

Tugas Praktikum 6
Analisis Algoritma



Sina Mustopa (140810180017)

S1 Teknik Informatika
Fakultas Matematika dan Ilmu Pengetahuan Alam
Universitas Padjadjaran

No 1.

```
#include<iostream>
```

```
using namespace std;
```

```
int main(){
```

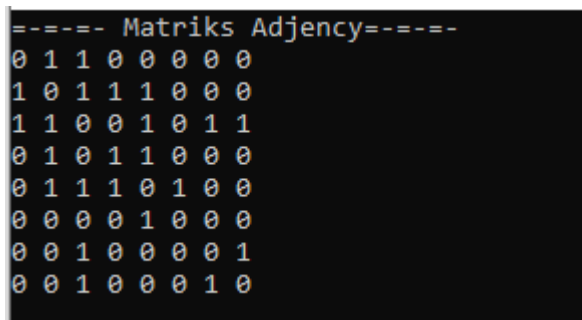
```
    int a[8][8] = {  
        {0,1,1,0,0,0,0,0},  
        {1,0,1,1,1,0,0,0},  
        {1,1,0,0,1,0,1,1},  
        {0,1,0,1,1,0,0,0},  
        {0,1,1,1,0,1,0,0},  
        {0,0,0,0,1,0,0,0},  
        {0,0,1,0,0,0,0,1},  
        {0,0,1,0,0,0,1,0}  
    };
```

```
    cout<<"==--== Matriks Adjency==--=="<<endl;
```

```
    for(int i=0;i<8;i++){  
        for(int j=0; j<8;j++){  
            cout<<a[i][j]<<" ";  
        }  
        cout<<endl;
```

```
}
```

Screenshot :



```
==--== Matriks Adjency==--==  
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
```

No 2.

```
/*
```

Nama : Sina Mustopa

Kelas : A

NPM : 140810180017

```
*/
```

```
#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<<"Induk = "<<pOut->node<<endl;
            if(pOut->firstAdj == NULL){
                cout<<"Tidak ada adjacency"<<endl;
            }else{
                pOutAdj = pOut->firstAdj;
            }
        }
    }
}

```

```

        cout<<"Anak = ";
        while(pOutAdj != NULL){
            cout<<pOutAdj->nodeAdj<<" ";
            pOutAdj = pOutAdj->nextAdj;
        }
    }
    cout<<endl;
    pOut = pOut->next;

}

}
}

```

```

int main(){
    list first;
    pointerNode node;

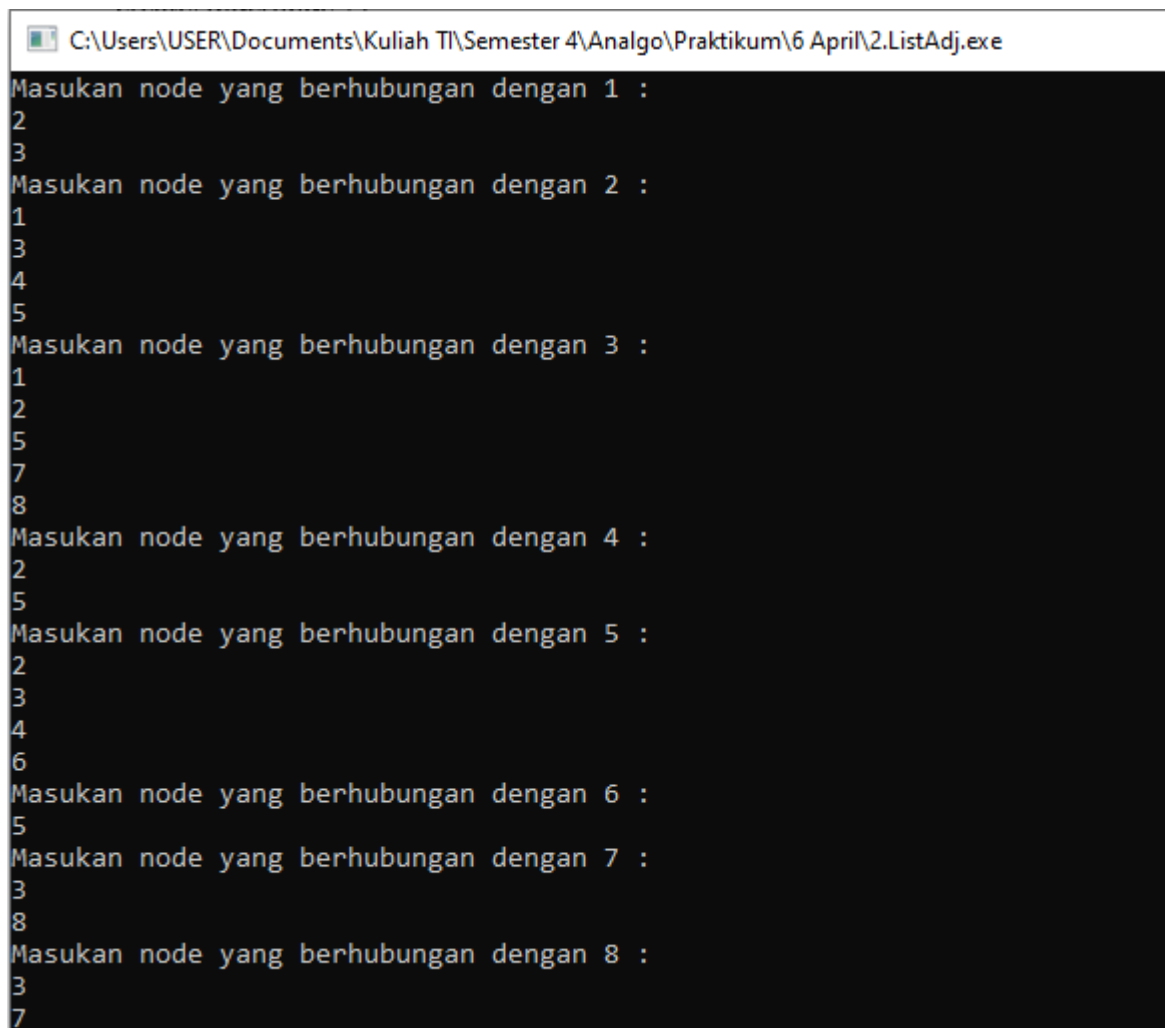
    createListNode(first);

    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 :



