

Tugas Kelompok Analisis Algoritma



Disusun Oleh :

Junia Adhani Juzar	(140810170015)
Karimah Azzuhdu T.M	(140810170027)
Rafi Candra	(140810170037)
Victor Wijaya	(140810170057)

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

1. Program Adjacency Matrix dari Undirected Graph Program :

```
/*  
  
    * C++ Program to Implement Adjacency Matrix  
  
    */  
  
#include <iostream>  
#include <cstdlib>  
using namespace std;  
#define MAX 20  
  
/*  
  
    * Adjacency Matrix Class  
  
    */  
  
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;  
                    }  
            }  
    }  
  
    /*  
  
        * Adding Edge to Graph  
  
        */  
  
    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;
    }
}

/*
 * Print the graph
 */

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

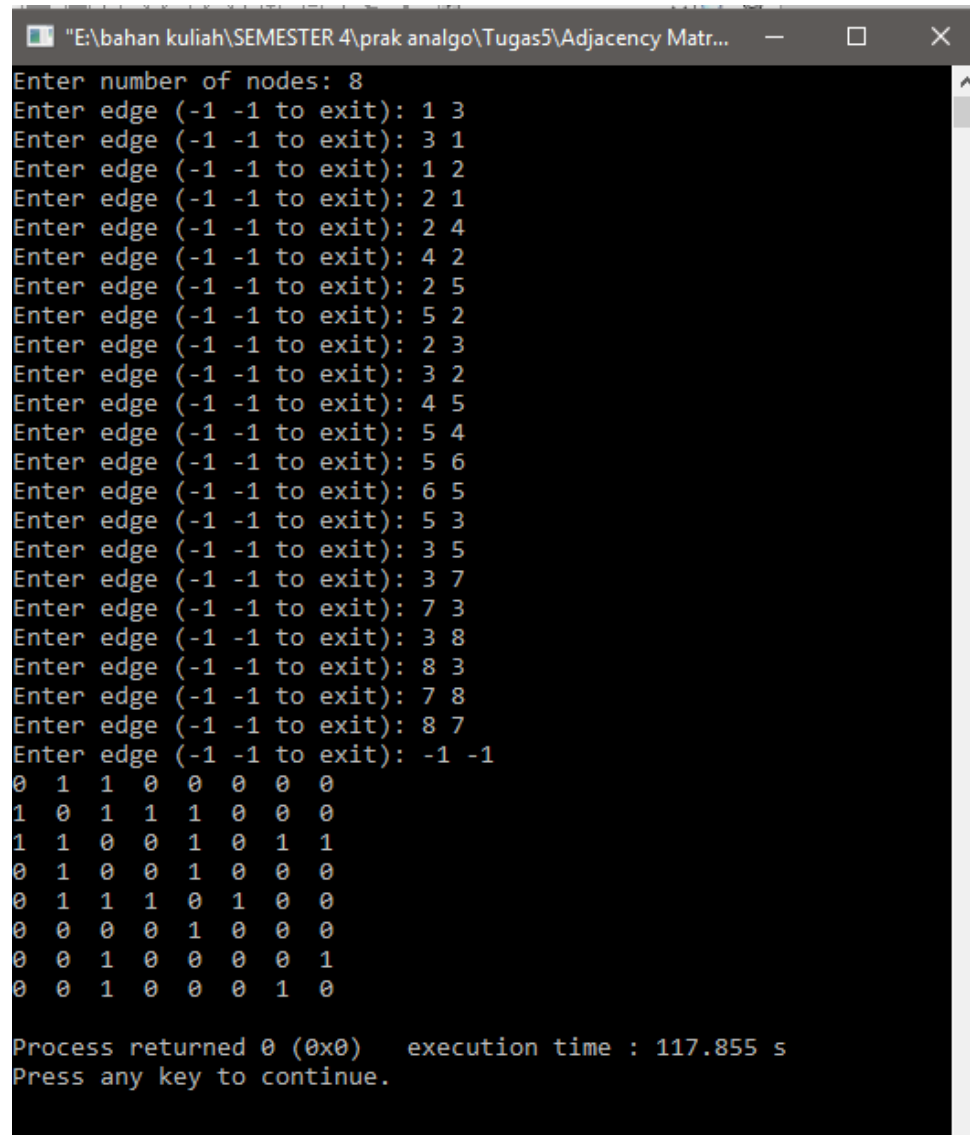
};

/*
 * 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);
    }
}

