Bunga Azizha N
140810180016
Tugas 6

Tugas

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

```
/* Bunga Azizha N - 140810180016
Kelas B - Program Adjacency Matrix */

#include<iostream>
using namespace std;

int vertArr[20][20]; //the adjacency matrix initially 0
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) { //function to add edge into the matrix
  vertArr[u][v] = 1;
  vertArr[v][u] = 1;
}

main(int argc, char* argv[]) {
  int v = 8;   //there are 8 vertices in the graph
  add_edge(1, 2);
  add_edge(1, 3);
  add_edge(2, 4);
  add_edge(2, 5);
  add_edge(3, 2);
  add_edge(3, 8);
  add_edge(4, 5);
  add_edge(5, 6);
  add_edge(7, 3);
  add_edge(8, 7);
  displayMatrix(v);
}
```

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

```
/* Bunga Azizha N - 140810180016
Kelas B - Program Adjacency list */

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

int main()
{
	vector<int> adj[100];
	int n,m,u,v,i;
	cout<<"Enter  number of vertices and edges:\n";
	cin>>n>>m;
	cout<<"Enter edges (u and v):\n";
	for(i=0;i<m;i++)
	{
		cin>>u>>v;
		adj[u].push_back(v);
		adj[v].push_back(u);
	}

	for(i=0;i<n;i++)
	{
		cout<<i;
		for(auto it: adj[i])
		{
			cout<<"->"<<it;
		}
		cout<<endl;
	}
}
```

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!

```
/* Bunga Azizha N - 140810180016
Kelas B - Program BFS */

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

struct Graph {
   int vertex;
   list<int>* edge;
};

void makeGraph(Graph& G, int vertex);
void addEdge(Graph& G, int i, int j);
void traversal(Graph G);
void BFS(Graph G, int s);

int main() {
   Graph G;
   int n, m, val, edge;

   cout << "Masukkan banyak simpul dalam graph : ";
   cin >> n;
   makeGraph(G, n);

   for (int i = 0; i < n; i++) {
      cout << endl << "Nilai simpul ke-" << i+1 << " : ";
      cin >> val;
      cout << "Banyak edge dari simpul " << val << " : ";
      cin >> m;

      for (int j = 0; j < m; j++) {
         cout << "Simpul edge ke-" << j+1 << " : ";
         cin >> edge;
         addEdge(G, val, edge);
      }
   }
   cout << "\nBFS dimulai dari simpul "
      << *G.edge[0].begin() - 1 << endl;
                  BFS(G, *G.edge[0].begin() - 1);
}
```

```
void makeGraph(Graph& G, int vertex) {
   G.vertex = vertex;
   G.edge = new list<int>[vertex];
}

void addEdge(Graph& G, int i, int j) {
   G.edge[i].push_back(j);
}

void traversal(Graph G) {
   for (int i = 0; i < G.vertex; ++i)
   {
      cout << "\n vertex "<<i<<"\n head";
      for (auto x : G.edge[i])
         cout << " -> " << x;
      cout << endl;
   }
}

void BFS(Graph G, int s) {
   bool *visited = new bool[G.vertex];
   for (int i=0; i<G.vertex; i++)
      visited[i] = false;

   list<int> queue;
   visited[s] = true;
   queue.push_back(s);

   while (!queue.empty()) {
      s = queue.front();
      cout<<s<<" ";
      queue.pop_front();

      for (list<int>::iterator i=G.edge[s].begin(); i != G.edge[s].end(); ++i) {
         if (!visited[*i]) {
            visited[*i] = true;
            queue.push_back(*i);
         }
      }
   }
}
```

Kompleksitas Waktu Asimptotik :

V : Jumlah vertex E : Jumlah edge

- Menandai setiap vertex belum dikunjungi : O(V)
- Menandai vertex awal telah dikunjungi lalu masukkan ke queue : O(1)
- Keluarkan vertex dari queue kemudian cetak : O(V)
- Kunjungi setiap vertex yang belum dikunjungi kemudian masukkan ke queue : O(E)

Maka :

T(n) = O(V) + O(1) + O(V) + O(E)

= O(maks(V,1)) + O(V) + O(E)

= O(V) + O(V) + O(E)

= O(maks(V,V)) + O(E))

= O(V) + O(E)

= O(V+E)

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!

```
/* Bunga Azizha N - 140810180016
Kelas B - Program DFS*/

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

struct Graph {
  int vertex;
  list<int>* edge;
};

void makeGraph(Graph& G, int vertex);
void addEdge(Graph& G, int i, int j);
void traversal(Graph G);
void DFSUtil(Graph G, int v, bool visited[]);
void DFS(Graph G, int s);

int main() {
  Graph G;
  int n, m, val, edge;
```

```
    cout << "Banyak simpul dalam graph : ";
    cin >> n;
    makeGraph(G, n);

    for (int i = 0; i < n; i++) {
        cout << endl << "Nilai simpul ke-" << i+1 << " : ";
        cin >> val;
        cout << "Banyak edge dari simpul " << val << " : ";
        cin >> m;

        for (int j = 0; j < m; j++) {
            cout << "Simpul edge ke-" << j+1 << " : ";
            cin >> edge;
            addEdge(G, val, edge);
        }
    }

    cout << "\nDFS dimulai dari simpul "
         << *G.edge[0].begin() - 1 << endl;
    DFS(G, *G.edge[0].begin());
}

void makeGraph(Graph& G, int vertex) {
    G.vertex = vertex;
    G.edge = new list<int>[vertex];
}

void addEdge(Graph& G, int i, int j) {
    G.edge[i].push_back(j);
}

void traversal(Graph G) {
    for (int i = 0; i < G.vertex; ++i) {
        cout << "\n vertex "<<i<<"\n head";
        for (auto x : G.edge[i])
            cout << " -> " << x;
        cout << endl;
    }
}

void DFSUtil(Graph G, int v, bool visited[]) {
    visited[v] = true;
    cout << v << " ";
```

```
    for (list<int>::iterator i = G.edge[v].begin(); i != G.edge[v].end(); ++i)
        if(!visited[*i])
            DFSUtil(G, *i, visited);
}

void DFS(Graph G, int s) {
    bool *visited = new bool[G.vertex];
    for (int i=0; i < G.vertex; i++)
        visited[i] = false;

    for (int i=0; i < G.vertex; i++)
        if (visited[i] == false)
            DFSUtil(G, i, visited);
}
```

Kompleksitas Waktu Asimptotik :

V : Jumlah vertex E : Jumlah edge

- Menandai vertex awal telah dikunjungi kemudian cetak : O(1)
- Rekursif untuk semua vertex : T(E/1)
- Tandai semua vertex belum dikunjungi : O(V)
- Rekursif untuk mencetak DFS : T(V/1)

Maka :

T(n) = O(1) + T(V/1)+ O(V) + T(V/1)

= O(1) + O(E) + O(V) + O(V)

= O(maks(1,E)) + O(V) + O(V)

= O(E) + O(V) + O(V)

= O(maks(V,V)) + O(E))

= O(V) + O(E)

= O(V+E)