

**Jawaban Soal 1**

**Program C++**

/\*  Nama        : Sitti Ufairoh Azzahra

    NPM         : 140810180002

    Kelas       : B

    Deskripsi   : Program Adjacency Matrix

\*/

#include <iostream>

using namespace std;

int node[20][20];

void addEdge(int u, int v){

    node[u][v] = 1;

    node[v][u] = 1;

}

void displayMatrix(int n){

    for (int i = 1; i <=n; i++)

    {

        for(int j = 1; j <= n; j++)

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

        cout << endl;

    }

}

int main(){

    int n = 8;

    cout << "Program Undirected Graph for Adjacency Matrix" << endl;

    cout << endl;

    addEdge(1,2);

    addEdge(1,3);

    addEdge(2,4);

    addEdge(2,5);

    addEdge(2,3);

    addEdge(3,5);

    addEdge(3,7);

    addEdge(3,8);

    addEdge(4,5);

    addEdge(5,6);

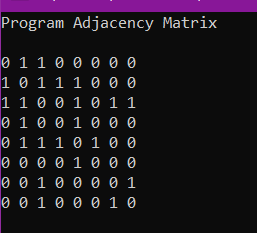
    addEdge(7,8);

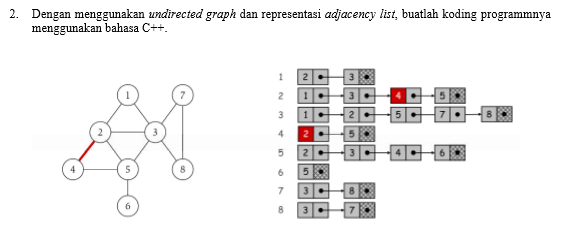
    displayMatrix(n);

    return 0;

}

**Hasil Program :**





**Jawaban Soal 2**

**Program C++**

/\*  Nama        : Sitti Ufairoh Azzahra

    NPM         : 140810180002

    Kelas       : B

    Deskripsi   : Program Adjacency List

\*/

#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 = 1; v <= V; ++v){

                AdjListNode\* pCrawl = array[v].head;

                cout<<"\n Adjacency list dari "<<v<<"\n head ";

                while (pCrawl)

                {

                    cout<<"-> "<<pCrawl->dest;

                    pCrawl = pCrawl->next;

                }cout<<endl;

            }

        }

};

int main(){

    cout << "Program Adjacency List" << endl;

    Graph gh(8);

        gh.addEdge(1, 2);

        gh.addEdge(1, 3);

        gh.addEdge(2, 4);

        gh.addEdge(2, 5);

        gh.addEdge(2, 3);

        gh.addEdge(3, 7);

        gh.addEdge(3, 8);

        gh.addEdge(4, 5);

        gh.addEdge(5, 3);

        gh.addEdge(5, 6);

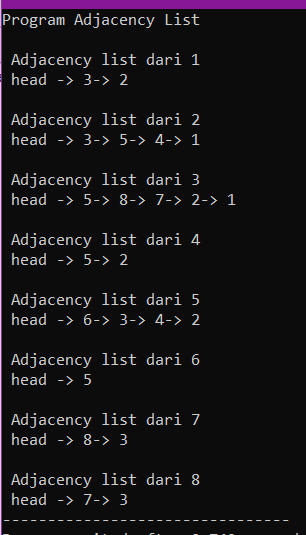
        gh.addEdge(7, 8);

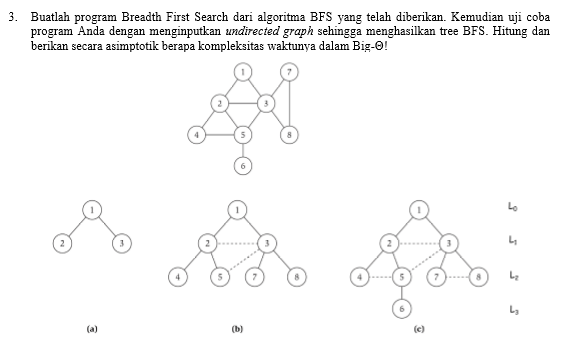
    gh.printGraph();

    return 0;

}

**Hasil Program :**





**Jawaban Soal 3**

**Program C++**

/\*  Nama        : Sitti Ufairoh Azzahra

    NPM         : 140810180002

    Kelas       : B

    Deskripsi   : Program  Breadth First Search

\*/

#include <iostream>

#include <list>

using namespace std;

class Graph{

    int N;

    list<int> \*adj;

    public :

        Graph(int N){

            this->N == N;

            adj = new list<int>[N];

        }

        void addEdge(int u, int v){

            adj[u].push\_back(v);

        }

        void BFS(int s){

            bool \*visited = new bool[N];

            for(int i = 0; i < N; 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(2,3);

    g.addEdge(2,4);

    g.addEdge(2,5);

    g.addEdge(3,7);

    g.addEdge(3,8);

    g.addEdge(4,5);

    g.addEdge(5,3);

    g.addEdge(5,6);

    g.addEdge(7,8);

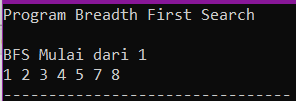
    cout << "BFS Mulai dari 1" << endl;

    g.BFS(1);

    return 0;

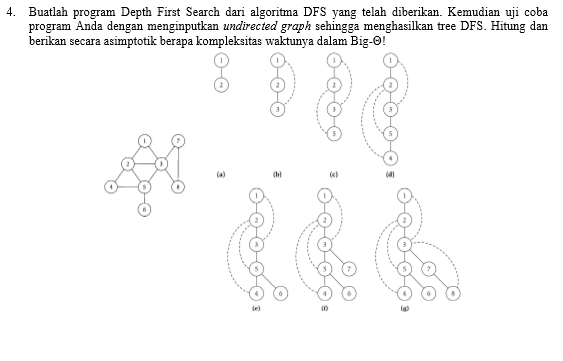
}

**Hasil Program :**



**Komplesitas Waktu BFS**

* 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-Ө nya adalah Ө(n)



**Jawaban Soal 3**

**Program C++**

/\*  Nama        : Sitti Ufairoh Azzahra

    NPM         : 140810180002

    Kelas       : B

    Deskripsi   : Program  Depth First Search

\*/

#include <iostream>

#include <list>

using namespace std;

class Graph{

    int N;

    list<int> \*adj;

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

        visited[u] = true;

        cout << u << " ";

        list<int>::iterator i;

        for(i = adj[u].begin(); i != adj[u].end(); i++)

        {

            if(!visited[\*i])

                DFSUtil(\*i, visited);

        }

    }

public :

    Graph(int N){

        this->N = N;

        adj = new list<int>[N];

    }

    void addEdge(int u, int v){

        adj[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(){

    cout << "Program Depth First Search " << endl;

    Graph g(8);

    g.addEdge(1,2);

    g.addEdge(1,3);

    g.addEdge(2,3);

    g.addEdge(2,4);

    g.addEdge(2,5);

    g.addEdge(3,7);

    g.addEdge(3,8);

    g.addEdge(4,5);

    g.addEdge(5,3);

    g.addEdge(5,6);

    g.addEdge(7,8);

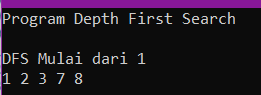
    cout << "\nDFS Mulai dari 1" << endl;

    g.DFS(1);

    return 0;

}

**Hasil Program :**



**Komplesitas Waktu DFS**

* Kompleksitas ruang algoritma DFS adalah O(bm), karena kita hanya 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