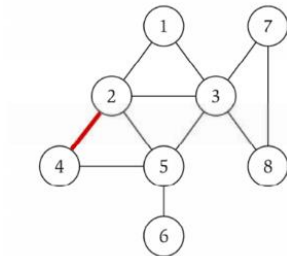


Nama : Tyko Zidane Badhawi

NPM : 140810180031

Kelas A

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

```
/*
Nama      : Tyko Zidane B
NPM       : 140810180031
*/
#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;
        int j;
        for(i=0; i<n; i++){
            for(j=0; j<n; j++){
                cout << adj[i][j]<<" ";
            }
            cout << endl;
        }
    }
};

int main(){
    int nodes;
    int max_edges;
    int origin;
    int destin;

    cout << "\nEnter Number of Nodes\t: "; cin >> nodes;

    AdjacencyMatrix am(nodes);
    max_edges = nodes * (nodes - 1);
    cout<<"\nEnter Edge (example: 1 2)\n(-1 -1 to exit)\n";
    for (int i=0; i<max_edges; i++){
        cout<<"Enter Edge\t: "; cin >> origin >> destin;
        if((origin== -1) && (destin== -1)){
            break;
        }
        am.add_edge(origin, destin);
    }
    cout << endl;
    am.display();
    return 0;
}

```

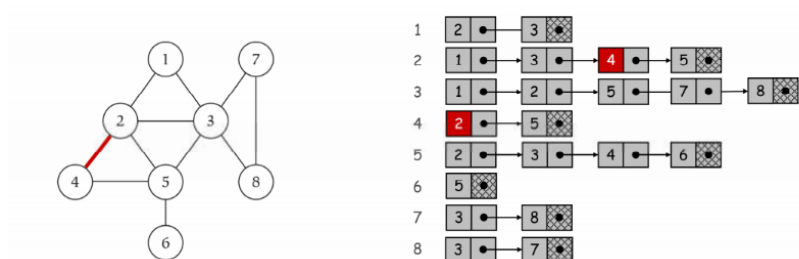
```

C:\Users\user\Documents\Analisis Algoritma\praktikum6
(-1 -1 to exit)
Enter Edge : 1 2
Enter Edge : 1 3
Enter Edge : 2 1
Enter Edge : 2 3
Enter Edge : 2 4
Enter Edge : 2 5
Enter Edge : 3 1
Enter Edge : 3 2
Enter Edge : 3 4
Enter Edge : 3 5
Enter Edge : 3 6
Enter Edge : 3 7
Enter Edge : 4 1
Enter Edge : 4 2
Enter Edge : 4 3
Enter Edge : 4 5
Enter Edge : 4 6
Enter Edge : 5 1
Enter Edge : 5 3
Enter Edge : -1 -1

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

```

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



```

/*
Nama      : Tyko Zidane B
NPM       : 140810180031
*/
#include <iostream>
#include <cstdlib>

using namespace std;

```

```

struct AdjListNode{
    int dest;
    struct AdjListNode* next;
};

struct AdjList{
    struct AdjListNode *head;
};

class Graph{
private:
    int V;
    struct AdjList* array;

public:
    Graph(int V){
        this->V = V;
        array = new AdjList [V];
        for (int i=0; i<V; i++)
            array[i].head = NULL;
    }

    AdjListNode* newAdjListNode(int dest){
        AdjListNode* newNode = new AdjListNode;
        newNode->dest = dest;
        newNode->next = NULL;
        return newNode;
    }

    void addEdge(int src, int dest){
        AdjListNode* newNode = newAdjListNode(dest);
        newNode->next = array[src].head;
        array[src].head = newNode;
        newNode = newAdjListNode(src);
        newNode->next = array[dest].head;
        array[dest].head = newNode;
    }

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

```

```

    }
}

};

int main(){
    Graph gh(8);

    gh.addEdge(1, 2);
    gh.addEdge(1, 3);
    gh.addEdge(2, 4);
    gh.addEdge(2, 5);
    gh.addEdge(2, 3);
    gh.addEdge(3, 7);
    gh.addEdge(3, 8);
    gh.addEdge(4, 5);
    gh.addEdge(5, 3);
    gh.addEdge(5, 6);
    gh.addEdge(7, 8);
    gh.printGraph();

    return 0;
}

