 7-> 8
4: head -> 2-> 5
5: head -> 2-> 3-> 4-> 6
6: head -> 5
7: head -> 3-> 8
8: head -> 3-> 7
```

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!



Jawab:

```
#include <iostream>
#include <list>
using namespace std;

class Graph
{
    int V;

    list<int> *adj;
```

```
public:
    Graph(int V);
    void addEdge(int v, int w);
    void BFS(int s);
};

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::BFS(int s)
{
    bool *visited = new bool[V];
    for (int i = 0; i < V; i++)
        visited[i] = false;

    list<int> queue;

    visited[s] = true;
    queue.push_back(s);

    list<int>::iterator i;

    while (!queue.empty())
    {
        s = queue.front();
        cout << s << " ";
        queue.pop_front();

        for (i = adj[s].begin(); i != adj[s].end(); ++i)
        {
            if (!visited[*i])
            {
                visited[*i] = true;
                queue.push_back(*i);
            }
        }
    }
}

int main()
{
    Graph g(10);

    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 << "Breadth First Traversal\n";
    g.BFS(1);
```

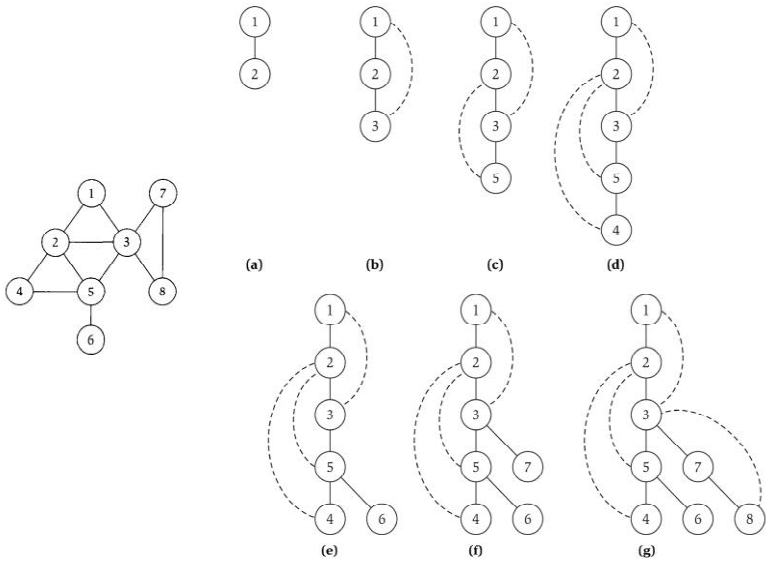
```
    return 0;
}
```

```
BFS
Breadth First Traversal
1 2 3 4 5 7 8 6
```

Kompleksitas asimptotiknya:

Pencarian Breadth-first memiliki waktu berjalan O(V + E) - di mana V adalah jumlah total simpul (atau simpul) yang dikunjungi dan E adalah jumlah total sisi yang dilalui - karena setiap titik dan setiap sisi akan diperiksa paling banyak satu kali. Bergantung pada input ke grafik, O (E) dapat antara O (1) dan O (V^2).

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!



Jawab:

```
#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(10);

    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 Traversal\n";

    g.DFS(1);

    return 0;
}
```

```
DFS
Depth First Traversal
1 2 3 5 6 7 8 4
```

Kompleksitas asimptotiknya:

Pencarian Kedalaman-pertama mengunjungi setiap simpul sekali dan memeriksa setiap tepi dalam grafik satu kali. Oleh karena itu, kompleksitas DFS adalah O(V + E) dengan mengasumsikan bahwa grafik direpresentasikan sebagai daftar adjacency. Di mana V adalah jumlah total simpul (atau simpul) yang dikunjungi dan E adalah jumlah total sisi yang dilalui.