CSC 116
Fundamentals of Programming with Engineering Applications II
Assignment 2

Note 1 **This assignment is to be done individually**
Note 2 You can discuss the assignment with others, but copying code is prohibited.

A note on Academic Integrity and Plagiarim

Please review the following documents:

- Standards for Professional Behaviour, Faculty of Engineering: `https://www.uvic.ca/engineering/assets/docs/professional-behaviour.pdf`
- Policies Academic Integrity, UVic: `https://www.uvic.ca/students/academics/academic-integrity/`
- Uvic's Calendar section on Plagirism: `https://www.uvic.ca/calendar/undergrad/index.php#/policy/Sk_0xsM_V`

Note specifically:

> Plagiarism
> Single or multiple instances of inadequate attribution of sources should result in a failing grade for the work. A largely or fully plagiarized piece of work should result in a grade of F for the course.

A program you submit will be considered a **piece of work**. You are responsible for your own submission, but you could also be responsible if somebody plagiarizes your submission.

1 Objectives

After completing this assignment, you will have experience with:

- Using vectors to create matrices
- Basic graph manipulation

2 Your task, should you choose to accept it

Suppose that you are a telecommunications engineer and you are told you have to connect with fiber-optic cable several cities together. The goal is to send information from one city to any other, even if such communication has to pass through several other cities. You are given a set of giving potential connections between two cities. Each of these connections is a pair of cities and has a cost (the cost of laying the fiber-optic between the pair of cities). If you are not given a connection between two cities, it means the connection between these two cities is not possible.

Your task is to connect all the cities at the minimum cost.

3 Representing the Problem as a Graph

This is a typical graph theory problem. The cities and the connections between them can be represented as a undirected graph. A graph is a set of vertices and a set of edges. An edge is a pair (a, b) where a, b are vertices of the graph.

For our problem, the cities represent vertices, and their potential connections are edges. Each edge has an associated cost (the "weight" of the edge).

3.1 Representing the graph in the computer

One easy way to represent a graph in a program is by using a matrix. A graph of N vertices can be represented with a symmetric NxN matrix. Each element of the matrix will correspond to the cost of laying the fiber-optic between the two cities. If it is not possible to lay fiber-optic between two cities then the corresponding element of the matrix will contain the infinite value.

Let us assume we have 10 cities. We represent these cities and their connections as a graph, as depicted in Figure 1. Each city is labelled from letters A to J and it is represented by an edge in the graph. The potential of laying fiber-optic between two cities is denoted by an edge, and its weight is its cost. For example, the cost of connecting A and B is 4. Note that the direction in this graph does not matter.

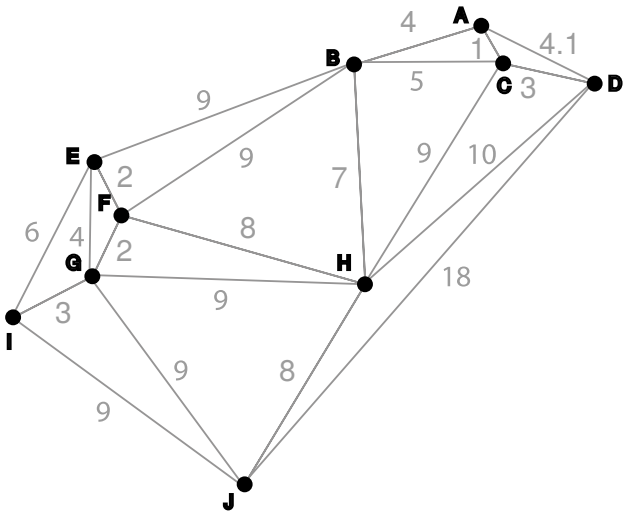


Figure 1: Graph represention of the Cities and their potential connections

The cities and potential connections in our example would be presented as shown in Table 1. Note that INF represents infinite.

