TUGAS MODUL PRAKTIKUM 6

Disusun oleh :

Anne Audistya Fernanda

140810180059

Kelas A

PROGRAM STUDI S1 TEKNIK INFORMATIKA

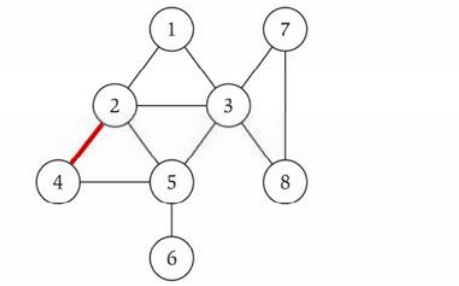
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS PADJADJARAN

2020

Tugas

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

a. Sourcecode

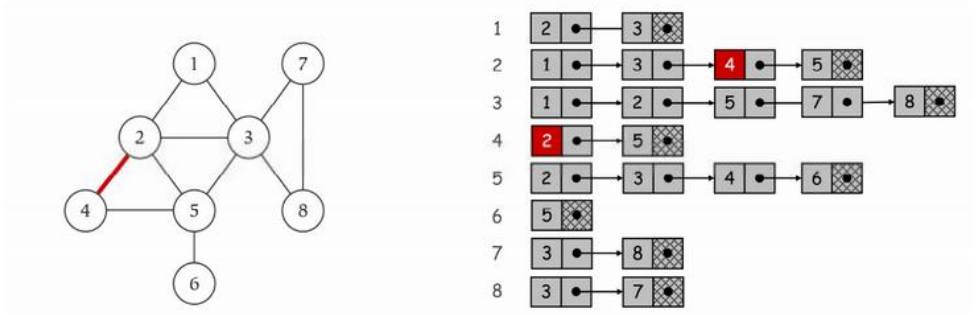
```
/*
Nama : Anne Audistya Fernanda
NPM  : 140810180059
Kelas: A
Deskripsi : Program Adjacency Matrix
*/


#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;
                }
            }
        }

        // Menambahkan edge ke graf
```

```
        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;
            }
        }

        // Mencetak graf

        void display()
        {
            int i,j;
            for(i = 0;i < n;i++)
            {
                for(j = 0; j < n; j++)
                    cout<<adj[i][j]<<"  ";
                cout<<endl;
            }
        }
};

// Main

int main()
{
    int nodes, max_edges, origin, destin;
    cout<<"Enter number of nodes: ";
    cin>>nodes;
    AdjacencyMatrix am(nodes);
    max_edges = nodes * (nodes - 1);
    for (int i = 0; i < max_edges; i++)
    {
        cout<<"Enter edge (-1 -1 to exit): ";
        cin>>origin>>destin;
        if((origin == -1) && (destin == -1))
            break;
        am.add_edge(origin, destin);
    }
    am.display();
    return 0;
}
```

b. Screenshot

```
C:\Users\ASUS\Documents\Analgo\AnalgoKu\AnalgoKu6\AdjacencyMatrix.exe
Enter number of nodes: 8
Enter edge (-1 -1 to exit): 1 2
Enter edge (-1 -1 to exit): 2 1
Enter edge (-1 -1 to exit): 1 3
Enter edge (-1 -1 to exit): 3 1
Enter edge (-1 -1 to exit): 2 3
Enter edge (-1 -1 to exit): 3 2
Enter edge (-1 -1 to exit): 2 4
Enter edge (-1 -1 to exit): 4 2
Enter edge (-1 -1 to exit): 2 5
Enter edge (-1 -1 to exit): 5 2
Enter edge (-1 -1 to exit): 3 5
Enter edge (-1 -1 to exit): 5 3
Enter edge (-1 -1 to exit): 4 5
Enter edge (-1 -1 to exit): 5 4
Enter edge (-1 -1 to exit): 3 7
Enter edge (-1 -1 to exit): 7 3
Enter edge (-1 -1 to exit): 3 8
Enter edge (-1 -1 to exit): 8 3
Enter edge (-1 -1 to exit): 5 6
Enter edge (-1 -1 to exit): 6 5
Enter edge (-1 -1 to exit): 7 8
Enter edge (-1 -1 to exit): 8 7
Enter edge (-1 -1 to exit): -1 -1
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

--------------------------------
Process exited after 125.3 seconds with return value 0
Press any key to continue . . .
```