```

```

C:\Users\user\Documents\Analisis Algoritma\praktikum66\Soal2.exe

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

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

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

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

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

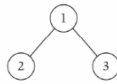
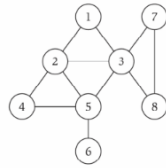
Adjacency list of vertex 6
head -> 5

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

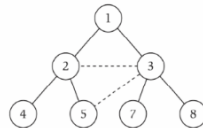
Adjacency list of vertex 8
head -> 7-> 3
-----
Process exited after 8.974 seconds with return value 3221225477
Press any key to continue . . .

```

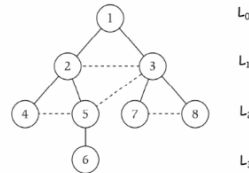
3. Buatlah program Breadth First Search dari algoritma BFS yang telah diberikan. Kemudian uji coba program Anda dengan menginputkan undirected graph sehingga menghasilkan tree BFS. Hitung dan berikan secara asimptotik berapa kompleksitas waktunya dalam Big- Θ !



(a)



(b)



(c)

L_0

L_1

L_2

L_3

```

/*
Nama      : Tyko Zidane B
NPM       : 140810180031
*/
#include<iostream>
using namespace std;

int main(){
    int vertexSize = 8;
    int adjacency[8][8] = {
        {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}
    };
    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<<"BFS : "<<endl;
for(int i = 0; i < vertexSize; i++){
    cout<<output[i]<<" ";
}
}

```

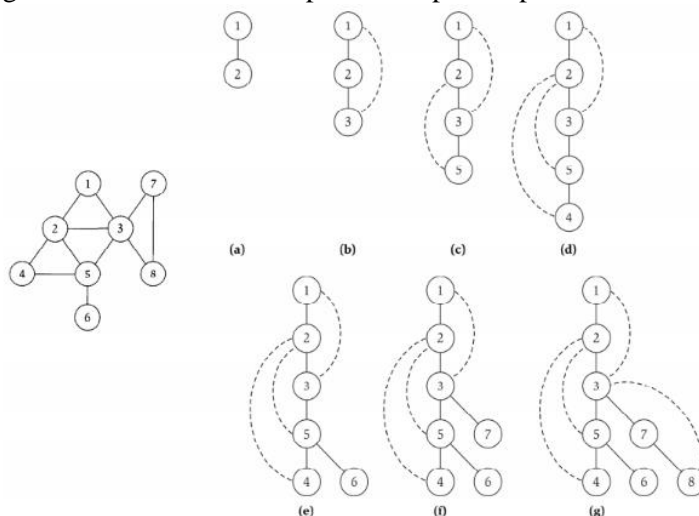
```

12. C:\Users\user\Documents\Analisis Algoritma\praktikum66\Soal3.exe
BFS :
1 2 3 4 5 7 8 6
-----
Process exited after 0.1065 seconds with return value 0
Press any key to continue . . .

```

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 $\Theta(n)$.

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 !



```

/*
Nama      : Tyko Zidane B
NPM       : 140810180031
*/
#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);

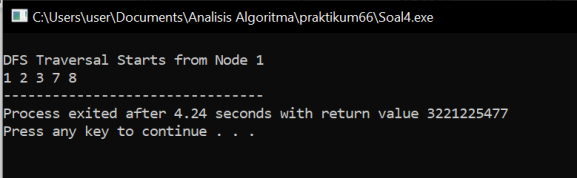
    g.addEdge(1,2);
    g.addEdge(1,3);
    g.addEdge(2,3);
    g.addEdge(2,4);
    g.addEdge(2,5);
    g.addEdge(3,7);
    g.addEdge(3,8);
    g.addEdge(4,5);
    g.addEdge(5,3);
    g.addEdge(5,6);
    g.addEdge(7,8);
}

```



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

return 0;
}
```



DFS merupakan metode pencarian mendalam, yang mengunjungi semua node dari yang ter kiri 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 n , ditambah dengan simpul-simpul saudara kandungnya yang belum dikembangkan.