**TUGAS MODUL PRAKTIKUM 6**

**ANALISIS ALGORITMA**



**Muhammad Fahmi Alwan 140810170052**

**PROGRAM STUDI S1 TEKNIK INFORMATIKA**

**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM**

**UNIVERSITAS PADJADJARAN**

**TAHUN AJARAN 2018/2019**

**No 1.**

#include<iostream>

using namespace std;

int vertArr[20][20]; // Inisialisasi Array 20 x 20 untuk adjMatrix

void displayMatrix(int v) { // fungsi display matrix

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) { // fungsi menambah edge

vertArr[u][v] = 1;

vertArr[v][u] = 1;

}

main(int argc, char\* argv[]) {

int v = 6; // ada 6 nodes

add\_edge(0, 4);

add\_edge(0, 3);

add\_edge(1, 2);

add\_edge(1, 4);

add\_edge(1, 5);

add\_edge(2, 3);

add\_edge(2, 5);

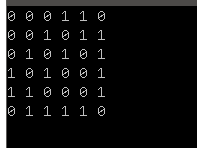
add\_edge(5, 3);

add\_edge(5, 4);

displayMatrix(v);

}

Screenshot :



**No 2.**

#include<bits/stdc++.h>

using namespace std;

// fungsi untuk menambah edge

void addEdge(vector<int> adj[], int u, int v)

{

adj[u].push\_back(v);

adj[v].push\_back(u);

}

// funsi untuk menampilkan adjecancy list

void printGraph(vector<int> adj[], int V)

{

for (int v = 0; v < V; ++v)

{

cout << "\n Adjacency list of vertex "

<< v << "\n head ";

for (auto x : adj[v])

cout << "-> " << x;

printf("\n");

}

}

int main()

{

int V = 5;

int v,u;

int i=0;

bool lagi;

vector<int> adj[V];

cout << "Masukan vertex beserta pasanganya!";

while(true){

cout << "\ninput";

cout << "\nVertex = "; cin >> v;

cout << "Pasangan = "; cin >> u;

addEdge (adj, u, v);

cout << "\n";

cout << "lagi ?"; cin >> lagi;

if (!lagi)

break;

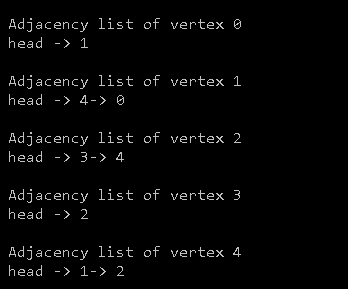
}

printGraph(adj, V);

return 0;

}

Screenshot :



**No 3.**

#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]<<" ";

}

}

Screenshot :



**No 4.**

#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()

{

int node,start;

cout<<"Input the amount of your nodes : ";cin>>node;

Graph g(node);

cout<<"Instructions :"<<endl;

cout<<"1. Enter the number of nodes from 0 to n-1"<<endl;

cout<<"2. Enter negative numbers (such as -1) on either node input to to exit the program"<<endl;

for(;;){

int node1,node2;

cout<<"Enter number between "<<0<<" to "<<node-1<<endl;

cout<<"Input node 1 : ";cin>>node1;

cout<<"Input node 2 : ";cin>>node2;

if(node1>=0&&node2>=0&&node1<node&&node2<node){

g.addEdge(node1,node2);

cout<<endl;

}

else if(node1<0||node2<0)

break;

else

cout<<"Wrong input. Please enter again"<<endl;

}

back:

cout<<"\nNode starts from : ";cin>>start;

if(start<0||start>node-1){

cout<<"Wrong input. Please enter again"<<endl;

goto back;

}

cout<<"Your Depth First Traversal (starting from vertex "<<start<<")"<<endl;

g.DFS(start);

return 0;

}

Screenshot:



Analisis :

* BFS merupakan metode pencarian yang bergeraknya melebar maksudnya mengunjungi node dari kiri ke kanan di level yang sama. Ketika seluruh node pada suatu level sudah dikunjungi semua, maka akan berpindah ke level selanjutnya. Dalam worst case BFS, algoritma harus mempertimbangkan semua jalur (path) untuk semua node yang mungkin, maka nilai kompleksitas waktunya adalah O( |V| + |E| ).
* DFS merupakan metode pencarian yang bergeraknya mendalam, yang mengunjungi semua node dari yang terkiri lalu geser ke kanan hingga semua node dikunjungi. 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.