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



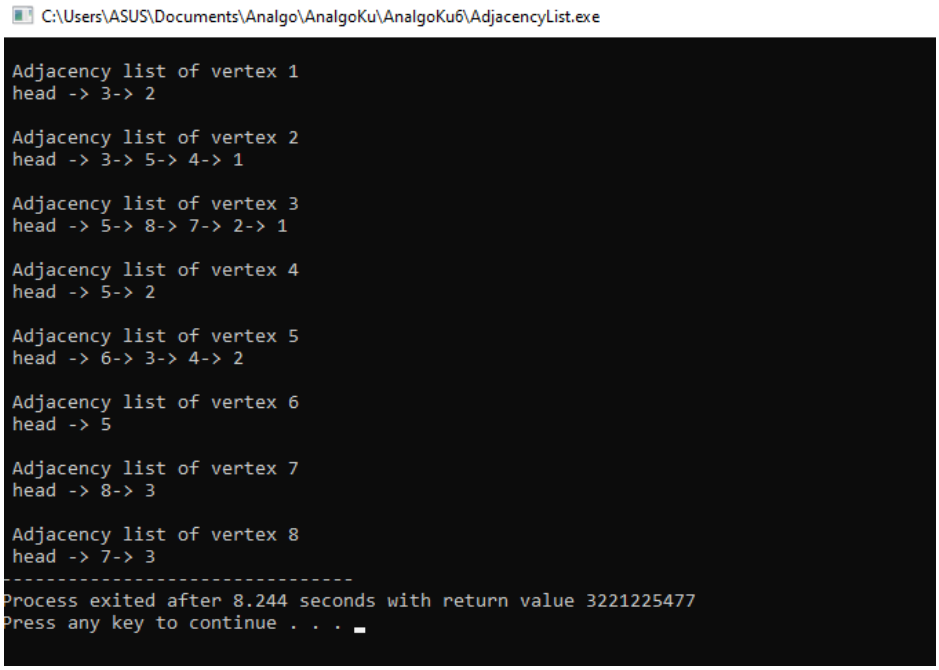
a. Sourcecode

```
/*
Nama : Anne Audistya Fernanda
NPM  : 140810180059
Kelas: A
Deskripsi : Program Adjacency List
*/

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

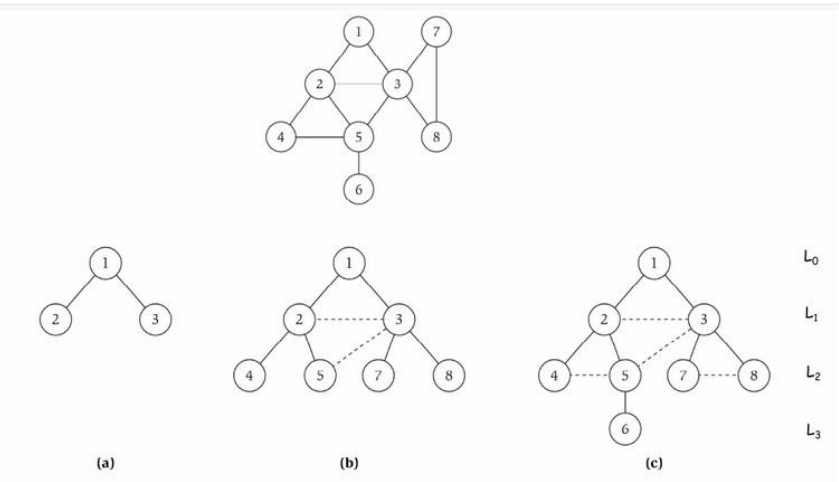
```
struct AdjListNode
{
    int dest;
    struct AdjListNode* next;
};


// Adjacency List
struct AdjList
{
    struct AdjListNode *head;
};


// Class Graph

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;
        }

        // Creating New Adjacency List Node

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

        // Adding Edge to Graph

        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;
        }
```

```
        //Print the graph

        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;
            }
        }
};

// Main

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);
    // print the adjacency list representation of the above graph
    gh.printGraph();

    return 0;
}
```

b. Screenshot



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!



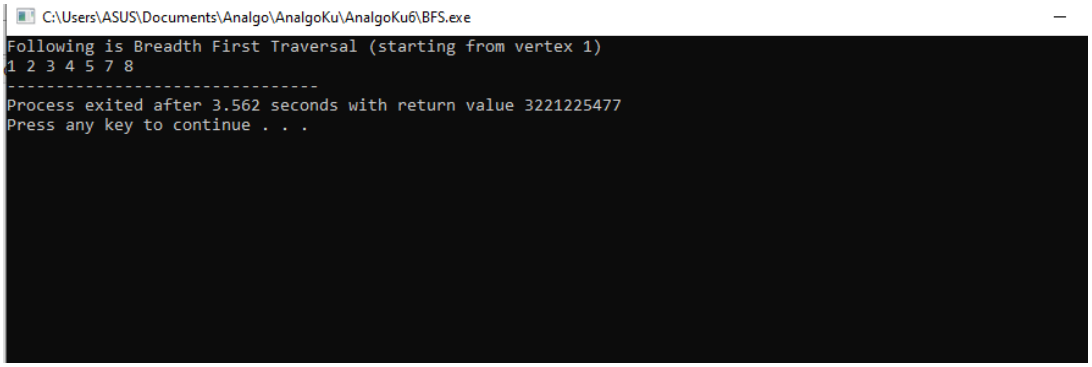
a. Sourcecode

```
/*
Nama : Anne Audistya Fernanda
NPM  : 140810180059
Kelas: A
Deskripsi : Program Breadth First Search
*/
```

