

**TUGAS 5 PRAKTIKUM
DASAR ANALISIS ALGORITMA GRAF**

**MATA KULIAH ANALISIS ALGORITMA
D10G.4205 & D10K.0400601**



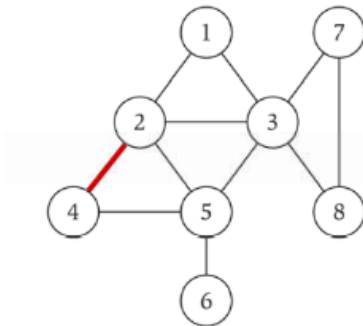
Disusun Oleh :

Rizal Fathur Rahman	(140810170009)
Muhammad Luthfiansyah	(140810170023)
Raihan Luthfiandi Muhammad	(140810170029)
David Ferdinand Immanuel Manurung	(140810170039)
Muhammad Ariq Farhansyah Mutyara	(140810170053)

**PROGRAM STUDI S1 TEKNIK INFORMATIKA FAKULTAS
MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS PADJADJARAN
JATINANGOR
2019**

Nomor 1

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

Jawab :

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

int node[20][20];

void addEdge(int u, int v)
{
    node[u][v] = 1;
    node[v][u] = 1;
}

void displayMatrix(int n)
{
    for (int i = 1; i <=n; i++)
    {
        for(int j = 1; j <= n; j++)
            cout << node[i][j] << " ";
        cout << endl;
    }
}

int main()
{
    int n = 8;
    cout << "Program Undirected Graph for Adjacency Matrix" << endl;
    cout << endl;

    addEdge(1,2);
    addEdge(1,3);
    addEdge(2,4);
    addEdge(2,5);
    addEdge(2,3);
    addEdge(3,5);
    addEdge(3,7);
    addEdge(3,8);
    addEdge(4,5);
    addEdge(5,6);
    addEdge(7,8);
    displayMatrix(n);

    return 0;
}
```

Program Undirected Graph for Adjacency Matrix

```

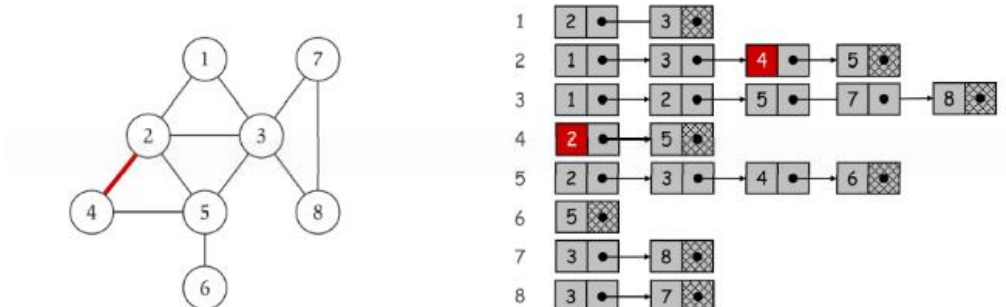
0 1 1 0 0 0 0 0
1 0 1 1 1 0 0 0
1 1 0 0 1 0 1 1
0 1 0 0 1 0 0 0
0 1 1 1 0 1 0 0
0 0 0 0 1 0 0 0
0 0 1 0 0 0 0 1
0 0 1 0 0 0 1 0

```

D:\> _

Nomor 2

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



Jawab :

```

#include <iostream>
#include <vector>
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 displayList(vector<int> adj[],int n)
{
    for (int i = 0; i < n; i++)
    {
        cout << "\n Adjacency list of node "
              << i+1 << "\n head ";
        for (auto x : adj[i])
            cout << "-> " << x;
        printf("\n");
    }
}

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

```

```

        cout << "Program Undirected Graph for Adjacency List" << endl;
        cout << endl;
        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);
        displayList(adj,n);