This representation is hard to read. There is an alternative representation called *adjacency list* that uses linked-list to represented connections between vertices. In this representation, a graph is presented as a list of N lists, each corresponding to a vertex in the graph. The list of a vertex is the list of all reachable vertices (and their weight) from such vertex. The adjacency list of the graph is shown in Table 2.

3.2 Minimum Spanning Tree of a Graph

Once we represent the cities as a graph, we need to find the graph's minimum spanning tree. Given a graph G, its minimum spanning tree is the subgraph of G such that it connects all the vertices together with the minimal total weight for its edges. In our problem the weight of the edges is their cost.

	A	B	C	D	E	F	G	H	I	J
A	INF	4	1	4	INF	INF	INF	INF	INF	INF
B	4	INF	5	INF	9	9	INF	7	INF	INF
C	1	5	INF	3	INF	INF	INF	9	INF	INF
D	4	INF	3	INF	INF	INF	INF	10	INF	18
E	INF	9	INF	INF	INF	2	4	INF	6	INF
F	INF	9	INF	INF	2	INF	2	8	INF	INF
J	INF	INF	INF	INF	4	2	INF	9	3	9
H	INF	7	9	10	INF	8	9	INF	INF	8
I	INF	INF	INF	INF	6	INF	3	INF	INF	9
J	INF	INF	INF	18	INF	INF	9	8	9	INF

Table 1: Matrix representation of the cities in Figure 1

A	→ (B,4)	(C,1)	(D,4)			
B	→ (A,4)	(C,5)	(E,9)	(F,9)	(H,7)	
C	→ (A,1)	(B,5)	(D,3)	(H,9)		
D	→ (A,4)	(C,3)	(H,10)	(J,18)		
E	→ (B,9)	(F,2)	(G,4)	(I,6)		
F	→ (B,9)	(E,2)	(G,2)	(H,8)		
G	→ (E,4)	(F,2)	(H,9)	(I,3)	(J,9)	
H	→ (B,7)	(C,9)	(D,10)	(F,8)	(G,9)	(J,8)
I	→ (E,6)	(G,3)	(J,9)			
J	→ (D,18)	(G,9)	(H,8)	(I,9)		

Table 2: Adjancency List presentation of our example in Figure 1.

The minimum spanning tree of the cities connections will be the cheapest way we can connect all the cities. The minimum spanning tree of the graph in the example above is depicted in in Figure 2.

3.3 Cost of laying fiber

The cost of laying fiber to connect the cities is the sum of the edges of a graph. In the original graph, the cost of lying fiber is 139. The cost of the minimum spanning tree is 38. Hence 38 is the minimum cost to lay fiber between all the cities in our example.

4 Computing a Minimum Spanning Tree with the Prim-Járnik algorithm

Several algorithms exist to compute minimum spanning trees. An algorithms course (such as CSC 225 or CSC 226) covers the theoretical aspects of the various algorithms. For this assignment, we are mostly concerned with computing a result rather than a theoretical analysis. We will use the Prim-Járnik algorithm to compute minimum spanning trees.

Wikipedia describes the Prim-Járnik Algorithm as "a greedy algorithm that finds a minimum spanning tree for a weighted undirected graph." It is fairly simple:

Step 0 Let G be the input graph, represented by an adjacency matrix. Let n be the number of vertices in G.

Step 1 Initialize a matrix R as the result graph, with every entry set to ∞. Create a set S to store the index of each vertex as it is added to the result graph. Add the first vertex to S (we can use any vertex, but let us keep it simple).

