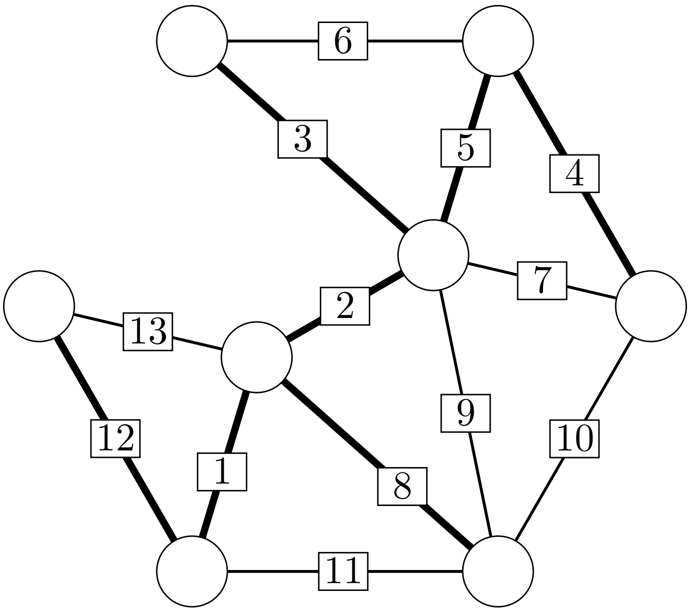
CSC 226 FALL 2014
ALGORITHMS AND DATA STRUCTURES II
ASSIGNMENT 1 - PROGRAMMING
UNIVERSITY OF VICTORIA

1 Programming Assignment

The assignment is to implement an algorithm to find a minimum weight spanning tree of an edge-weighted graph. A Java template has been provided containing an empty method `MWST`, which takes a single argument consisting of a weighted adjacency matrix for an edge-weighted graph G. The expected behavior of the method is as follows:

Input: A $n \times n$ array G representing an edge-weighted graph.
Output: An integer value corresponding to the total weight of a minimum weight spanning tree of G.

A correct implementation of the `MWST` function will use one of the three algorithms for minimum weight spanning trees (Barůvka's algorithm, the Jarník-Prim algorithm or Kruskal's algorithm) to find a minimum weight spanning tree, then sum the weights of each selected edge to produce the return value. For example, consider the edge-weighted graph below.



The darkened edges of the graph form a minimum weight spanning tree, and the total weight is 35.

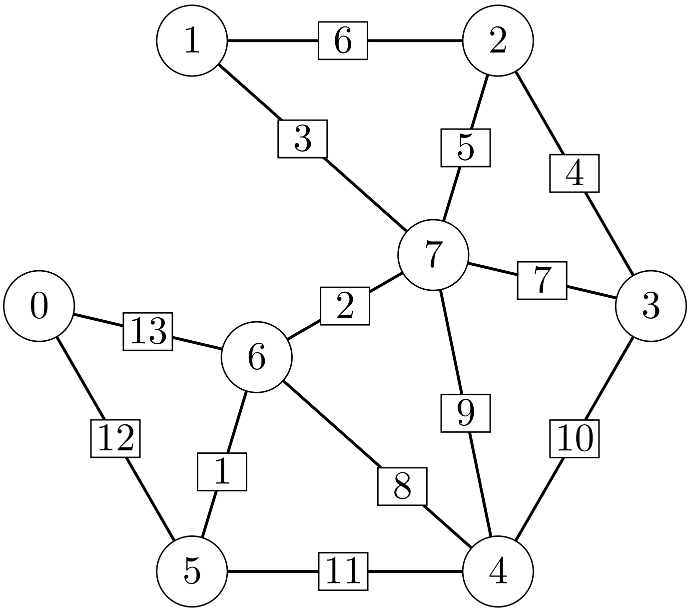
You must use the provided Java template as the basis of your submission, and put your implmentation inside the `MWST` method in the template. You may not change the name, return type or parameters of the `MWST` method. You may add additional methods as needed. The `main` method in the template contains code to help you test your implementation by entering test data or reading it from a file. You may modify the `main` method to help with testing, but only the contents of the `MWST` method (and any methods you have added) will be marked, since the `main` function will be deleted before marking begins. Please read through the comments in the template file before starting.

