Problem C: Minimum Spanning Tree

In this assignment you will implement Prim's algorithm for finding a minimum weight spanning tree in a weighted (undirected) graph.

One significant difference between this problem and problem B is that you will be working with undirected graphs. It will therefore be necessary to create a function called addEdge(u, v) (instead of addDirectedEdge) which establishes an undirected edge joining vertices u and v. Another difference is that each edge will posses a weight, which should be stored as type `double`. These weights can be maintained by including a two dimensional array of doubles as part of your Graph struct. If vertex u is adjacent to vertex v, then the u^{th} row v^{th} column this array contains the weight of the corresponding edge. If u is not adjacent to v, then the u^{th} row v^{th} column contains infinity. As usual it is recommended that you avoid index 0, and therefore this array will be of size $(n+1)\times(n+1)$, where n is the number of vertices in the graph.

As in the last two projects vertices will be labeled 1 through n. Prim's algorithm (page 634 of the text) requires that vertices posses the attributes *parent* and *key*. It is recommended that these attributes be included in your Graph as a pair of parallel arrays of types `int` and `double` respectively.

Your project will be tested on connected graphs with no more that 1,000 vertices and 100,000 edges in which no edge weight exceeds 1,000. Therefore an adequate value to represent infinity is 10,000. It is recommended that you #define macros for infinity and nil. It may seem that Prim should have as precondition that the input graph G is connected, since if this is false, G does not contain any spanning tree. However a glance at MST-Prim on page 634 of the text reveals that nothing bad happens when the algorithm is run on a disconnected graph. What does Prim return in this case? In any case, we avoid this question by guaranteeing that the input graph will be connected. Therefore, you don't need to worry about that.

Input and Output Specifications

There will be N test cases which will be given in the first line of the input.
Following this, for each of the N test graphs, first line of input gives the number of vertices in the graph, and the second line gives the number of edges. (Thus it is not necessary to terminate the input by a dummy line 0 0, as was done in problem A or B.) Each of the remaining lines gives the end vertices and the weight of an edge.

So, in a nutshell, your program must be doing the following for each of the test graph:

- Read the input file.
- Store the graph using the list representation.
- Print the adjacency list representation of G to the output file.
- Run Prim on G (using any root r) to determine a minimum weight spanning tree in G, encoded by the parent of each vertex.
- Print a listing of the edges in that tree, as well as it's total weight, to the output file. Include the end vertices and weight of each edge.

Sample Input	Sample Output
1	
10 20 1 2 1.0 1 4 5.0 2 3 3.0 2 4 1.0 2 5 5.0 3 5 5.0 3 6 2.0 3 7 2.0 4 5 5.0 4 8 4.0 4 9 3.0 5 6 5.0 5 8 2.0 5 9 5.0 5 10 1.0 6 7 1.0 6 10 4.0 7 10 4.0 8 9 4.0 9 10 4.0	Graph #1: Adjacency list: 1: 2 4 2: 1 3 4 5 3: 2 5 6 7 4: 1 2 5 8 9 5: 2 3 4 6 8 9 10 6: 3 5 7 10 7: 3 6 10 8: 4 5 9 9: 4 5 8 10 10: 5 6 7 9 A minimum weight spanning tree consists of: Edge (1, 2) of weight 1.0 Edge (2, 3) of weight 3.0 Edge (2, 4) of weight 1.0 Edge (10, 5) of weight 1.0 Edge (3, 6) of weight 2.0 Edge (6, 7) of weight 1.0 Edge (5, 8) of weight 2.0 Edge (4, 9) of weight 3.0 Edge (6, 10) of weight 4.0 Total weight = 18.0