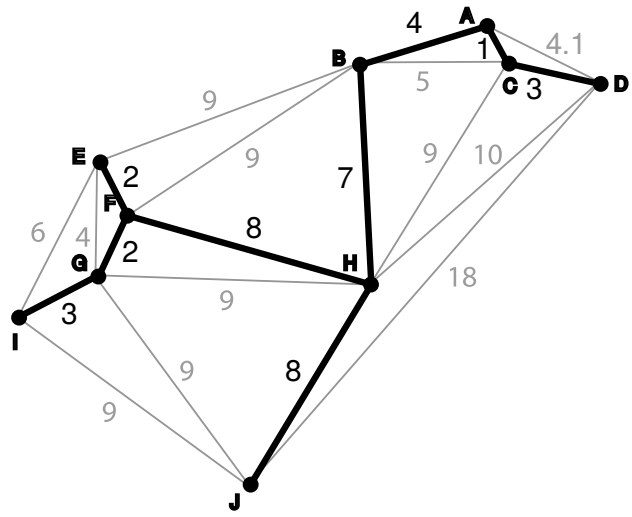


Figure 2: Minimum spanning tree of graph in Figure 1.

Step 2 Repeat until all vertices are the result graph and the size of S is equal to n (this step will add one vertex at each iteration):

Step 2.1 In the input graph find the minimum-weight edge between a vertex that is already in S and a vertex that is not S.

Step 2.2 Add the chosen edge to R.

Step 2.3 Add both endpoints of the chosen edge to S (one endpoint will already be in S).

Step 3 The matrix for the result graph will now correspond to a minimum spanning tree of the input graph.

At the end of this process, the result graph contains the minimum spanning tree of the original graph.

4.1 Example

Let us see how the Prim-Járnik algorithm works. Assume that we have a graph of 4 vertices depicted in Figure 3.

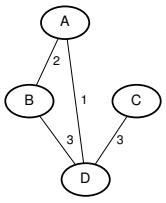


Figure 3: Example input graph to the Prim-Járnik algorithm.

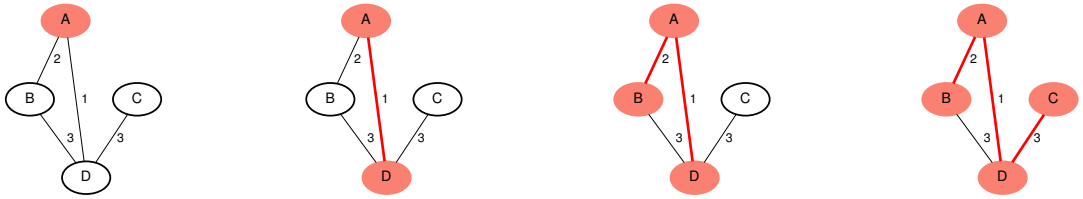
Figure 4 shows the output graph after each iteration:

a) We select vertex A as the starting one (Step 1).

b) After the first iteration of Step 2 the edge (A, D) with weight 1 is the smallest that links the vertices in the current result subgraph (containing only vertex A, from step a) with the rest of the graph. It is added to the result graph.

c) In the second iteration the edge (A, B) with weight 2 is the smallest that links the vertices in the result graph (as in step b) with the rest of the graph. It is added to the result graph.

d) In the third and last iteration the edge (C, D) with weight 3 is the smallest that links the vertices in the result graph (as in step c) with the rest of the graph. It is added to the result graph. The algorithm stops because all vertices of the original graph are in the result graph.



(a) After selecting starting vertex.

(b) After 1st iteration.

(c) After 2nd iteration.

(d) After 3rd and final iteration.

Figure 4: Value of result graph (the minimum spanning tree) after each step of Prim's algorithm.

4.2 Further resources

There are lots of resource for the Prim-Járnik algorithm:

- The Wikipedia has a good description of the Minimum Spanning Tree: `https://en.wikipedia.org/wiki/Minimum_spanning_tree`
- A visualization of the Prim-Járnik algorithm that will help you understand how it works: `http://www.cs.usfca.edu/~galles/visualization/Prim.html`

5 Code provided

Download the source code from conneX. Run `cmake` as usual and load into your favorite environment.

