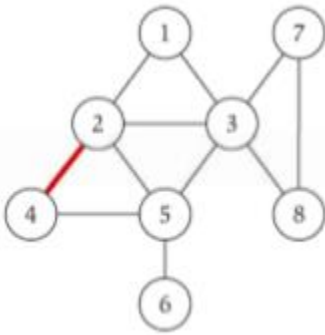


Nama : Sharashena Chairani

NPM : 140810180022

### Tugas 6

1. Dengan menggunakan undirected graph dan adjacency matrix berikut, buatlah koding programnya menggunakan bahasa C++.



	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	1	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

```
/*
Nama : Sharashena Chairani
NPM : 140810180022
Program : Adjacency Matriks
*/

#include <iostream>
using namespace std;

void input(int &n){
    cout << "~~~~~" << endl;
    cout<< "Banyak node dalam graph agar membentuk matriks n x n: ";
    cin >> n;
    cout << "~~~~~" << endl;
}

void input(int arr[], int size){
    for (int i = 0; i < size; i++) {
        cout << "Masukkan nilai simpul ke-" << i+1 << " : ";
        cin >> arr[i];
    }
}

void input(int &num, int value){
    num = value;
}

void printMatrix(int arr[], int n){
```

```

    for (int i = 0; i < n; i++) {
        cout << arr[i] << "\t";
    }
}

bool check(int arr[], int n, int value){
    for (int i = 0; i < n; i++) {
        if (arr[i] == value)
            return true;
    }

    return false;
}

main(){
    int n, val;
    input(n);

    int simpul[n];
    input(simpul, n);

    int garis[n][n];

    cout << "\n~~~~~Adjacency Matrix~~~~~" << endl;
    for (int i = 0; i < n; i++) {
        int tepi;
        cout << "\nJumlah garis simpul ke-" << i+1
            << " (" << simpul[i] << ") : ";
        cin >> tepi;

        for (int j = 0; j < tepi; j++) {
            bool found = false;

            do {
                cout << "Simpul garis ke-" << i+1 << " : ";
                cin >> val;

                found = check(simpul, n, val);
                if (!found)
                    cout << endl << "Simpul tidak ditemukan!" << endl;
            } while (!found);

            input(garis[i][j], val);
        }
    }

    cout << endl;

```

```

        cout << "~~~~~" <<
endl;
cout << endl << "Adjacency matrix dari undirected graph : \n\t" << endl;
printMatrix(simpul, n);

for (int i = 0; i < n; i++) {
    cout << endl << simpul[i] << "\t";

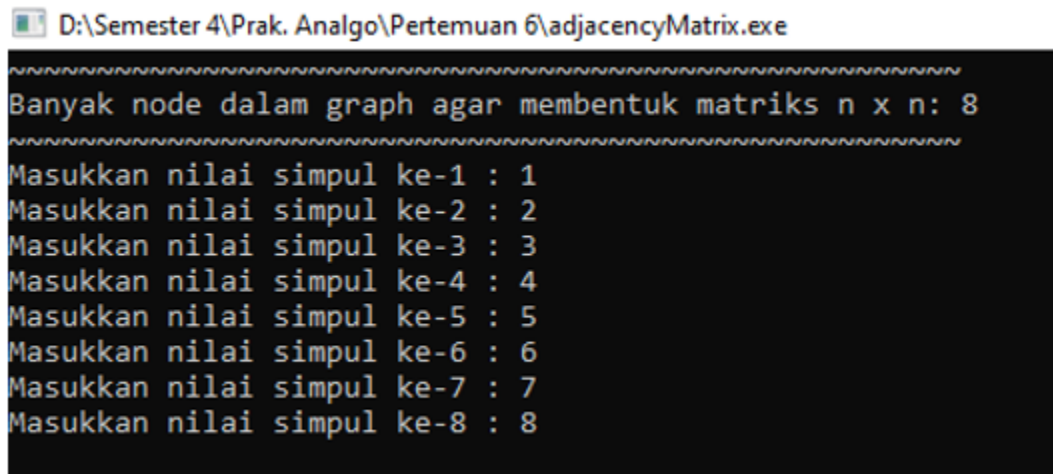
    for (int j = 0; j < n; j++) {
        bool found = false;

        for (int k = 0; k < n; k++) {
            if (garis[i][k] == simpul[j])
                found = true;
        }

        if(found)
            cout << "1\t";
        else
            cout << "0\t";
        }
    }
}

