Weekly Homework 8

Ngày 21 tháng 5 năm 2025

Exercises

1. The file contains an adjacency matrix. Read the file and output the corresponding adjacency list.

```
vector<vector<int>> convertMatrixToList(const string& filename);
```

2. The file contains an adjacency list. Read the file and output the corresponding adjacency matrix.

```
vector<vector<int>> convertListToMatrix(const string& filename);
```

adjacency matrix	adjacency list
9	9
0 0 1 0 0 1 0 0 0	2 2 5
0 0 0 0 0 0 1 0 0	1 6
0 0 0 0 0 0 1 0 0	1 6
0 0 0 0 1 0 0 0 0	1 4
0 0 0 0 0 1 0 0 0	1 5
0 0 0 1 0 0 0 1 0	2 3 7
0 0 0 0 0 0 0 0 0	0
0 0 1 0 0 0 0 0 1	2 2 8
0 0 0 0 0 0 0 0 0	0

Bảng 1: Adjacency matrix and corresponding Adjacency list

3. Implement functions to provide the following information about a given graph:

```
// Directed or Undirected Graph.
bool isDirected(const vector<vector<int>>& adjMatrix);

// The number of vertices.
int countVertices(const vector<vector<int>>& adjMatrix);

// The number of edges.
```

```
int countEdges(const vector<vector<int>>& adjMatrix);

// List of isolated vertices.
vector<int> getIsolatedVertices(const vector<vector<int>>& adjMatrix);

// Undirected Graph.
bool isCompleteGraph(const vector<vector<int>>& adjMatrix);

// Undirected Graph
bool isBipartite(const std::vector<std::vector<int>>& adjMatrix);

// Undirected Graph
bool isCompleteBipartite(const vector<vector<int>>& adjMatrix);
```

4. Generate a base undirected graph from a given directed graph.

```
vector<vector<int>> convertToUndirectedGraph(const vector<vector<int>>& adjMatrix);
```

5. Generate a complement graph from a given undirected graph and output its adjacency matrix (*undirected graph).

```
vector<vector<int>> getComplementGraph(const vector<vector<int>>& adjMatrix);
```

6. Determine the Euler cycle from a given graph using Hierholzer's Algorithm.

```
vector<int> findEulerCycle(const vector<vector<int>>& adjMatrix);
```

7. Find the spanning tree of a given graph using(*undirected graph):

```
vector<vector<int>> dfsSpanningTree(const vector<vector<int>>& adjMatrix, int start);
vector<vector<int>> bfsSpanningTree(const vector<vector<int>>& adjMatrix, int start);
```

8. Verify the connection between two vertices of a given graph.

```
bool isConnected(int u, int v, const vector<vector<int>>& adjMatrix);
```

9. Find the shortest path between two vertices of a given graph using (*Weighted Graph):

```
vector<int> dijkstra(int start, int end, const vector<vector<int>>& adjMatrix);
vector<int> bellmanFord(int start, int end, const vector<vector<int>>& adjMatrix);
```

1 Submission Rules

Students must adhere to the following submission guidelines:

1. The submission must be in a **compressed zip file** named **MSSV.zip**, containing:

   - **Note:** All functions should be implemented in the `main.cpp` file.
     Please ensure that the **function signatures remain exactly the same** as provided above.
     Important: Do *not* include the `main` function in the `main.cpp` file.
   - A `report.pdf` file describing the approach used in each solution. The image of GitHub home page to verify code is pushed on GitHub
   - Don't use $< bits/stdc++.h >$ library