5.1 Specification

1. The program `minimum` takes no command line parameters. It reads a graph from standard input and outputs its results to standard output.
2. Its input is a graph.
3. It outputs the graph in both matrix and adjancy list forms.
4. **It computes the minimum spanning tree of the input graph.**
5. It outputs the minimum spanning tree in adjancy list form.
6. **It computes the overall cost of the original graph and the minimum spanning tree.**
7. It outputs these values.

Your job is to implement items **4** and **6**

5.2 Input

The input to the program is a graph. The first number in the output is the number of vertices N. Vertices are labelled from 0 to N-1 (they correspond to vertices with labels A, B, etc.). The next lines correspond each to an edge: source edge, destination edge, weight. The example in Figure 3 would be represented as follows:

```
4
0 1 2.0
0 3 1.0
1 3 3.0
2 3 3.0
```

Note that you do not have to implement reading the input. It is already done for you.

5.3 Output

The expected output for this example is:

```
Input matrix. Size : 4 x 4
 -  :   2 :  -  :   1 :
  2 :  -  :  -  :   3 :
 -  :  -  :  -  :   3 :
  1 :   3 :   3 :  -  :

Input Adjacency list. Size : 4
From City: A -> (B,2) (D,1)
From City: B -> (A,2) (D,3)
From City: C -> (D,3)
From City: D -> (A,1) (B,3) (C,3)

Minimum Spanning Tree. Size : 4
From City: A -> (B,2) (D,1)
From City: B -> (A,2)
From City: C -> (D,3)
From City: D -> (A,1) (C,3)

Cost of original matrix: 9
Cost of minimum spanning tree: 6
```

Note that you do not have to implement creating the output. It is already done for you.

5.4 `matrix_type`

A graph is represented using a symmetric matrix. We have created a type with the following declaration (see `matrix.hpp`). This matrix is always square, and you should make sure it is always symmetric.

```
typedef std::vector <
    std::vector <double>
    > matrix_type;
```

5.5 `INFINITY`

We have declared a constant called `INFINITY` that represents a lack of edge between two vertices. It is declared as follows:

```
const double INFINITY = std::numeric_limits<double>::infinity();
```

If an element of the matrix is equal to `INFINITY` then no edge exists between the two corresponding vertices.

5.6 `Print_Matrix` and `Print_Adjacency`

These two functions (declared in `main.cpp`) can be used during debugging to inspect the contents of your minimum spanning tree as you build it. See `main.cpp` for an example of how to use them.

6 What you must do

There are two functions you must implement. Both are found in the file `minimum.cpp`:

```
matrix_type Minimum_Spanning_Tree(const matrix_type & matrix)
{
...
}

double Calculate_Cost(const matrix_type &matrix)
{
...
}
```

1. `Minimum_Spanning_Tree` takes as a parameter a matrix corresponding to its input graph and computes its minimum spanning tree, which becomes its return value..
2. `Calculate_Cost` takes a matrix representing a graph and computes its overall cost: the sum of the cost of all the edges. You should ignore any element of the matrix whose value is `INFINITY`. Remember also that the matrix is symmetric.

You must only modify the file `minimum.cpp`.

7 Tests provided

This assignment has 10 tests. The tests are

1. A graph with a single vertex.
2. A graph with two vertices.
3. A graph with three vertices.
4. The graph in Figure 3.
5. A graph with five vertices.
6. A graph with three vertices and negative costs.
7. A graph with seven vertices.
8. A graph with eight vertices.
9. A graph with nine vertices.
10. The graph in Figure 1.

8 Evaluation

Solutions should be:

1. Correct. They should pass all the tests we have provided.
2. In good style, including indentation and line breaks

What to submit

- Using conneX, submit your version of the file `minimum.cpp`. **Do not submit any other files or alter the filename of `minimum.cpp`**.
- You should assume that your submission was received correctly only if both of the following two conditions hold.
  - You receive a confirmation email from conneX.
  - You can download and view your submission, and it contains the correct data.
- In the event that your submission is not received by the marker, you will need to demonstrate that both of the above conditions were met if you want to argue that there was a submission error.