```

### Screenshot Compile



```

D:\Semester 4\Prak. Analgo\Pertemuan 6\adjacencyMatrix.exe
~~~~~
Banyak node dalam graph agar membentuk matriks n x n: 8
~~~~~
Masukkan nilai simpul ke-1 : 1
Masukkan nilai simpul ke-2 : 2
Masukkan nilai simpul ke-3 : 3
Masukkan nilai simpul ke-4 : 4
Masukkan nilai simpul ke-5 : 5
Masukkan nilai simpul ke-6 : 6
Masukkan nilai simpul ke-7 : 7
Masukkan nilai simpul ke-8 : 8

```

~~~~~Adjacency Matrix~~~~~

Jumlah garis simpul ke-1 (1) : 2  
Simpul garis ke-1 : 2  
Simpul garis ke-1 : 3

Jumlah garis simpul ke-2 (2) : 4  
Simpul garis ke-2 : 1  
Simpul garis ke-2 : 3  
Simpul garis ke-2 : 4  
Simpul garis ke-2 : 5

Jumlah garis simpul ke-3 (3) : 5  
Simpul garis ke-3 : 1  
Simpul garis ke-3 : 2  
Simpul garis ke-3 : 5  
Simpul garis ke-3 : 7  
Simpul garis ke-3 : 8

Jumlah garis simpul ke-4 (4) : 2  
Simpul garis ke-4 : 2  
Simpul garis ke-4 : 5

Jumlah garis simpul ke-5 (5) : 4  
Simpul garis ke-5 : 2  
Simpul garis ke-5 : 3  
Simpul garis ke-5 : 4  
Simpul garis ke-5 : 6

Jumlah garis simpul ke-6 (6) : 1  
Simpul garis ke-6 : 5

D:\Semester 4\Prak. Analgo\Pertemuan 6\adjacencyMatrix.exe

Jumlah garis simpul ke-7 (7) : 2  
Simpul garis ke-7 : 3  
Simpul garis ke-7 : 8

Jumlah garis simpul ke-8 (8) : 2  
Simpul garis ke-8 : 3  
Simpul garis ke-8 : 7

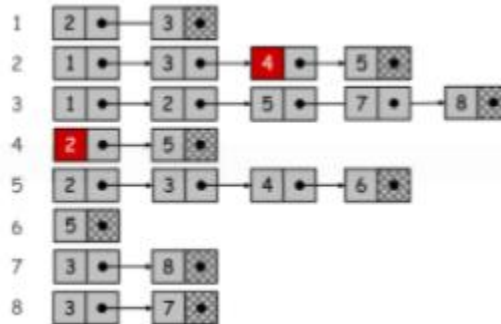
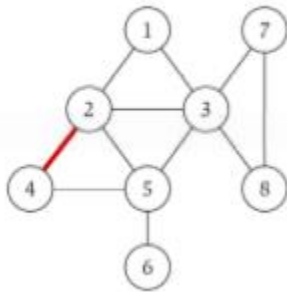
~~~~~

Adjacency matrix dari undirected graph :

1	2	3	4	5	6	7	8	
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	0	1	0	0	1
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	1

-----  
Process exited after 108 seconds with return value 0  
Press any key to continue . . .

2. Dengan menggunakan undirected graph dan representasi adjacency list, buatlah koding programnya menggunakan bahasa C++.



```

/*
Nama : Sharashena Chairani
NPM : 140810180022
Program : Adjacency List
*/

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

struct nodeList
{
    int dest;
    struct nodeList* next;
};

struct AdjList
{
    struct nodeList *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;
    }
};

```

```

    }

    nodeList* newAdjListNode(int dest)
    {
        nodeList* simpulBaru = new nodeList;
        simpulBaru->dest = dest;
        simpulBaru->next = NULL;
        return simpulBaru;
    }

    void addEdge(int src, int dest)
    {
        nodeList* simpulBaru = newAdjListNode(dest);
        simpulBaru->next = array[src].head;
        array[src].head = simpulBaru;
        simpulBaru = newAdjListNode(src);
        simpulBaru->next = array[dest].head;
        array[dest].head = simpulBaru;
    }


    void printGraph()
    {
        int v;
        for (v = 1; v < V; ++v)
        {
            nodeList* 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()
{
    Graph v(8);
    v.addEdge(1, 2);
    v.addEdge(1, 3);
    v.addEdge(2, 4);
    v.addEdge(2, 5);
    v.addEdge(2, 3);
    v.addEdge(3, 7);

```

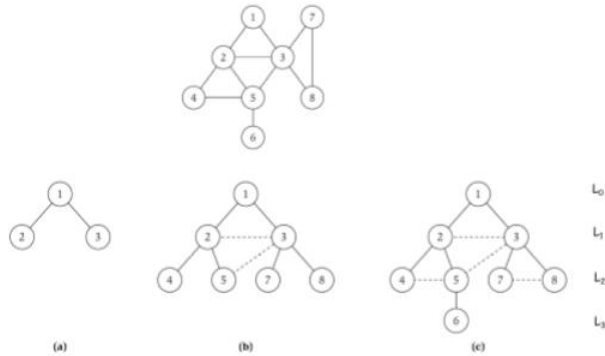
```
v.addEdge(3, 8);  
v.addEdge(4, 5);  
v.addEdge(5, 3);  
v.addEdge(5, 6);  
v.addEdge(7, 8);  
v.printGraph();  
  
return 0;  
}
```