```

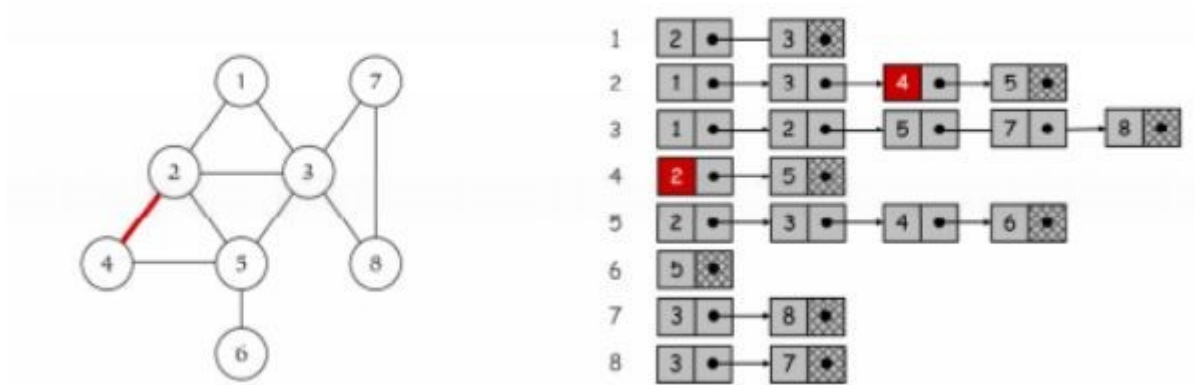
```
    am.display();  
return 0;  
}
```

Output:



```
"E:\bahan kuliah\SEMESTER 4\prak analgo\Tugas5\Adjacency Matr...  
Enter number of nodes: 8  
Enter edge (-1 -1 to exit): 1 3  
Enter edge (-1 -1 to exit): 3 1  
Enter edge (-1 -1 to exit): 1 2  
Enter edge (-1 -1 to exit): 2 1  
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): 2 3  
Enter edge (-1 -1 to exit): 3 2  
Enter edge (-1 -1 to exit): 4 5  
Enter edge (-1 -1 to exit): 5 4  
Enter edge (-1 -1 to exit): 5 6  
Enter edge (-1 -1 to exit): 6 5  
Enter edge (-1 -1 to exit): 5 3  
Enter edge (-1 -1 to exit): 3 5  
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): 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 returned 0 (0x0)   execution time : 117.855 s  
Press any key to continue.
```

2. Adjacency List dari Undirected Graph



Program :

```
/*  
 * C++ Program to Implement Adjacency List  
 */  
  
#include  
<iostream>  
#include  
<cstdlib>  
using  
namespace std;  
  
/*  
 * Adjacency List Node  
 */  
struct AdjListNode  
{  
    i  
    n  
    t  
    d  
    e  
    s  
    t  
    ;  
    struct AdjListNode* next;  
};
```

```

/*
* Adjacency List
*/
struct
AdjL
ist
{
    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

```

```

*
/
i
n
t
m
a
i
n
(
)
{
    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;
}

