**TUGAS PRAKTIKUM**

**ANALISIS ALGORITMA**



**Disusun Oleh:**

Diemas Aksya Fachriza (140810170013)

Fadhli Hibatul Haqqi (140810170019)

Muhammad Afif (140810170045)

Kevin Akbar Adhiguna (140810170055)

**PROGRAM STUDI TEKNIK INFORMATIKA**

**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM**

**UNIVERSITAS PADJADJARAN**

**2018/2019**

No 1.

#include<iostream>

using namespace std;

int main(){

int matriks[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},

{0,1,0,1,1,0,0,0},

{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}

};

for(int i=0;i<8;i++){

for(int j=0; j<8;j++){

cout<<matriks[i][j]<<" ";

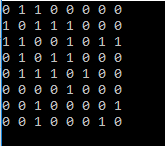
}

cout<<endl;

}

}

Screenshot :



No 2.

#include<iostream>

#include<windows.h>

using namespace std;

struct adjacent{

int nodeAdj;

adjacent\* nextAdj;

};

struct elemen{

int node;

elemen\* next;

adjacent\* firstAdj;

};

typedef elemen\* pointerNode;

typedef adjacent\* pointerAdj;

typedef pointerNode list;

void createListNode(list& first){

first = NULL;

}

void createNode(pointerNode& pBaru,int vertex){

pBaru = new elemen;

pBaru->node = vertex;

pBaru->next = NULL;

pBaru->firstAdj = NULL;

}

void createAdjacent(pointerAdj& pBaru,int vertex){

pBaru = new adjacent;

pBaru->nodeAdj = vertex;

pBaru->nextAdj = NULL;

}

void insertAdjacent(pointerNode& curNode,pointerAdj pBaruAdj){

pointerAdj last;

if(curNode->firstAdj == NULL){

curNode->firstAdj = pBaruAdj;

}else{

last = curNode->firstAdj;

while(last->nextAdj != NULL){

last = last->nextAdj;

}

last->nextAdj = pBaruAdj;

}

}

void insertElement(list& first, pointerNode pBaruNode, int size){

pointerNode last;

pointerAdj pBaruAdj;

if(first == NULL){

first = pBaruNode;

}else{

last = first;

while(last->next != NULL){

last = last->next;

}

last->next = pBaruNode;

}

if(size>0){

cout<<"Masukan node yang berhubungan dengan "<<pBaruNode->node<<" : "<<endl;

}

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

int vertex;

cin>>vertex;

createAdjacent(pBaruAdj,vertex);

insertAdjacent(pBaruNode,pBaruAdj);

}

}

void output(list first){

pointerNode pOut;

pointerAdj pOutAdj;

if(first == NULL){

cout<<"Tidak ada Node"<<endl;

}else{

pOut = first;

while(pOut != NULL){

cout<<"Parent = "<<pOut->node<<endl;

if(pOut->firstAdj == NULL){

cout<<"Tidak ada adjacency"<<endl;

}else{

pOutAdj = pOut->firstAdj;

cout<<"Child = ";

while(pOutAdj != NULL){

cout<<pOutAdj->nodeAdj<<" ";

pOutAdj = pOutAdj->nextAdj;

}

}

cout<<endl;

pOut = pOut->next;

}

}

}

int main(){

list first;

pointerNode node;

createListNode(first);

createNode(node,1);

insertElement(first,node,2);

createNode(node,2);

insertElement(first,node,4);

createNode(node,3);

insertElement(first,node,5);

createNode(node,4);

insertElement(first,node,2);

createNode(node,5);

insertElement(first,node,4);

createNode(node,6);

insertElement(first,node,1);

createNode(node,7);

insertElement(first,node,2);

createNode(node,8);

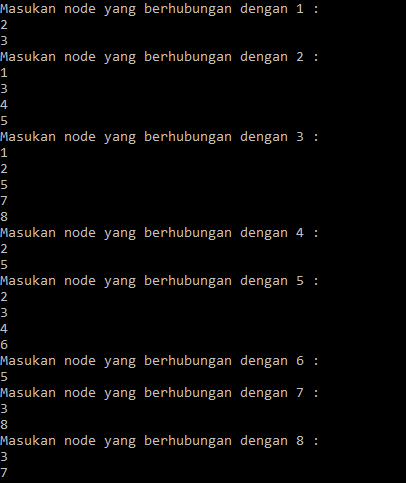
insertElement(first,node,2);

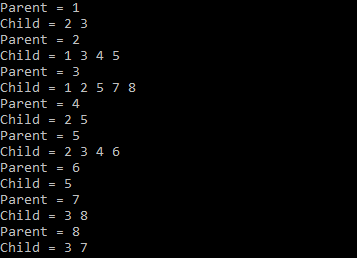
output(first);

system("pause");

}

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;

}