

**TUGAS PRAKTIKUM 6**  
**ANALISIS ALGORITMA**



**Disusun Oleh:**

**Putri Nabila - 140810180007**

**PROGRAM STUDI S1 TEKNIK INFORMATIKA**  
**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM**  
**UNIVERSITAS PADJADJARAN**  
**2020**

## 1. Matrix Adjacency

```
#include <iostream>

using namespace std;

int vertArr[20][20];

int count = 0;

void displayMatrix(int v)
{
    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)
{
    vertArr[u][v] = 1;
    vertArr[v][u] = 1;
}

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

    cout << "Masukkan jumlah matrix : ";cin >> v;

    int pilihan,a,b;
    while(true){
        cout << "Pilihan menu : " << endl;
        cout << "1. Tambah edge " << endl;
```

```

cout << "2. Print " << endl;

cout << "3. Exit " << endl;

cout << "Masukan pilihan : "; cin >> pilihan;

switch (pilihan)
{
    case 1:

        cout << "Masukkan node A : "; cin >> a;

        cout << "Masukkan node B : "; cin >> b;

        add_edge(a,b);

        cout << "Edge telah ditambahkan\n";

        system("Pause");

        system("CLS");

        break;

    case 2:

        displayMatrix(v);

        system("Pause");

        system("CLS");

        break;

    case 3:

        return 0;

        break;

    default:

        break;

}

}

}

```

**Screenshot**

```
D:\Semester 4\Analgo\Praktikum\AnalgoKu\AnalgoKu6\matrks adjacency.exe
Pilihan menu :
1. Tambah edge
2. Print
3. Exit
Masukan pilihan : 2
0 0 0 0
0 0 0 1
0 0 1 0
0 1 0 0
Press any key to continue . . .
```

## 2. List Adjacency

```
#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 = 0; 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()
{
    int pilihan,a,b,n;
    cout<<"Banyak node : ";cin>>n;
    Graph gh(n);
    for(;;)
    {
        cout<<"\nMenu\n"
        <<"1. Tambah edge\n"
        <<"2. Print Edge\n"
        <<"0. Exit\n\n"
        <<"Pilihan : ";cin>>pilihan;

        switch (pilihan)
        {
            case 1:
                cout<<"\nedge(a,b)\n"
                <<"Input a : ";cin>>a;
                cout<<"Input b : ";cin>>b;

```

```

        gh.addEdge(a,b);
        continue;

    case 2:
        gh.printGraph();
        continue;
    case 0:
        return 0;
        break;

    default:
        continue;
    }
}
return 0;
}

```

### Screenshot

```

D:\Semester 4\Analgo\Praktikum\AnalgoKu\AnalgoKu6\list adjacency.exe
Banyak node : 8

Menu
1. Tambah edge
2. Print Edge
0. Exit

Pilihan : 1

edge(a,b)
Input a : 3
Input b : 2

Menu
1. Tambah edge
2. Print Edge
0. Exit

Pilihan : 2

Adjacency list of vertex 0
head

Adjacency list of vertex 1
head

Adjacency list of vertex 2
head -> 3

Adjacency list of vertex 3
head -> 2

Adjacency list of vertex 4
head

Adjacency list of vertex 5
head

Adjacency list of vertex 6
head

Adjacency list of vertex 7
head

Menu
1. Tambah edge
2. Print Edge
0. Exit

Pilihan : 0

```

### 3. BFS

```
#include<iostream>
#include <list>

using namespace std;

class Graph
{
    int V;
    list<int> *adj;
public:
    Graph(int V);
    void addEdge(int v, int w);

    void BFS(int s);
};

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::BFS(int s)
{
    bool *visited = new bool[V];
    for(int i = 0; i < V; i++)
        visited[i] = false;

    list<int> queue;

    visited[s] = true;
    queue.push_back(s);

    list<int>::iterator i;

    while(!queue.empty())
    {
```

```

        s = queue.front();
        cout << s << " ";
        queue.pop_front();

        for (i = adj[s].begin(); i != adj[s].end(); ++i)
        {
            if (!visited[*i])
            {
                visited[*i] = true;
                queue.push_back(*i);
            }
        }
    }
}

int main()
{

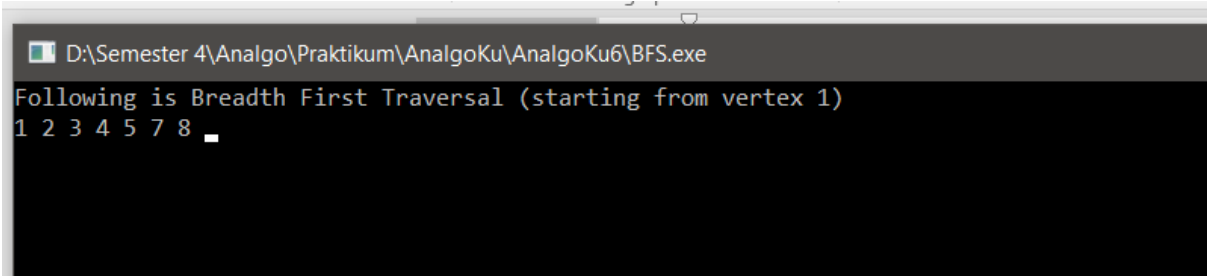
    Graph g(8);
    g.addEdge(1, 2);
    g.addEdge(1, 3);
    g.addEdge(2, 4);
    g.addEdge(2, 5);
    g.addEdge(2, 3);
    g.addEdge(3, 7);
    g.addEdge(3, 8);
    g.addEdge(4, 5);
    g.addEdge(5, 3);
    g.addEdge(5, 6);
    g.addEdge(7, 8);

    cout << "Following is Breadth First Traversal "
        << "(starting from vertex 1) \n";
    g.BFS(1);

    return 0;
}

```

### Screenshot



The screenshot shows a Windows command prompt window with the title bar "D:\Semester 4\Analgo\Praktikum\AnalgoKu\AnalgoKu6\BFS.exe". The output of the program is displayed in the command prompt, showing the text "Following is Breadth First Traversal (starting from vertex 1)" followed by the sequence of vertices "1 2 3 4 5 7 8" on the next line. A cursor is visible at the end of the second line of output.

```

D:\Semester 4\Analgo\Praktikum\AnalgoKu\AnalgoKu6\BFS.exe
Following is Breadth First Traversal (starting from vertex 1)
1 2 3 4 5 7 8

```



Analisis :

- 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  $O(|V| + |E|)$ .
- Karena Big-O dari BFS adalah  $O(V+E)$  dimana V itu jumlah vertex dan E itu adalah jumlah edges maka Big-O =  $O(n)$  dimana  $n = v + e$
- Maka dari itu Big- $\Theta$  nya adalah  $\Theta(n)$ .

#### 4. DFS

// C++ program to print DFS traversal from

// a given vertex in a given graph

```
#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()
{
    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 << "Following is Depth First Traversal"
      " (starting from vertex 1) \n";
g.DFS(1);
return 0;
}

```

### Screenshot

```

D:\Semester 4\Analgo\Praktikum\AnalgoKu\AnalgoKu6\DFS.exe
Following is Depth First Traversal (starting from vertex 1)
1 2 4 5 3 7 8
-----
Process exited after 3.11 seconds with return value 3221225477
Press any key to continue . . .

```

Analisis :

- 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-simpul saudara kandungnya yang belum dikembangkan.
- Big O kompleksitas total DFS () adalah  $(V + E)$ .  
 $O(n)$   
 Dengan  $V$  = Jumlah Verteks  
 Dan  $E$  = Jumlah Edges