```

Output :


```

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

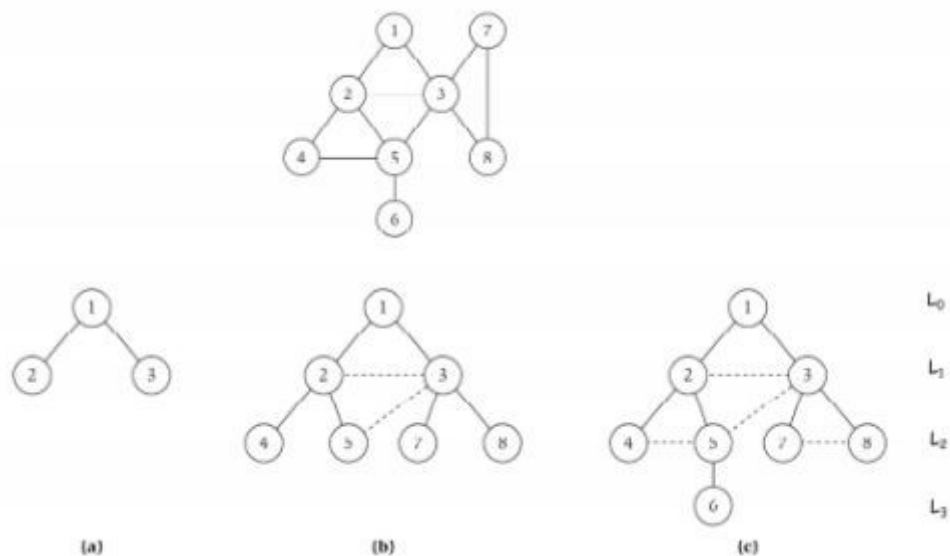
Adjacency list of vertex 6
head -> 5

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

Adjacency list of vertex 8
head -> 7-> 3
-----
Process exited after 5.362 seconds with return value 3221225477
Press any key to continue . . .

```

3. . Program BFS dari Undirected Graph



Program :

```

// 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
        (!vis
        ited[
        *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);

```

```

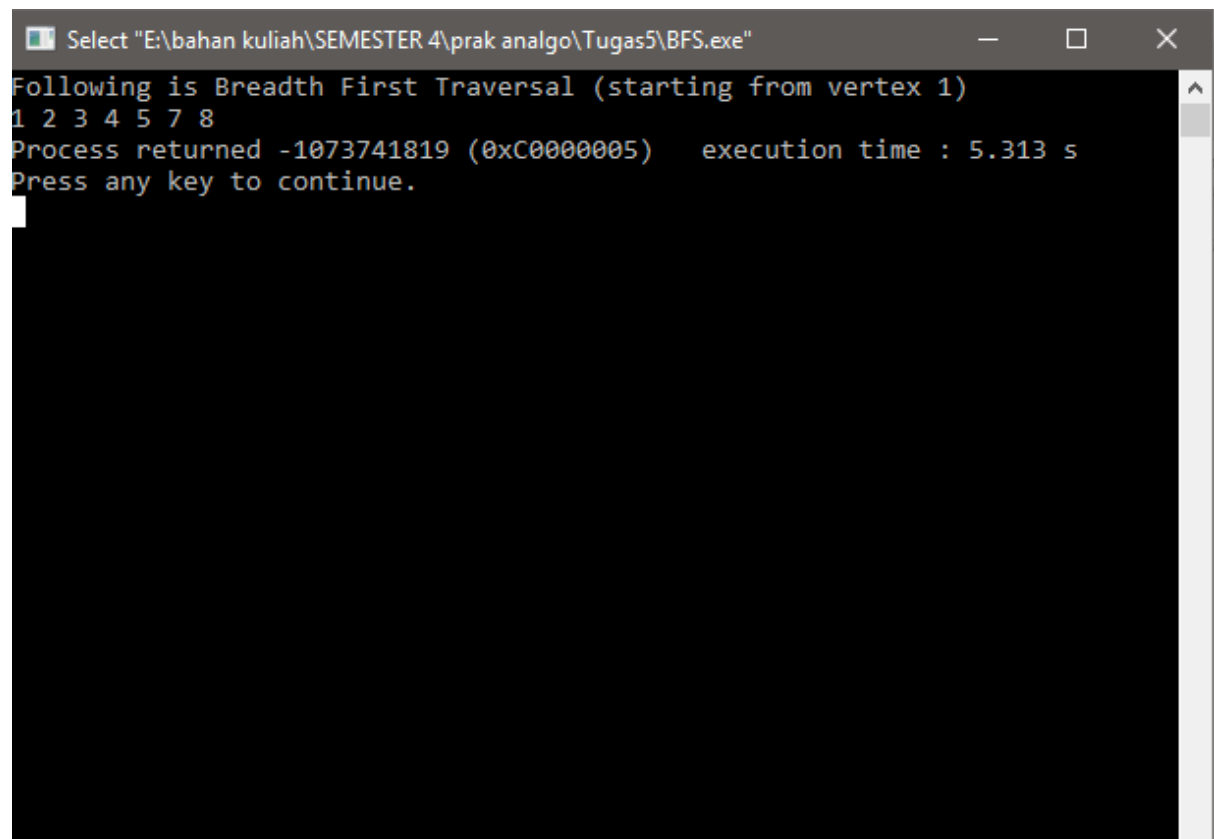
        g.addEdge(1, 2);
        g.addEdge(1, 3);
    g.addEdge(2, 4);
        g.addEdge(2, 5);
        g.addEdge(2, 3);
        g.addEdge(3, 7);
        g.addEdge(3, 8);
        g.addEdge(4, 5);
        g.addEdge(5, 3);
        g.addEdge(5, 6);
        g.addEdge(7, 8);

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

    return 0;
}

