**LAPORAN PRAKTIKUM**

**ANALISIS ALGORITMA**

**­**

**DISUSUN OLEH**

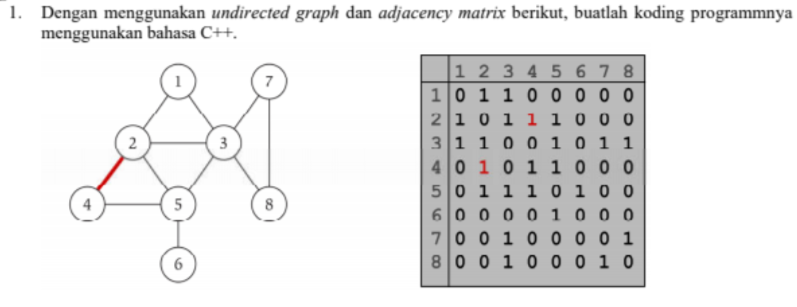
SALMA ALIFIA SHAFIRA 140810180058

**UNIVERSITAS PADJADJARAN**

**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM**

**TEKNIK INFORMATIKA**

**2020**



**SOURCECODE**

/\*

Nama : Salma Alifia Shafira

NPM : 140810180058

Kelas : B

Program : Undirected and 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;

}

}

}

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;

}

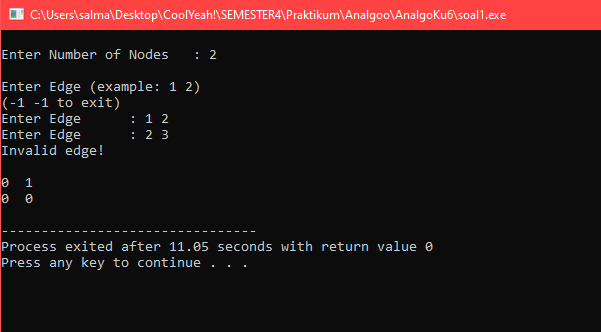
am.add\_edge(origin, destin);

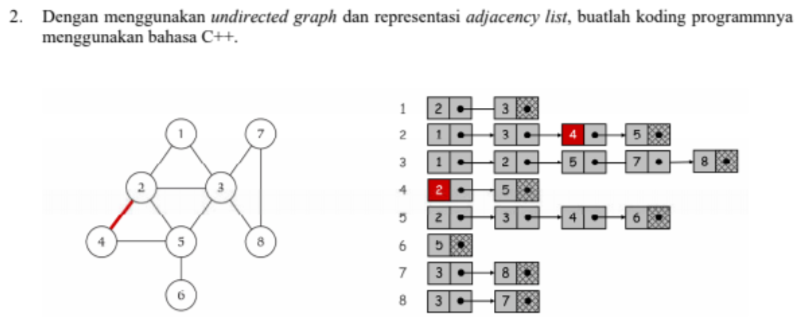
}

cout << endl;

am.display();

return 0;}





**SOURCECODE**

/\*

Nama : Salma Alifia Shafira

NPM : 140810180058

Kelas : B

Program : Undirected graph and 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 << "\n Adjacency list of vertex " << v << "\n head ";

while (pCrawl){

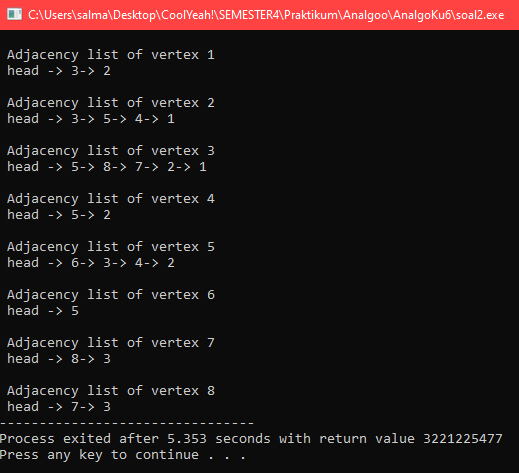
cout << "-> " << pCrawl->dest;

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);

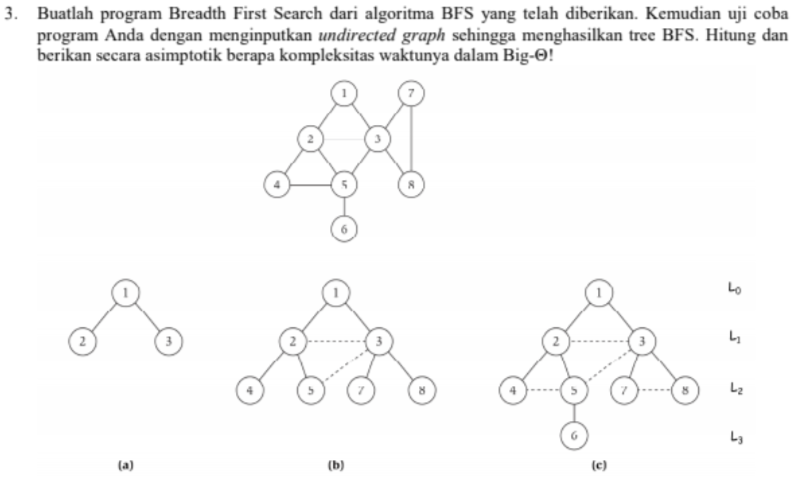
gh.addEdge(5, 6);

gh.addEdge(7, 8);

gh.printGraph();

return 0;

}



**SOURCECODE**

/\*

Nama : Salma Alifia Shafira

NPM : 140810180058

Kelas : B

Program : Breadth First Search

\*/

#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++;

}

}

}

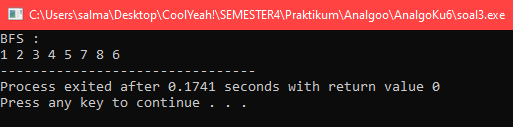
cout<<"BFS : "<<endl;

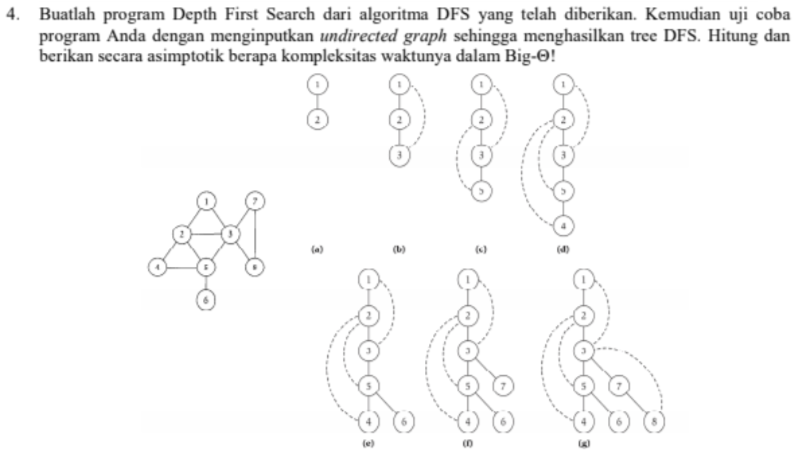
for(int i = 0; i < vertexSize; i++){

cout<<output[i]<<" ";

}

}





**SOURCECODE**

/\*

Nama : Salma Alifia Shafira

NPM : 140810180058

Kelas : B

Program : Depth First Search

\*/

#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;

}

