Homework 8 - Discrete Mathematics

Nguyen Tien Duc - ITITIU18029

December 22, 2019

Part I
Finding shortest path

Dijkstra Algorithm

```cpp
/* Dijkstra.cpp */
#include <string>
#include <vector>
using namespace std;
#define INF 999

void findPathDijkstra(vector<int> parent, int src, string& path) {
  if (parent[src] == -1) return;
  findPathDijkstra(parent, parent[src], path);
  path += "->" + to_string(src + 1);
}

vector<string> Dijsktra(vector<vector<int>>& graph, int start, int end) {
  vector<string> result(2);
  start--;
  end--;
  size_t V = graph.size();
  vector<int> D(V, INF);
  vector<bool> T(V, true);
  vector<int> P(V);
  D[start] = 0;
  P[start] = -1;

  while (T[end]) {
    int min_length = INF;
    int min_index = start;
    for (size_t i = 0; i < V; i++) {
      int weight = D[i] + graph[min_index][i];
      if (T[i] && weight < min_length) {
        min_length = D[i];
        min_index = i;
      }
    }
    T[min_index] = false;

    for (size_t i = 0; i < V; i++) {
      if (T[i] && graph[min_index][i] > 0) {
```

```
        int weight = D[min_index] + graph[min_index][i];
        if (weight < D[i]) {
          P[i] = min_index;
          D[i] = weight;
        }
      }
    }
  }
  result[0] = to_string(start + 1);
  findPathDijkstra(P, end, result[0]);
  result[1] = to_string(D[end]);
  return result;
}
```

Floyd Algorithm

```
/* Floyd.cpp */
#include <string>
#include <vector>
using namespace std;
#define INF 999

void findPathFloyd(vector<vector<int>> parents, int start, int end, string& path) {
  if (start == end)
    path += "->" + to_string(start + 1);
  else if (parents[start][end] == -1)
    path += to_string(start + 1) + "->" + to_string(end + 1);
  else {
    findPathFloyd(parents, start, parents[start][end], path);
    path += "->" + to_string(end + 1);
  }
}

vector<string> Floyd(vector<vector<int>>& graph, int start, int end) {
  vector<string> result(2);
  vector<vector<int>> D;
  vector<vector<int>> P;
  start--;
  end--;
  size_t V = graph.size();
  for (size_t i = 0; i < V; i++) {
    vector<int> tmp;
    vector<int> tmp2;
    for (size_t j = 0; j < V; j++) {
      tmp2.push_back(i);
      if (graph[i][j] == 0) {
        graph[i][j] = INF;
        if (i != j) tmp2[j] = -1;
      }
      tmp.push_back(graph[i][j]);
    }
    D.push_back(tmp);
    P.push_back(tmp2);
  }

  for (size_t k = 0; k < V; k++) {
    for (size_t i = 0; i < V; i++) {
      for (size_t j = 0; j < V; j++) {
```

```
        if (D[i][k] + D[k][j] < D[i][j]) {
          D[i][j] = D[i][k] + D[k][j];
          P[i][j] = P[k][j];
        }
      }
    }
  }

  findPathFloyd(P, start, end, result[0]);
  result[1] = to_string(D[start][end]);
  return result;
}
```

Main Function

```
#include <fstream>
#include <iomanip>
#include <iostream>
#include <vector>
#include "Dijkstra.cpp"
#include "Floyd.cpp"
using namespace std;

int main() {
  istream& input = cin;
  /*  Change the above line to:
        ifstream file("/path/to/data/file");
        istream& input = file;
      to input big adjacency matrices from file. */

  int verticeNum, start, end;
  cout << "Input number of vertices: ";
  input >> verticeNum;

  vector<vector<int>> adjacencyMatrix;
  cout << endl << "Input adjacency matrix" << endl;
  for (int i = 0; i < verticeNum; i++) {
    vector<int> tmp;
    for (int j = 0; j < verticeNum; j++) {
      int tmp_length;
      cout << "Input row " << i + 1 << " column " << j + 1 << ": ";
      input >> tmp_length;
      tmp.push_back(tmp_length);
    }
    adjacencyMatrix.push_back(tmp);
  }

  cout << "Your input matrix is: " << endl;
  for (int i = 0; i < verticeNum; i++) {
    for (int j = 0; j < verticeNum; j++)
      cout << setw(2) << adjacencyMatrix[i][j] << " ";
    cout << endl;
  }

  cout << "Input starting vertice: ";
  input >> start;
  cout << "Input destination vertice: ";
  input >> end;
```

```
44
45    vector<string> result = Dijsktra(adjacencyMatrix, start, end);
46    // Or change the above line to this for Floyd algorithm
47    // vector<string> result = Floyd(adjacencyMatrix, start, end);
48
49    cout << endl;
50    cout << "Shortest path is: " << result[0] << endl;
51    cout << "Shortest path length: " << result[1] << endl;
52    return 0;
53 }
```