        return 0;
    }

```

Program Undirected Graph for Adjacency List

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

Adjacency list of node 2
head -> 1-> 3-> 4-> 5

Adjacency list of node 3
head -> 1-> 2-> 5-> 7-> 8

Adjacency list of node 4
head -> 2-> 5

Adjacency list of node 5
head -> 2-> 3-> 4-> 6

Adjacency list of node 6
head -> 5

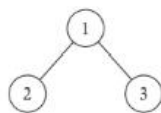
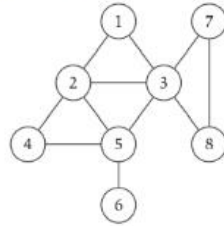
Adjacency list of node 7
head -> 3-> 8

Adjacency list of node 8
head -> 3-> 7

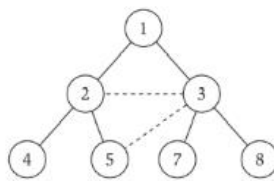
D:\>

Nomor 3

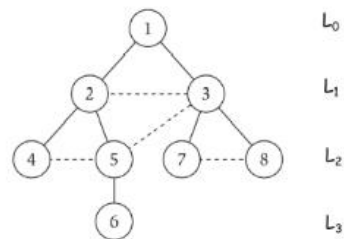
3. Buatlah program Breadth First Search dari algoritma BFS yang telah diberikan. Kemudian uji coba program Anda dengan menginputkan *undirected graph* sehingga menghasilkan tree BFS. Hitung dan berikan secara asimptotik berapa kompleksitas waktunya dalam Big- Θ !



(a)



(b)



(c)

Jawab :

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

class Graph
{
    int N;
    list<int> *adj;
public :
    Graph(int N)
    {
        this->N == N;
        adj = new list<int>[N];
    }
    void addEdge(int u, int v)
    {
        adj[u].push_back(v);
    }
    void BFS(int s)
    {
        bool *visited = new bool[N];
        for(int i = 0; i < N; 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(8);
    g.addEdge(1,2);
    g.addEdge(1,3);
    g.addEdge(2,3);
    g.addEdge(2,4);
    g.addEdge(2,5);
    g.addEdge(3,7);
    g.addEdge(3,8);
    g.addEdge(4,5);
    g.addEdge(5,3);
    g.addEdge(5,6);
    g.addEdge(7,8);

    cout << "BFS Traversal Starts from Node 1" << endl;
    g.BFS(1);

    return 0;
}

```

```

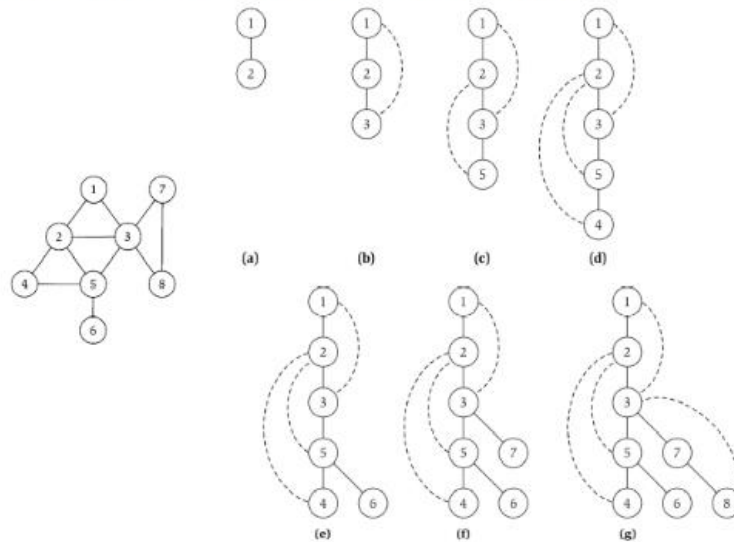
BFS Traversal Starts from Node 1
1 2 3 4 5 7 8
D:\>

```

- 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 $O(|V| + |E|)$.
- Karena Big-O dari BFS adalah $O(V+E)$ dimana V itu jumlah vertex dan E itu adalah jumlah edges maka Big-O = $O(n)$ dimana $n = v + e$
- Maka dari itu Big- Θ nya adalah $\Theta(n)$.

Nomor 4

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

class Graph
{
    int N;

    list<int> *adj;

    void DFSUtil(int u, bool visited[])
    {
        visited[u] = true;
        cout << u << " ";

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

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

    void addEdge(int u, int v)
    {
        adj[u].push_back(v);
    }

    void DFS(int u)
    {
        bool *visited = new bool[N];
        for(int i = 0; i < N; i++)
            visited[i] = false;
    }
};
```

```

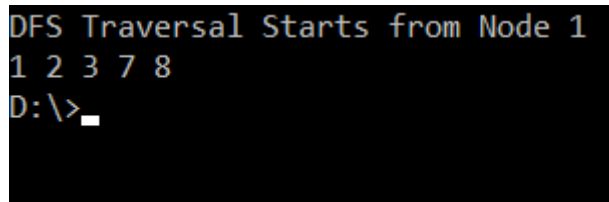
        DFSUtil(u, visited);
    }
};

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

    cout << "DFS Traversal Starts from Node 1" << endl;
    g.DFS(1);

    return 0;
}

```



```

DFS Traversal Starts from Node 1
1 2 3 7 8
D:\>

```

- 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.
- Big O kompleksitas total DFS () adalah $(V + E) \cdot O(n)$ Dengan V = Jumlah Verteks Dan E = Jumlah Edges