2 Input Format

The testing code in the `main` function of the template reads a sequence of graphs in a weighted adjacency matrix format and uses the `MWST` function to compute the weight of a minimum weight spanning tree for each. A weighted adjacency matrix A for an edge-weighted graph G on n vertices is an $n \times n$ matrix where entry (i, j) gives the weight of the edge between vertices i and j (or 0 if no edge exists). For example, the matrix

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 12 & 13 & 0 \\ 0 & 0 & 6 & 0 & 0 & 0 & 0 & 3 \\ 0 & 6 & 0 & 4 & 0 & 0 & 0 & 5 \\ 0 & 0 & 4 & 0 & 10 & 0 & 0 & 7 \\ 0 & 0 & 0 & 10 & 0 & 11 & 8 & 9 \\ 12 & 0 & 0 & 0 & 11 & 0 & 1 & 0 \\ 13 & 0 & 0 & 0 & 8 & 1 & 0 & 2 \\ 0 & 3 & 5 & 7 & 9 & 0 & 2 & 0 \end{bmatrix}$$

corresponds to the edge-weighted graph below. Note that the weighted adjacency matrix for an undirected graph is always symmetric.



The input format used by the testing code in `main` consists of the number of vertices n followed by the $n \times n$ weighted adjacency matrix. The graph above would be specified as follows:

```
8
 0   0   0   0   0  12  13   0
 0   0   6   0   0   0   0   3
 0   6   0   4   0   0   0   5
 0   0   4   0  10   0   0   7
 0   0   0  10   0  11   8   9
12   0   0   0  11   0   1   0
13   0   0   0   8   1   0   2
 0   3   5   7   9   0   2   0
```

3 Test Datasets

A collection of randomly generated edge-weighted graphs has been uploaded to conneX. Your assignment will be tested on graphs similar but not identical to the uploaded graphs. You are encouraged to create your own test inputs to ensure that your implementation functions correctly in all cases.

4 Sample Run

The output of a model solution on the graph above is given in the listing below. Console input is shown in blue.

```
Reading input values from stdin.
Reading graph 1
8
 0   0   0   0   0  12  13   0
 0   0   6   0   0   0   0   3
 0   6   0   4   0   0   0   5
 0   0   4   0  10   0   0   7
 0   0   0  10   0  11   8   9
12   0   0   0  11   0   1   0
13   0   0   0   8   1   0   2
 0   3   5   7   9   0   2   0
Graph 1: Total weight is 35
Processed 1 graph.
Average Time (seconds): 0.00
```

5 Evaluation Criteria

The programming assignment will be marked out of 50, based on a combination of automated testing and human inspection. To receive full marks, the running time of the implemented algorithm should be at most $O(n^2 + m\log(m))$ on a graph G with n vertices and m edges.

Score (/50)	Description
$0 - 5$	Submission does not compile or does not conform to the provided template.
$5 - 25$	The implemented algorithm is not one of the three MWST algorithms given in class or is substantially inaccurate on the tested inputs.
$26 - 40$	The implemented algorithm is accurate on all tested inputs.
$41 - 50$	The implemented algorithm is accurate on all tested inputs and has a $O(n^2 + m\log(m))$ running time.

To be properly tested, every submission must compile correctly as submitted, and must be based on the provided template. You may only submit one source file. **If your submission does not compile for any reason (even trivial mistakes like typos), or was not based on the template, it will receive at most 5 out of 50**. The best way to make sure your submission is correct is to download it from conneX after submitting and test it. You are not permitted to revise your submission after the due date, and late submissions will not be accepted, so you should ensure that you have submitted the correct version of your code before the due date. conneX will allow you to change your submission before the due date if you notice a mistake. After submitting your assignment, conneX will automatically send you a confirmation email. If you do not receive such an email, your submission was not received. If you have problems with the submission process, send an email to the instructor **before** the due date.