LAPORAN PRAKTIKUM

ANALISIS ALGORITMA

Disusun Oleh:

Delanika Olympiani T. C. 140810180026

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS PADJADJARAN

2020

Nama : Delanika Olympiani T. C.

NPM : 140810180026

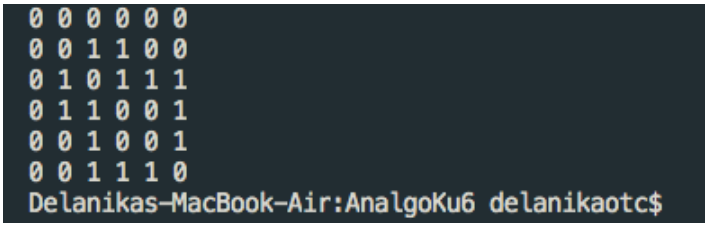
Tugas 6

1. Adjacency Matrix
    a. Source code

```
/*
Nama    : Delanika Olympiani
NPM     : 140810180026
Kelas   : B
Program : Merepresentasikan graph dengan matrix (Adjacency Matrix)
*/

#include <iostream>

using namespace std;

int vertArr[20][20];
int count = 0;
void displayMatrix(int v) {
   int i, j;
   for(i = 0; i < v; i++) {
      for(j = 0; j < v; j++) {
         cout << vertArr[i][j] << " ";
      }
      cout << endl;
   }
}
void add_edge(int u, int v) {
   vertArr[u][v] = 1;
   vertArr[v][u] = 1;
}
int main(int argc, char* argv[]) {
   int v = 6;
   add_edge(1, 2);
   add_edge(1, 3);
   add_edge(2, 3);
   add_edge(2, 4);
```

```
  add_edge(2, 5);
  add_edge(3, 5);
  add_edge(3, 7);
  add_edge(3, 8);
  add_edge(4, 5);
  add_edge(5, 6);
  displayMatrix(v);
}
```

b. Screenshot



2. Adjacency List

a. Source code

```
/*
Nama    : Delanika Olympiani
NPM     : 140810180026
Kelas   : B
Program : Merepresentasikan graph dengan list (Adjacency List)
*/

#include<iostream>
#include<list>
#include<iterator>

using namespace std;

void displayAdjList(list<int> adj_list[], int v) {
  for(int i = 0; i<v; i++) {
     cout << i << "--->";
     list<int> :: iterator it;
```

```
    for(it = adj_list[i].begin(); it != adj_list[i].end(); ++it) {
      cout << *it << " ";
    }
    cout << endl;
  }
}
void add_edge(list<int> adj_list[], int u, int v) {
  adj_list[u].push_back(v);
  adj_list[v].push_back(u);
}
int main(int argc, char* argv[]) {
  int v = 6;

  list<int> adj_list[v];
  add_edge(adj_list, 0, 4);
  add_edge(adj_list, 0, 3);
  add_edge(adj_list, 1, 2);
  add_edge(adj_list, 1, 4);
  add_edge(adj_list, 1, 5);
  add_edge(adj_list, 2, 3);
  add_edge(adj_list, 2, 5);
  add_edge(adj_list, 5, 3);
  add_edge(adj_list, 5, 4);
  displayAdjList(adj_list, v);
}
```

b. Screenshot

```
0--->4 3
1--->2 4 5
2--->1 3 5
3--->0 2 5
4--->0 1 5
5--->1 2 3 4
Delanikas-MacBook-Air:AnalgoKu6 delanikaotc$
```

3. BFS

a. Source code

```
/*
Nama    : Delanika Olympiani
NPM     : 140810180026
Kelas   : B
```

```
Program : Program BFS
*/

#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(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    cout << "Following is Breadth First Traversal "
         << "(starting from vertex 2) \n";
    g.BFS(2);

    return 0;
}
```

b. Screenshot

```
Following is Breadth First Traversal (starting from vertex 2)
2 0 3 1
Delanikas-MacBook-Air:AnalgoKu6 delanikaotc$
```

c. Time Complexity

T(n) = O(V+E)

4. DFS

a. Source code

```
/*
Nama    : Delanika Olympiani
NPM     : 140810180026
Kelas   : B
Program : Program DFS
*/

#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(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);
```

```
    cout << "Following is Depth First Traversal"
         " (starting from vertex 2) \n";
    g.DFS(2);

    return 0;
}
```

b. Screenshot

```
Following is Depth First Traversal (starting from vertex 2)
2 0 1 3
Delanikas-MacBook-Air:AnalgoKu6 delanikaotc$
```

c. Time Complexity

T(n) = O(V+E)