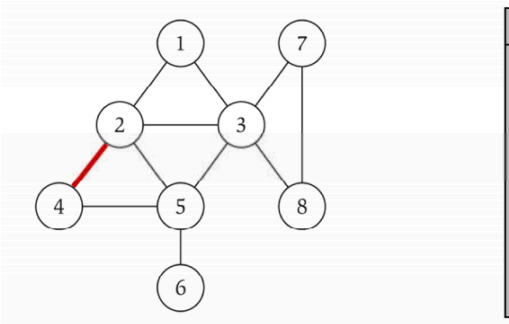
Aprischa Nauva Miliantari
140810180063
Tugas 6

Tugas

1. Dengan menggunakan *undirected graph* dan *adjacency matrix* berikut, buatlah koding programmnya 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

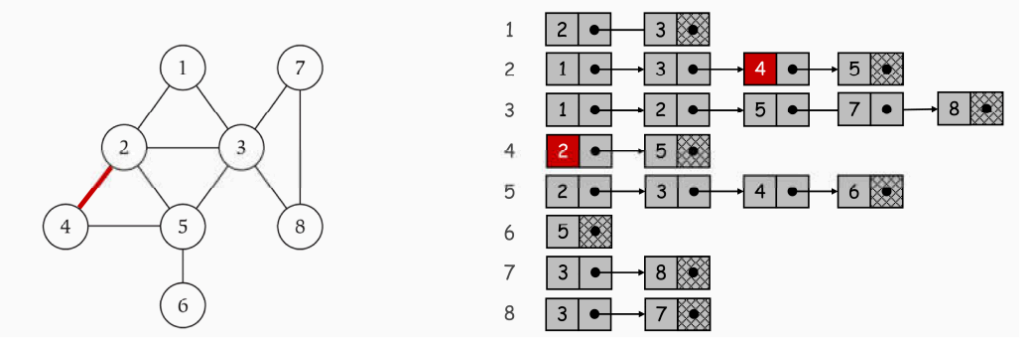
Jawab :

```
/*
 Nama  : Aprischa Nauva M
 NPM   : 140810180063
 Kelas : A
*/

#include <iostream>
#include <cstdlib>
using namespace std;
#define MAX 20

class AdjacencyMatrix {
private:
    int n;
    int **adj;
    bool *visited;
public:
    AdjacencyMatrix(int n) {
        this->n = n;
        visited = new bool [n];
        adj = new int* [n];
        for (int i = 0; i < n; i++){
            adj[i] = new int [n];
            for(int j = 0; j < n; j++){
                adj[i][j] = 0;
            }
        }
    }

    void add_edge(int origin, int destin){
        if( origin > n || destin > n || origin < 0 || destin < 0){
            cout<<"Invalid edge!\n";
        }
        else{
            adj[origin - 1][destin - 1] = 1;
        }
    }
```

```
    void display(){
        int i,j;
        for(i = 0;i < n;i++){
            for(j = 0; j < n; j++)
                cout<<adj[i][j]<<"  ";
            cout<<endl;
        }
    }
};

int main(){
    int nodes, max_edges, origin, destin;
    cout<<"Enter number of nodes: ";
    cin>>nodes;
    AdjacencyMatrix am(nodes);
    max_edges = nodes * (nodes - 1);
    for (int i = 0; i < max_edges; i++){
        cout<<"Enter edge (-1 -1 to exit): ";
        cin>>origin>>destin;
        if((origin == -1) && (destin == -1))
            break;
        am.add_edge(origin, destin);
    }
    am.display();
    return 0;
}
```

```
Enter number of nodes: 8
Enter edge (-1 -1 to exit): 1 2
Enter edge (-1 -1 to exit): 1 3
Enter edge (-1 -1 to exit): 2 1
Enter edge (-1 -1 to exit): 2 3
Enter edge (-1 -1 to exit): 2 4
Enter edge (-1 -1 to exit): 2 5
Enter edge (-1 -1 to exit): 3 1
Enter edge (-1 -1 to exit): 3 2
Enter edge (-1 -1 to exit): 3 5
Enter edge (-1 -1 to exit): 3 7
Enter edge (-1 -1 to exit): 3 8
Enter edge (-1 -1 to exit): 4 2
Enter edge (-1 -1 to exit): 4 4
Enter edge (-1 -1 to exit): 4 5
Enter edge (-1 -1 to exit): 5 2
Enter edge (-1 -1 to exit): 5 3
Enter edge (-1 -1 to exit): 5 4
Enter edge (-1 -1 to exit): 5 6
Enter edge (-1 -1 to exit): 6 5
Enter edge (-1 -1 to exit): 7 3
Enter edge (-1 -1 to exit): 7 8
Enter edge (-1 -1 to exit): 8 3
Enter edge (-1 -1 to exit): 8 7
Enter edge (-1 -1 to exit): -1 -1
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
Program ended with exit code: 0
```