## Screenshot Compile

 D:\Semester 4\Prak. Analgo\Pertemuan 6\adjacencyList.exe

```
Adjacency list of vertex 1  
head -> 3-> 2  
  
Adjacency list of vertex 2  
head -> 3-> 5-> 4-> 1  
  
Adjacency list of vertex 3  
head -> 5-> 8-> 7-> 2-> 1  
  
Adjacency list of vertex 4  
head -> 5-> 2  
  
Adjacency list of vertex 5  
head -> 6-> 3-> 4-> 2  
  
Adjacency list of vertex 6  
head -> 5  
  
Adjacency list of vertex 7  
head -> 8-> 3  
  
-----  
Process exited after 0.1076 seconds with return value 0  
Press any key to continue . . .
```

3. Buatlah program Breadth First Search dari algoritma BFS yang telah diberikan. Kemudian uji coba program Anda dengan menginputkan undirected graph sehingga menghasilkan tree BFS. Hitung dan berikan secara asimptotik berapa kompleksitas waktunya dalam Big- $\Theta$ !



### Program BFS

/\*

Nama : Sharashena Chairani

NPM : 140810180022

Program : Breadth First Search

\*/

#include<iostream>

using namespace std;

```
int main(){
    int simpulSize = 8;
    int adj[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 visited[simpulSize];
    for(int i = 0; i < simpulSize; i++){
        visited[i] = false;
    }
    int output[simpulSize];

    //inisialisasi start
```



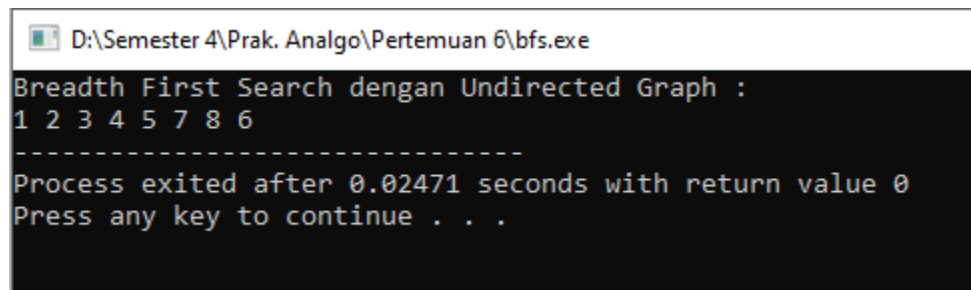
```

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

int counter = 1;
for(int i = 0; i < simpulSize; i++){
    for(int j = 0; j < simpulSize; j++){
        if((adj[i][j] == 1)&&(visited[j] == false)){
            output[counter] = j+1;
            visited[j] = true;
            counter++;
        }
    }
}
cout<<"Breadth First Search dengan Undirected Graph : "<<endl;
for(int i = 0; i < simpulSize; i++){
    cout<<output[i]<<" ";
}
}

```

### Screenshot Compile



```

D:\Semester 4\Prak. Analgo\Pertemuan 6\bfs.exe
Breadth First Search dengan Undirected Graph :
1 2 3 4 5 7 8 6
-----
Process exited after 0.02471 seconds with return value 0
Press any key to continue . . .

```

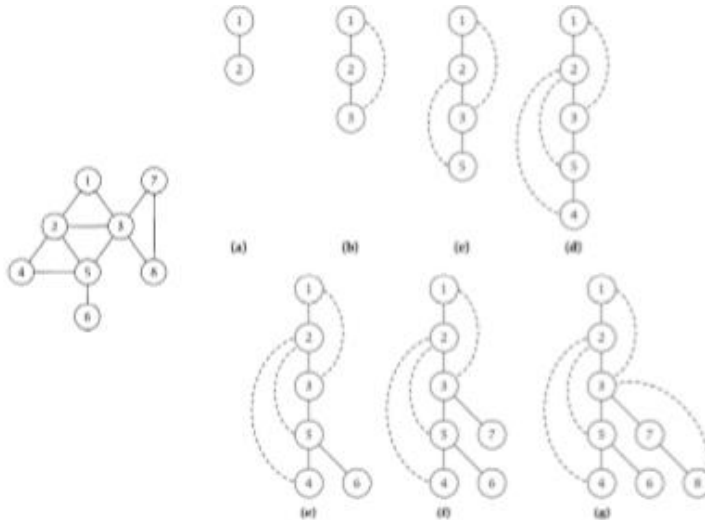
### Kompleksitas Waktu Big $\Theta$

BFS adalah metode pencarian secara melebar, jadi mencari di 1 level dulu dari kiri ke kanan. Bila semua node atau simpulnya telah dikunjungi maka pencarian dilanjutkan ke level berikutnya.

