

## 1. Adjacency Matrix

### Kode

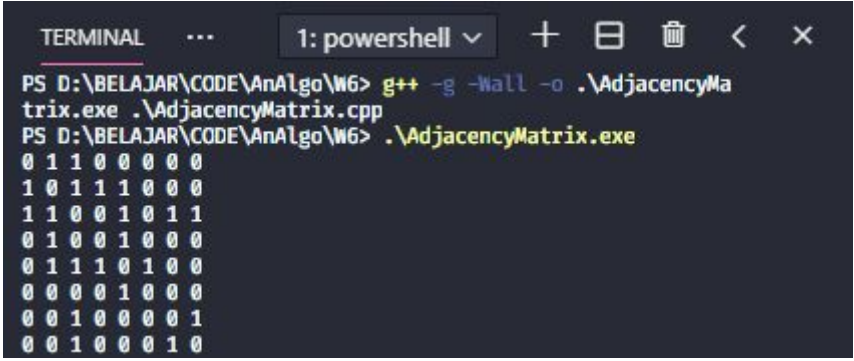
```
#include <iostream>
using namespace std;

int vertArr[20][20];
void add_edge(int origin, int destination)
{
    vertArr[origin-1][destination-1] = 1;
    vertArr[destination-1][origin-1] = 1;
}

void displayMatrix(int node_count)
{
    for (int i = 0; i < node_count; i++)
    {
        for (int j = 0; j < node_count; j++)
        {
            cout << vertArr[i][j] << " ";
        }
        cout << endl;
    }
}

main(int argc, char *argv[])
{
    int n = 8;
    add_edge(1, 2);
    add_edge(1, 3);
    add_edge(2, 3);
    add_edge(2, 4);
    add_edge(2, 5);
    add_edge(3, 5);
    add_edge(3, 7);
    add_edge(3, 8);
    add_edge(4, 5);
    add_edge(5, 6);
    add_edge(7, 8);
    displayMatrix(n);
}
```

## Screenshot



```
TERMINAL 1: powershell
PS D:\BELAJAR\CODE\AnAlgo\W6> g++ -g -Wall -o .\AdjacencyMatrix.exe .\AdjacencyMatrix.cpp
PS D:\BELAJAR\CODE\AnAlgo\W6> .\AdjacencyMatrix.exe
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
```

## 2. Adjacency List

### Kode

```
#include <iostream>

using namespace std;

int vertArr[20][20];
void add_edge(int origin, int destination)
{
    vertArr[origin-1][destination-1] = 1;
    vertArr[destination-1][origin-1] = 1;
}

void displayList(int node_count)
{
    for (int i = 0; i < node_count; i++)
    {
        cout << i+1 << " [ ";
        for (int j = 0; j < node_count; j++)
        {
            if(vertArr[i][j] == 1)
            {
                cout << j+1 << " ";
            }
        }
        cout << "]\n";
    }
}

int main(){
    int n = 8;
    add_edge(1, 2);
    add_edge(1, 3);
```

```

    add_edge(2, 4);
    add_edge(2, 5);
    add_edge(2, 3);
    add_edge(3, 5);
    add_edge(3, 7);
    add_edge(3, 8);
    add_edge(4, 5);
    add_edge(5, 6);
    add_edge(7, 8);
    displayList(n);

    return 0;
}

```

Screenshot

```

TERMINAL  ...  1: powershell v
PS D:\BELAJAR\CODE\AnAlgo\W6> g++ -g -Wall -o .\AdjacencyList.exe .\AdjacencyList.cpp
PS D:\BELAJAR\CODE\AnAlgo\W6> .\AdjacencyList.exe
1 [ 2 3 ]
2 [ 1 3 4 5 ]
3 [ 1 2 5 7 8 ]
4 [ 2 5 ]
5 [ 2 3 4 6 ]
6 [ 5 ]
7 [ 3 8 ]
8 [ 3 7 ]

```

### 3. Breadth First Search (BFS)

Kode

```

#include<iostream>
using namespace std;

int main(){
    int node_count = 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[node_count];
    for(int i = 0; i < node_count; i++){
        discovered[i] = false;
    }
}

```

```

    }
    int output[node_count];

    discovered[0] = true;
    output[0] = 1;

    int counter = 1;
    for(int i = 0; i < node_count; i++){
        for(int j = 0; j < node_count; 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 < node_count; i++){
        cout<<output[i]<<" ";
    }
}

```

Screenshot

```

TERMINAL  ...  1: powershell
PS D:\BELAJAR\CODE\AnAlgo\W6> g++ -g -Wall -o BFS.exe BFS.c
pp
PS D:\BELAJAR\CODE\AnAlgo\W6> .\BFS.exe
BFS :
1 2 3 4 5 7 8 6

```

Analisa

BFS merupakan metode pencarian secara melebar sehingga mengunjungi node dari kiri ke kanan di level yang sama. Apabila semua node pada suatu level sudah dikunjungi semua, maka akan berpindah ke level selanjutnya. Dalam worst case BFS harus mempertimbangkan semua jalur (path) untuk semua node yang mungkin, maka nilai kompleksitas waktu dari BFS adalah  $O(|V| + |E|)$ .

#### 4. Depth First Search (DFS)

Kode

```

#include <iostream>
#include <list>
using namespace std;

class Graph
{

```

```

    int node_count;

    list<int> *adj;

    void DFSUtil(int v, bool visited[]);
public:
    Graph(int node_count)
    {
        this->node_count = node_count;
        adj = new list<int>[node_count];
    }

    void addEdge(int v, int w)
    {
        adj[v].push_back(w);
    }

    void 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 DFS(int v)
    {
        bool *visited = new bool[node_count];
        for (int i = 0; i < node_count; i++)
            visited[i] = false;

        DFSUtil(v, visited);
    }
};

int main()
{
    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 << "Depth First Traversal";
cout << " (Mulai dari Node 1) \n";
g.DFS(1);

return 0;
}

```

Screenshot

```

TERMINAL  ...  1: powershell
PS D:\BELAJAR\CODE\AnAlgo\W6> g++ -g -Wall -o DFS.exe DFS.c
pp
PS D:\BELAJAR\CODE\AnAlgo\W6> .\DFS.exe
Depth First Traversal (Mulai dari Node 1)
1 2 4 5 3 7 8

```

Analisa

DFS merupakan metode pencarian mendalam, yang mengunjungi semua node dari yang ter kiri lalu geser ke kanan hingga semua node dikunjungi. Kompleksitas ruang algoritma DFS adalah  $O(bm)$ , karena kita hanya perlu menyimpan satu buah lintasan tunggal dari akar sampai daun, ditambah dengan simpul saudara kandungnya yang belum dikembangkan.