```
// Program to print BFS traversal from a given
// source vertex. BFS(int s) traverses vertices
// reachable from s.
#include<iostream>
#include <list>

using namespace std;

// This class represents a directed graph using
// adjacency list representation
class Graph
{
   int V; // No. of vertices

   // Pointer to an array containing adjacency
   // lists
   list<int> *adj;
public:
   Graph(int V); // Constructor

   // function to add an edge to graph
   void addEdge(int v, int w);

   // prints BFS traversal from a given source s
   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); // Add w to v's list.
}

void Graph::BFS(int s)
{
   // Mark all the vertices as not visited
   bool *visited = new bool[V];
   for(int i = 0; i < V; i++)
         visited[i] = false;

   // Create a queue for BFS
   list<int> queue;

   // Mark the current node as visited and enqueue it
   visited[s] = true;
   queue.push_back(s);

   // 'i' will be used to get all adjacent
```

```
    // vertices of a vertex
    list<int>::iterator i;

    while(!queue.empty())
    {
          // Dequeue a vertex from queue and print it
          s = queue.front();
          cout << s << " ";
          queue.pop_front();

          // Get all adjacent vertices of the dequeued
          // vertex s. If a adjacent has not been visited,
          // then mark it visited and enqueue it
          for (i = adj[s].begin(); i != adj[s].end(); ++i)
          {
                if (!visited[*i])
                {
                      visited[*i] = true;
                      queue.push_back(*i);
                }
          }
    }
}

// Driver program to test methods of graph class
int main()
{
    // Create a graph given in the above diagram
    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 << "Following is Breadth First Traversal "
          << "(starting from vertex 1) \n";
    g.BFS(1);

    return 0;
}
```

b. Screenshot



c. Kompleksitas Waktu
Karena Big-O dari BFS adalah O(|V| + |E|) dimana V itu jumlah vector dan E itu adalah jumlah edges maka Big-O = O(n) dimana n = v+e

Maka dari itu Big-Θ nya adalah Θ(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!

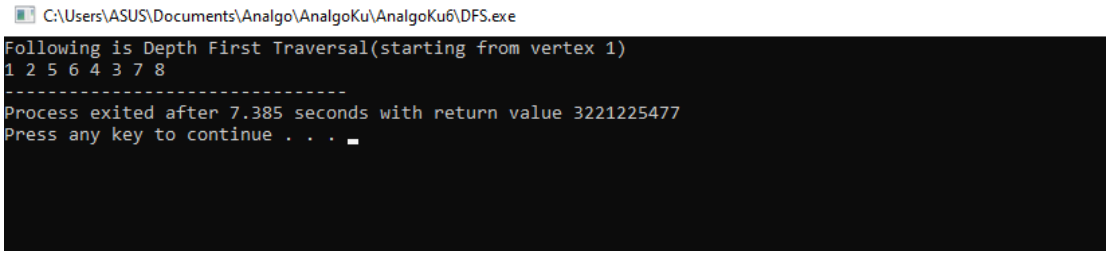
a. Sourcecode

```
/*
Nama : Anne Audistya Fernanda
NPM  : 140810180059
Kelas: A
Deskripsi : Program Depth First Search
*/

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

// Graph class merepresentasikan graf berarah menggunakan
representasi adjacency list
class Graph
{
   int V; // No. simpul

   // Pointer ke array yang memiliki adjacency lists
   list<int> *adj;

   // Fungsi rekursif yang digunakan DFS
   void DFSUtil(int v, bool visited[]);
public:
   Graph(int V); // Constructor

   // fungsi untuk menambah tepian ke graf
```

```
   void addEdge(int v, int w);

   // DFS traversal dari simpul yang terjangkau dari v
   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); // Menambah w ke list v.
}

void Graph::DFSUtil(int v, bool visited[])
{
   // Menandakan node bersangkutan sudah dikunjungi lalu cetak
   visited[v] = true;
   cout << v << " ";

   // Ulang simpul berdekatan ke node ini
   list<int>::iterator i;
   for (i = adj[v].begin(); i != adj[v].end(); ++i)
         if (!visited[*i])
               DFSUtil(*i, visited);
}

// DFS traversal dari simpul terjangkau dari v.
// Menggunakan rekursif DFSUtil()
void Graph::DFS(int v)
{
   // Menandakan semua simpul belum dikunjungi
   bool *visited = new bool[V];
   for (int i = 0; i < V; i++)
         visited[i] = false;

   // Memanggil fungsi rekursif pembantu untuk mencetak DFS
traversal
   DFSUtil(v, visited);
}

int main()
{
   // Membuat graf di diagram
   Graph g(8);
   g.addEdge(1, 2);
   g.addEdge(1, 3);
   g.addEdge(2, 5);
   g.addEdge(2, 4);
   g.addEdge(5, 6);
```

```
    g.addEdge(3, 7);
    g.addEdge(3, 8);
    g.addEdge(7, 8);

    cout << "Following is Depth First Traversal"
                "(starting from vertex 1) \n";
    g.DFS(1);

    return 0;
}
```

b. Screenshot



c. Kompleksitas Waktu

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