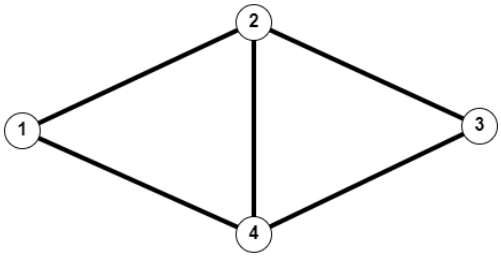
CSE 102
Introduction to Analysis of Algorithms
Programming Project

In this sole programming project for CSE 102, you will write a program that determines a minimum weight spanning tree in a weighted connected graph. Begin by reading the handout on Graph Theory posted on the class webpage, especially if you did not take CSE 101 (or CMPS 101) from me.

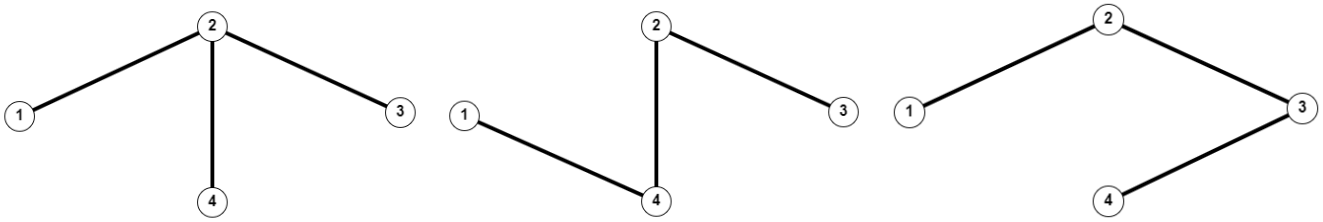
A *weighted graph* is a triple (V, E, w) where $G = (V, E)$ is an (undirected) graph and $w: E \to \mathbb{R}$ is a weight function on edges. A *spanning subgraph* of a graph G is a subgraph H such that $V(H) = V(G)$, i.e. H includes all vertices of G. A *spanning tree* in G is a spanning subgraph that is also a tree. The *weight* of a spanning tree T in a weighted graph G is the sum of the weights of all edges in T.

$$w(T) = \sum_{e \in E(T)} w(e)$$

The following graph contains 8 distinct spanning trees.

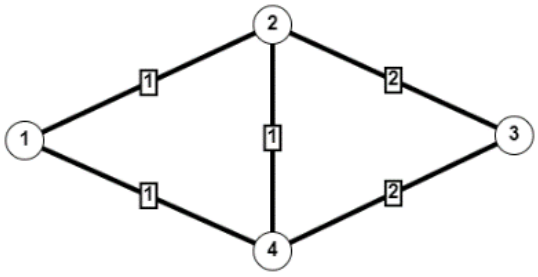


Here are three of them.



Exercise
Draw the other 5 spanning trees. Prove that a graph contains a spanning tree if and only if it is connected.

A *minimum weight spanning tree* (MWST) in a weighted connected graph G, is a spanning tree whose weight is minimum amongst all spanning trees in G. If we alter the preceding example by assigning the following edge weights, then the first two spanning trees are of minimum weight, while the third is not.



Exercise
Determine the weights of all 8 spanning trees in the above example, and observe that there are 6 minimum weight spanning trees (weight $= 4$), and 2 not of minimum weight (weight $= 5$, which happens to be maximum.)

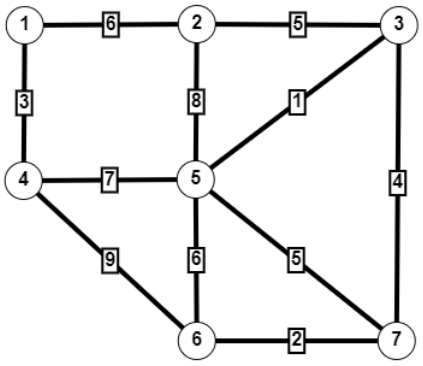
To see more on minimum weight spanning trees, view the following webcast and notes from CMPS 101 Fall 2019.

> Direct Link on Yuja: CSE 101 Fall 2019
> See lecture 28.2 from 21:40 to the end
> Lecture Notes: https://classes.soe.ucsc.edu/cse101/Fall19/Notes/12-4-19.pdf
> See pages 2-10

The *MWST problem* is, given a connected weighted graph G, to determine a MWST in G. There are a number of efficient algorithms that solve this problem. Two of them are covered at length in section 23.2 our text (CLRS 3rd edition), namely the famous algorithms of Kruskal and Prim. (A third, which I refer to as Dual-Kruskal, is discussed in the above webcast.) Supporting data structures for these algorithms are covered in sections 6.1-6.5 (binary heaps and priority queues), and in sections 21.1-21.4 (disjoint sets) of CLRS.

