Alvina Vania Kirana

140810180010

Praktikum Analgo

Worksheet 6

Studi Kasus 1

```
/*
Alvina Vania Kirana
140810180010
Praktikum Analisis Algoritma
Worksheet 6 Soal 1
Adjacency Matriks
*/

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

int matrix[20][20];
int count = 0;

void cetakMatrix(int m);
void add_edge(int x, int y);

int main(int argc, char *argv[]){
    int m=8;

    add_edge(1, 2);
    add_edge(1, 3);
    add_edge(2, 1);
    add_edge(2, 3);
    add_edge(2, 4);
    add_edge(2, 5);
    add_edge(3, 1);
    add_edge(3, 2);
    add_edge(3, 5);
    add_edge(3, 7);
    add_edge(3, 8);
    add_edge(4, 2);
```

```
    add_edge(4, 5);
    add_edge(5, 2);
    add_edge(5, 3);
    add_edge(5, 4);
    add_edge(5, 6);
    add_edge(6, 5);
    add_edge(7, 3);
    add_edge(7, 8);
    add_edge(8, 3);
    add_edge(8, 7);
    cout<<"Maka Adjacency Matrixnya: "<<endl;
    cetakMatrix(m);
}

void cetakMatrix(int m){
    int i, j;
    for (i = 1; i <= m; i++){
        for (j = 1; j<=m; j++)
        {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }
}

void add_edge(int x, int y){
    matrix[x][y] = 1;
    matrix[y][x] = 1;
}
```

Studi Kasus 2

```
/*
Alvina Vania Kirana
140810180010
Praktikum Analisis Algoritma
Worksheet 6 Soal 2
Adjacency List
*/

#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<<"Node "<<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);
    cout<< "Maka Adjacency Listnya:"<<endl;
    gh.printGraph();

    return 0;
}
```

Studi Kasus 3

```
/*
Alvina Vania Kirana
140810180010
Praktikum Analisis Algoritma
Worksheet 6 Soal 3
Breadth First Search
*/

#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(8);
    g.addEdge(1, 2);
    g.addEdge(1, 3);
    g.addEdge(2, 4);
    g.addEdge(2, 5);
    g.addEdge(2, 3);
    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 << "Maka Breadth First Traversalnya ";
    cout << "(mulai dari vertex 1) \n:";
    g.BFS(1);

    return 0;
}
```

Karena dalam worst case BFS harus mempertimbangkan semua jalur (path) untuk semua node yang mungkin, maka nilai kompleksitas waktu dari BFS adalah $O(b^d)$. Kompleksitas waktu juga bisa didefinisikan sebagai O(| E | + | V |) karena setiap vertex dan ujung (edge) akan dijelajahi dalam worst case.

| E | + | V |=n

Maka Big-Θ nya adalah Θ(n).

Studi Kasus 4

```
/*
Alvina Vania Kirana
140810180010
Praktikum Analisis Algoritma
Worksheet 6 Soal 4
Depth First Search
*/
```

```
#include<iostream>
#include<list>
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);
    g.addEdge(1, 2);
    g.addEdge(1, 3);
    g.addEdge(2, 4);
    g.addEdge(2, 5);
    g.addEdge(2, 3);
    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 << "Maka Depth First Traversal";
    cout << " (mulai dari vertex 1) \n:";
    g.DFS(1);

    return 0;
}
```

Kompleksitas ruang algoritma DFS adalah O(bm), karena kita hanya hanya perlu menyimpan satu buah lintasan tunggal dari akar sampai daun, ditambah dengan simpul-simpul saudara kandungnya yang belum dikembangkan