Demo

As the requirements are the same for both algorithms, so both algorithms take the same inputs and also output the same thing.
Therefore, I only took 1 demo picture.

```
Input row 2 column 2: 0
Input row 2 column 3: 0
Input row 2 column 4: 0
Input row 2 column 5: 5
Input row 2 column 6: 0
Input row 3 column 1: 2
Input row 3 column 2: 0
Input row 3 column 3: 0
Input row 3 column 4: 5
Input row 3 column 5: 0
Input row 3 column 6: 12
Input row 4 column 1: 4
Input row 4 column 2: 7
Input row 4 column 3: 0
Input row 4 column 4: 0
Input row 4 column 5: 3
Input row 4 column 6: 6
Input row 5 column 1: 0
Input row 5 column 2: 0
Input row 5 column 3: 0
Input row 5 column 4: 3
Input row 5 column 5: 0
Input row 5 column 6: 2
Input row 6 column 1: 0
Input row 6 column 2: 0
Input row 6 column 3: 12
Input row 6 column 4: 0
Input row 6 column 5: 2
Input row 6 column 6: 0
Your input matrix is:
 0 13  2  0  0  0
13  0  0  0  5  0
 2  0  0  5  0 12
 4  7  0  0  3  6
 0  0  0  3  0  2
 0  0 12  0  2  0
Input starting vertice: 1
Input destination vertice: 6
Shortest path is: 1->3->4->5->6
Shortest path length: 12
```

Exersize 4 / page 655

I solved the exercise using both algorithms.
Although there is a slight difference in the path, the shortest path length is still the same.

Dijkstra's Result

```
 0  2  4  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 2  0  3  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 4  3  0  0  2  2  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 1  0  0  0  0  5  4  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  1  2  0  0  0  0  3  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  2  5  0  0  3  3  2  4  0  0  0  0  0  0  0  0  0  0
 0  0  0  4  0  3  0  0  0  0  2  0  0  0  0  0  0  0  0  0
 0  0  0  0  3  3  0  0  0  0  0  1  0  0  8  0  0  0  0  0
 0  0  0  0  0  2  0  0  0  3  0  3  2  0  0  0  0  0  0  0
 0  0  0  0  0  4  0  0  3  0  6  0  6  3  0  0  0  0  0  0
 0  0  0  0  0  0  2  0  0  6  0  0  0  4  0  0  0  2  0  0
 0  0  0  0  0  0  0  1  3  0  0  0  3  0  6  0  0  0  0  0
 0  0  0  0  0  0  0  0  2  6  0  3  0  5  4  2  0  0  0  0
 0  0  0  0  0  0  0  0  0  3  4  0  5  0  0  0  2  1  0  0
 0  0  0  0  0  0  0  8  0  0  0  6  4  0  0  2  0  0  6  0
 0  0  0  0  0  0  0  0  0  0  0  0  2  0  2  0  1  0  2  1
 0  0  0  0  0  0  0  0  0  0  0  0  0  2  0  1  0  8  0  3
 0  0  0  0  0  0  0  0  0  0  2  0  0  1  0  0  8  0  0  5
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  6  2  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  3  5  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  2  8
Input starting vertice: Input destination vertice:
Shortest path is: 1->4->6->9->13->16->19->21
Shortest path length: 16
```