```

Output :

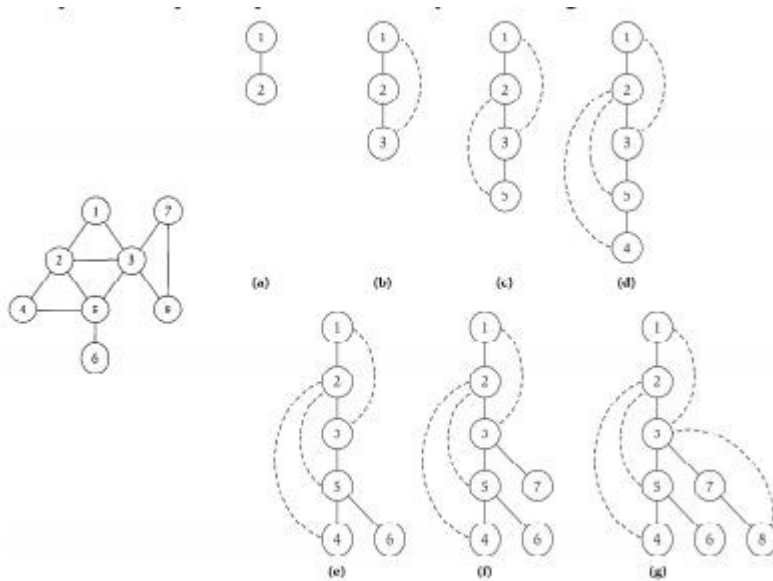


```

Select "E:\bahan kuliah\SEMESTER 4\prak analgo\tugas5\BFS.exe"
Following is Breadth First Traversal (starting from vertex 1)
1 2 3 4 5 7 8
Process returned -1073741819 (0xC0000005) execution time : 5.313 s
Press any key to continue.

```

4. Program DFS undirected Graph



Program

```
// C++ program to print DFS traversal from
// a given vertex in a given graph
#include<iostream>
using namespace std;

// Graph 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;

// A recursive function used by
// DFS void DFSUtil(int v, bool
// visited[]); public:
// Graph(int V); // Constructor
```

```

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

        // DFS traversal of the
vertices      // reachable
from 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); // Add w to
v's list.
} void Graph::DFSUtil(int v, bool
visited[])
{
    // Mark the current node as visited and
    // print it
visited[v] = true;
cout << v << " ";

    // Recur for all the vertices adjacent
// to this vertex      list<int>::iterator i;
for (i = adj[v].begin(); i != adj[v].end();
++i)
    if (!visited[*i])
DFSUtil(*i, visited);
}

// DFS traversal of the vertices reachable
from v.
// It uses recursive
DFSUtil() void
Graph::DFS(int v)
{
    // Mark all the vertices as not
visited      bool *visited = new

```

```

bool[V];    for (int i = 0; i < V;
i++)        visited[i] = false;

    // Call the recursive helper function
    // to print DFS traversal
    DFSUtil(v, visited);
}
in
t
ma
in
()
{
    // 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 Depth First
Traversal"          " (starting from
vertex 1) \n";
        g.DFS(1);

    retur
n 0;
}

```

"E:\bahan kuliah\SEMESTER 4\prak analgo\Tugas5\DFS.exe"

Following is Depth First Traversal (starting from vertex 1)

1 2 4 5 3 7 8

Process returned -1073741819 (0xC0000005) execution time : 3.116 s

Press any key to continue.