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



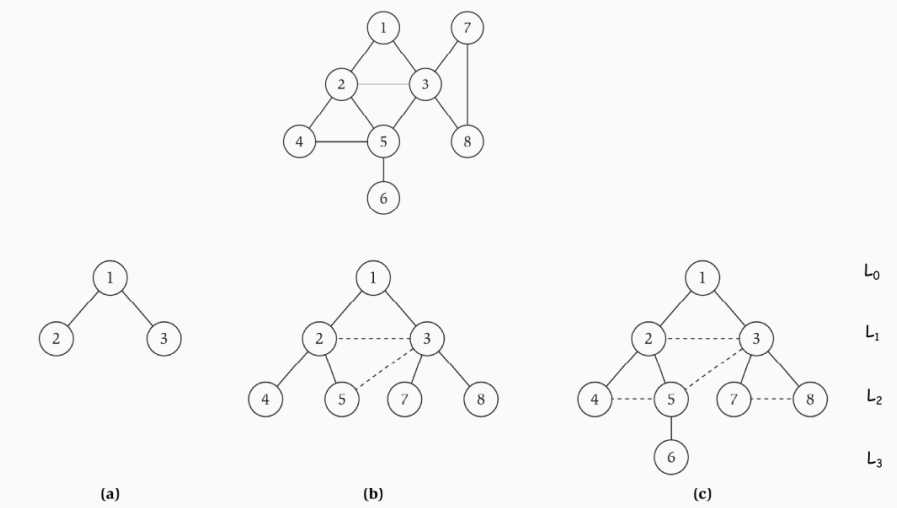
Jawab :

```
/*
 Nama  : Aprischa Nauva Miliantari
 NPM   : 140810180063
 Kelas : A
*/

#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 of vertex "<<v<<"\n head ";
            while (pCrawl){
                cout<<"-> "<<pCrawl->dest;
                pCrawl = pCrawl->next;
            }
            cout<<endl;
        }
    }
};

int main(){
    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;
}
```

```
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

Adjacency list of vertex 8
head -> 7-> 3
Program ended with exit code: 0
```

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-Θ!



Jawab :

```
/*
 Nama  : Aprischa Nauva Miliantari
 NPM   : 140810180063
 Kelas : A
*/

#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;
}
```

```
Following is Breadth First Traversal (starting from vertex 1)
1 2 3 4 5 7 8
```

➔ Kompleksitas Waktu
Karena Big-O dari BFS adalah O(|V| + |E|) dimana V itu jumlah vector dan E itu adalah jumlah edges maka Big-O = O(n) dimana n = v+e
Maka dari itu Big-Θ nya adalah Θ(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-Θ!

   Jawab :

```
/*
 Nama  : Aprischa Nauva M
 NPM   : 140810180063
 Kelas : A
 */

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

```
    g.addEdge(7, 8);

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

    return 0;
}
```

```
Following is Depth First Traversal(starting from vertex 1)
1 2 5 6 4 3 7 8
```

➔ Kompleksitas Waktu

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