Kruskal's algorithm begins with an empty set F of edges and repeats one simple step: amongst all edges whose addition to F would not create a cycle, add one of mimimum weight to F. We continue until it is no longer possible to do so, i.e. when all edges in $E(G) - F$ would, if added to F, create a cycle with the edges already in F. At that point we have a subset $F \subseteq E(G)$ that is maximal with respect to the property of being acyclic, which implies that $T = (V(G),\ F)$ is a spanning tree in G (see the "treeness" theorem in the handout on Graph Theory.) It's proved in the same handout that necessarily $|F| = |V(G)| - 1$. Section 23.1 of CLRS establishes that T is in fact a MWST.

We illustrate Kruskal's algorithm on the weighted connected graph below.



Sort the edges of G by increasing weights, then step through the list, marking those edges whose addition will not create a cycle with those previously marked. We obtain

Edge:	$(3,5)$	$(6,7)$	$(1,4)$	$(3,7)$	$(2,3)$	$(5,7)$	$(1,2)$	$(5,6)$	$(4,5)$	$(2,5)$	$(4,6)$
Weight:	1	2	3	4	5	5	6	6	7	8	9

where we have shaded the marked edges in red. Therefore $F = \{(3,5), (6,7), (1,4), (3,7), (2,3), (1,2)\}$, and $T = (\{1,2,3,4,5,6,7\},\ F)$ has weight $w(T) = 21$. Further inspection shows that, although this graph has many spanning trees, this is its only minimum weight spanning tree.

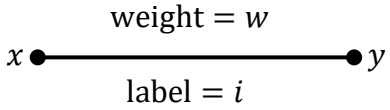
You may use any algorithm you like to solve the MWST problem, even invent one of your own. Also, you are not required to use any particular data structure or ADT in your solution. Some, but not all, of the above mentioned sections in CLRS will be covered in class, but those lectures are not guaranteed to occur before the project due date. I recommend that you do at least a week of self-study, exercises and examples to familiarize yourself with the problem before you start coding.

Your program will be written in one of the following 4 programming languages: C, C++, Python, or Java. All of these languages are installed on the Unix Timeshare. Your project will be evaluated on a similar platform, so testing on the timeshare or an equivalent environment is essential. You will submit a Makefile along with any other source code files for this project. When these files are placed in a directory on the test environment, and after the grader runs `make` at the command line, there must exist an executable file called `MWST` in that directory. This executable will take two command line arguments giving the names two files which store program input and output, and whose format is described below. Upon doing the Unix command

```
$ MWST input_file output_file
```

your program will read `input_file`, solve the MWST problem for the connected weighted graph it encodes, and create `output_file` containing the solution.

The first two lines of an input file will contain single integers n and m, giving the number of vertices and the number of edges in G, respectively. The vertices will be labeled by the set $V = \{1, 2, 3, \ldots, n\}$. The next m lines will each contain two integers (vertex labels) and one real number, separated by spaces, giving the end vertices and weight of an edge in G. The edge labels will be determined by the position of an edge in this list. In general, a triple x, y, w on line $i + 2$ defines an edge with label i (for $1 \leq i \leq m$).



Each of the $m + 2$ lines of the input file will end in a Unix newline character. The following example defines the weighted graph on page 2.

```
7
11
1 2 6
1 4 3
2 3 5
2 5 8
3 5 1
3 7 4
4 5 7
4 6 9
5 6 6
5 7 5
6 7 2
```

An output file will contain n lines. The first $n - 1$ lines will give the edges of an MWST in the following format.

```
label: (end1, end2) weight
```

The last line will give the total weight of the spanning tree. A possible output file for the above example is therefore

```
  5: (3, 5) 1.0
 11: (6, 7) 2.0
  2: (1, 4) 3.0
  6: (3, 7) 4.0
  3: (2, 3) 5.0
  1: (1, 2) 6.0
Total Weight = 21.00
```

Note that a valid output file for this project is not unique, since there may be more than one MWST for a given graph, and the edge list may be given in any order. Also the pair (x, y) denotes an *un-orded* pair which could also be specified as (y, x). Other than these variations, an output file must match the above format exactly. In particular the label will be right justified in a field of width 4, followed by a colon, then a space, then (x, y) or (y, x), then the weight w rounded to one decimal place. The last line will begin at column 1, and the weight will be rounded to 2 decimal places.

A utility for checking the validity of an `input_file`, `output_file` pair will be made available, as well as a random input file generator. It is recommended that you test your program thoroughly on the Unix timeshare or similar environment. Instructions for submitting your program, as well as the exact due date, will be posted on the class webpage. (You'll have approximately 7 weeks to do it.) There will be no extensions on this due date and no late programs will be accepted. **Only source code files may be turned in, and the presence of any binary files in your submission will invalidate your project.** This assignment will be graded only as to its performance. In other words, we will determine whether or not your program produces a valid output file for every input file we test it on. In particular, there will be no unit tests of required functions or ADTs (since there are none.) No re-grades or re-submissions will be considered.

It is recommended that you start early, not by writing code, but by studying the problem and carefully planning your approach.