Floyd's Result

```
 0  2  4  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 2  0  3  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 4  3  0  0  2  2  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 1  0  0  0  0  5  4  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  1  2  0  0  0  0  3  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  2  5  0  0  3  3  2  4  0  0  0  0  0  0  0  0  0  0
 0  0  0  4  0  3  0  0  0  0  2  0  0  0  0  0  0  0  0  0
 0  0  0  0  3  3  0  0  0  0  0  1  0  0  8  0  0  0  0  0
 0  0  0  0  0  2  0  0  0  3  0  3  2  0  0  0  0  0  0  0
 0  0  0  0  0  4  0  0  3  0  6  0  6  3  0  0  0  0  0  0
 0  0  0  0  0  0  2  0  0  6  0  0  0  4  0  0  0  2  0  0
 0  0  0  0  0  0  0  1  3  0  0  0  3  0  6  0  0  0  0  0
 0  0  0  0  0  0  0  0  2  6  0  3  0  5  4  2  0  0  0  0
 0  0  0  0  0  0  0  0  0  3  4  0  5  0  0  0  2  1  0  0
 0  0  0  0  0  0  0  8  0  0  0  6  4  0  0  2  0  0  6  0
 0  0  0  0  0  0  0  0  0  0  0  0  2  0  2  0  1  0  2  1
 0  0  0  0  0  0  0  0  0  0  0  0  0  2  0  1  0  8  0  3
 0  0  0  0  0  0  0  0  0  0  2  0  0  1  0  0  8  0  0  5
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  6  2  0  0  0  0  2
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  3  5  0  0  8
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  2  8  0
Input starting vertice: Input destination vertice:
Shortest path is: ->1->3->6->9->13->16->19->21
Shortest path length: 16
```

Part II
Minimum spanning tree

Kruskal Algorithm

Code

```cpp
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
using namespace std;

#define edge pair<int,int>

class Graph {
private:
    vector<pair<int, edge>> G; // graph
    vector<pair<int, edge>> T; // mst
    int *parent;
    int V; // number of vertices/nodes in graph
public:
    Graph(int V);
    void AddWeightedEdge(int u, int v, int w);
```

```
    int find_set(int i);
    void union_set(int u, int v);
    void kruskal();
    void print();
};
Graph::Graph(istream& data) {
    parent = new int[V];
    for (int i = 0; i < V; i++)
        parent[i] = i;

    G.clear();
    T.clear();
}
void Graph::AddWeightedEdge(int u, int v, int w) {
    G.push_back(make_pair(w, edge(u, v)));
}
int Graph::find_set(int i) {
    // If i is the parent of itself
    if (i == parent[i])
        return i;
    else
        // Else if i is not the parent of itself
        // Then i is not the representative of his set,
        // so we recursively call Find on its parent
        return find_set(parent[i]);
}

void Graph::union_set(int u, int v) {
    parent[u] = parent[v];
}
void Graph::kruskal() {
    int i, uRep, vRep;
    sort(G.begin(), G.end()); // increasing weight
    for (i = 0; i < G.size(); i++) {
        uRep = find_set(G[i].second.first);
        vRep = find_set(G[i].second.second);
        if (uRep != vRep) {
            T.push_back(G[i]); // add to tree
            union_set(uRep, vRep);
        }
    }
}
void Graph::print() {
    cout << "Edge :" << " Weight" << endl;
    for (int i = 0; i < T.size(); i++) {
        cout << T[i].second.first << " - " << T[i].second.second << " : "
                << T[i].first;
        cout << endl;
    }
}
int main() {
    Graph g(6);
    g.AddWeightedEdge(0, 1, 4);
    g.AddWeightedEdge(0, 2, 4);
    g.AddWeightedEdge(1, 2, 2);
    g.AddWeightedEdge(1, 0, 4);
    g.AddWeightedEdge(2, 0, 4);
    g.AddWeightedEdge(2, 1, 2);
    g.AddWeightedEdge(2, 3, 3);
```

```
    g.AddWeightedEdge(2, 5, 2);
    g.AddWeightedEdge(2, 4, 4);
    g.AddWeightedEdge(3, 2, 3);
    g.AddWeightedEdge(3, 4, 3);
    g.AddWeightedEdge(4, 2, 4);
    g.AddWeightedEdge(4, 3, 3);
    g.AddWeightedEdge(5, 2, 2);
    g.AddWeightedEdge(5, 4, 3);
    g.kruskal();
    g.print();
    return 0;
}
```

Demo