Worst case BFS harus mempertimbangkan semua jalur (path) untuk semua simpul atau 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. Buatlah program Depth First Search dari algoritma DFS yang telah diberikan. Kemudian uji coba program Anda dengan menginputkan undirected graph sehingga menghasilkan tree DFS. Hitung dan berikan secara asimptotik berapa kompleksitas waktunya dalam Big- $\Theta$ !



### Program DFS

/\*

Nama : Sharashena Chairani

NPM : 140810180022

Program : Depth First Search

\*/

```
#include <iostream>
```

```
#include <list>
```

```
using namespace std;
```

```
class Graph{
```

```
    int N;
```

```
    list<int> *adjacency;
```

```
    void DFSUtil(int u, bool visited[]){
```

```
        visited[u] = true;
```

```
        cout << u << " ";
```

```
        list<int>::iterator i;
```

```
        for(i = adjacency[u].begin(); i != adjacency[u].end(); i++){
```

```
            if(!visited[*i]){
```

```

        DFSUtil(*i, visited);
    }
}

public :
    Graph(int N){
        this->N = N;
        adjacency = new list<int>[N];
    }

    void addEdge(int u, int v){
        adjacency[u].push_back(v);
    }

    void DFS(int u){
        bool *visited = new bool[N];
        for(int i = 0; i < N; i++){
            visited[i] = false;
        }
        DFSUtil(u, visited);
    }
};

int main(){
    Graph gh(8);

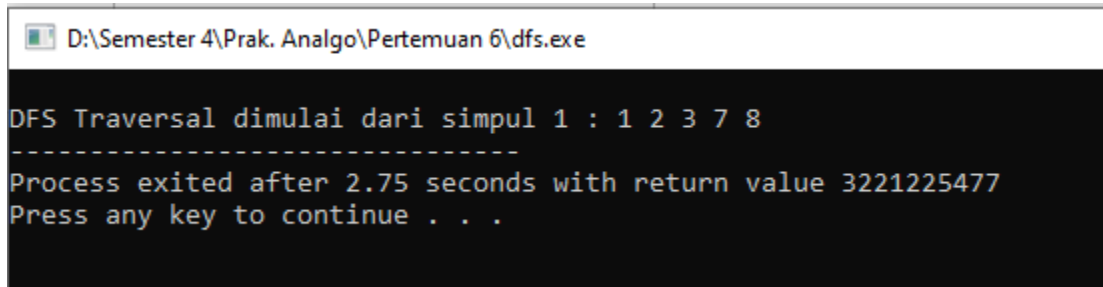
    gh.addEdge(1,2);
    gh.addEdge(1,3);
    gh.addEdge(2,3);
    gh.addEdge(2,4);
    gh.addEdge(2,5);
    gh.addEdge(3,7);
    gh.addEdge(3,8);
    gh.addEdge(4,5);
    gh.addEdge(5,3);
    gh.addEdge(5,6);
    gh.addEdge(7,8);

    cout << "\nDFS Traversal dimulai dari simpul 1 : ";
    gh.DFS(1);

}

```

## Screenshot Compile



```
D:\Semester 4\Prak. Analgo\Pertemuan 6\dfs.exe

DFS Traversal dimulai dari simpul 1 : 1 2 3 7 8
-----
Process exited after 2.75 seconds with return value 3221225477
Press any key to continue . . .
```

DFS merupakan metode pencarian mendalam, yang **mengunjungi semua node** dari yang ter kiri lalu geser ke kanan hingga semua node dikunjungi.

### Kompleksitas waktu algoritma DFS

- a. Vertex awal (kiri) dikunjungi, lalu dicetak :  $O(1)$
- b. Semua vertexnya rekursif :  $T(E/1)$
- c. Vertex yang belum dikunjungi ditandai :  $O(V)$
- d. Rekursif untuk mencetak DFS :  $T(V/1)$

$$\begin{aligned}T(n) &= O(1) + T(E/1) + O(V) + T(V/1) \\&= O(1) + O(E) + O(V) + O(V) \\&= O(\max(1, E)) + O(V) + O(V) \\&= O(E) + O(\max(V, V)) \\&= O(E) + O(V)\end{aligned}$$

$$T(n) = O(V+E)$$