```
C:\Users\USER\Documents\Kuliah TI\Semester 4\Analgo\Praktikum\6 April\2.ListAdj.exe  
Masukan node yang berhubungan dengan 1 :  
2  
3  
Masukan node yang berhubungan dengan 2 :  
1  
3  
4  
5  
Masukan node yang berhubungan dengan 3 :  
1  
2  
5  
7  
8  
Masukan node yang berhubungan dengan 4 :  
2  
5  
Masukan node yang berhubungan dengan 5 :  
2  
3  
4  
6  
Masukan node yang berhubungan dengan 6 :  
5  
Masukan node yang berhubungan dengan 7 :  
3  
8  
Masukan node yang berhubungan dengan 8 :  
3  
7
```

C:\Users\USER\Documents\Kuliah TI\Semester 4\Analgo\Praktikum\6 April\2.ListAdj.exe

```
Induk = 1
Anak = 2 3
Induk = 2
Anak = 1 3 4 5
Induk = 3
Anak = 1 2 5 7 8
Induk = 4
Anak = 2 5
Induk = 5
Anak = 2 3 4 6
Induk = 6
Anak = 5
Induk = 7
Anak = 3 8
Induk = 8
Anak = 3 7
Press any key to continue . . .
```

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];
```



```

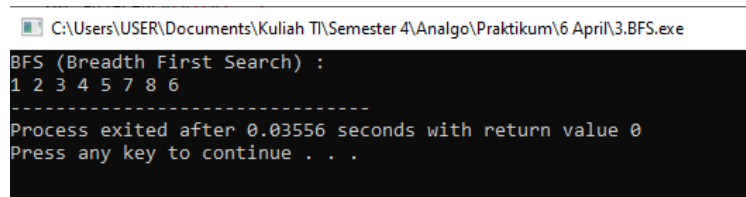
//inisialisasi start
discovered[0] = true;
output[0] = 1;

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 (Breadth First Search) : "<<endl;
for(int i = 0; i < vertexSize; i++){
    cout<<output[i]<<" ";
}
}

```

Screenshot :



```

C:\Users\USER\Documents\Kuliah TI\Semester 4\Analgo\Praktikum\6 April\3.BFS.exe
BFS (Breadth First Search) :
1 2 3 4 5 7 8 6
-----
Process exited after 0.03556 seconds with return value 0
Press any key to continue . . .

```

No 4.

```

#include<bits/stdc++.h>

using namespace std;

// Graph class represents a directed graph

```

```
// using adjacency list representation
```

```
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[])
```

```
{
```

```

        // Mark the current node as visited and
        // print it
        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()
{

    Graph g(8); //Node yang ada
    g.addEdge(1, 2);
    g.addEdge(1, 3);
    g.addEdge(2, 3);

```

```

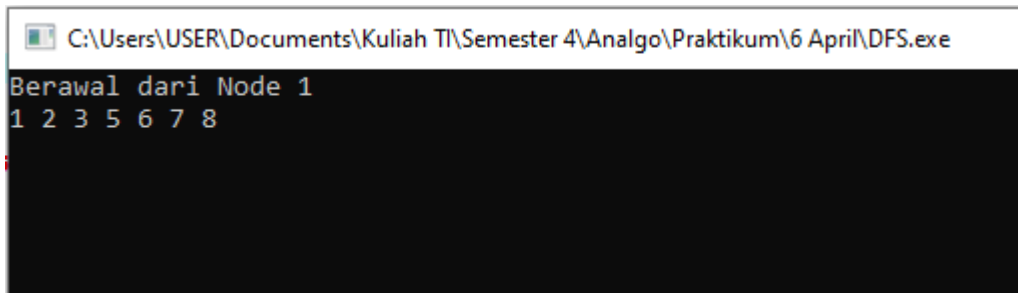
g.addEdge(2, 4);
g.addEdge(2, 5);
g.addEdge(3, 5);
g.addEdge(3, 7);
g.addEdge(3, 8);
    g.addEdge(4, 5);
g.addEdge(5, 6);
g.addEdge(7, 8);

cout << "Berawal dari Node 1 \n";
g.DFS(1);

return 0;
}

```

Screenshot:



```

C:\Users\USER\Documents\Kuliah TI\Semester 4\Analgo\Praktikum\6 April\DFS.exe
Berawal dari Node 1
1 2 3 5 6 7 8

```

Penjelasan

BFS adalah suatu metode yang melakukan pencarian secara melebar, dengan mengunjungi node dari kiri ke kanan dengan level yang sama, apabila semua node sudah dikunjungi pada suatu level, maka lanjut ke level selanjutnya. Dalam worst casenya, BFS menimbang semua jalur untuk semua node yang mungkin, maka $BFS = O(|V| + |E|)$.

DFS adalah suatu metode pencarian mendalam, yang mengunjungi semua node dari yang terkiri lalu geser ke kanan hingga semua node dikunjungi. Kompleksitas ruang DFS ini adalah $O(bm)$, dikarenakan kita hanya butuh menyimpan satu lintasan tunggal dari akar (node induk) sampai daun, ditambah dengan simpul saudara yang belum dikembangkan.