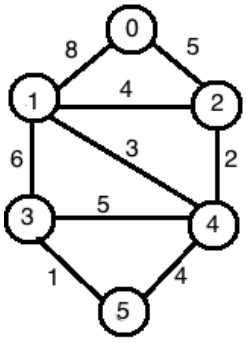
CS 340 Programming Assignment 4:
Prim and Dijkstra

Description: You are to implement **Prim's Algorithm** for finding Minimum Spanning Trees and **Dijkstra's Algorithm** for single-source Shortest Paths for positively weighted graphs adhering to the specifications detailed below.

I/O Specifications: For both problems, you will read your input graph from an input file using the following adjacency list representation: Each line of the file begins with a vertex number. Following the vertex number are pairs of numbers, x_{ij} is the jth neighbor of vertex i (vertex labels are 0 through n), and wij is the weight of that edge.

0 x_{01} w_{01} x_{02} w_{02} x_{03} w_{03}...
1 x_{11} w_{11} x_{12} w_{12} x_{13} w_{13} ...
.
n x_{n1} w_{n1} x_{n2} w_{n2} x_{n3} w_{n3} ...

Each input file will have all vertices in order starting at 0. The adjacent vertices and weights will also be in order. You are allowed to assume vertices and adjacencies listed in order when programming your solution.



For the graph to the left, the contents of the input file would be:

0 1 8.0 2 5.0
1 0 8.0 2 4.0 3 6.0 4 3.0
2 0 5.0 1 4.0 4 2.0
3 1 6.0 4 5.0 5 1.0
4 1 3.0 2 2.0 3 5.0 5 4.0
5 3 1.0 4 4.0

The output for the minimum spanning tree should be:
0 2 5.0
1 4 3.0
2 0 5.0 4 2.0
3 5 1.0
4 1 3.0 2 2.0 5 4.0
5 3 1.0 4 4.0

The output for shortest paths with vertex 0 as the source should be:
0 1 8.0 2 5.0
1 0 8.0
2 0 5.0 4 7.0
3 4 12.0
4 3 12.0 2 7.0 5 11.0
5 4 11.0

In addition to the above graph (example.txt), a second graph (sample.txt) with solutions (computing distances from node 0) is provided.

Your output for Prim's algorithm will be to a file named **primout.txt** that gives the minimum spanning tree and is of the same adjacency list format as above. For Dijkstra's algorithm, you should prompt the user to input the source vertex s. After calculating the shortest path tree rooted at s, you will output it to the file **dijkstraout.txt** with the format described above.

Algorithmic specifications:
Your algorithms must compute correctly and run in O(|E|log|V|) time on any input graph (which you may assume to be connected). Your implementation should be able to read an input file of any length (and create a corresponding graph data structure) without knowing the number of vertices in advance. Your implementations should incorporate useful classes for graphs and vertices. Because a graph is an object, and Prim and Dijkstra are algorithms that are run on that object, a logical way to structure your program would be as a Graph class with methods that execute Prim and Dijkstra.

You have already implemented a priority queue with location index, and you should use that priority queue with Dijkstra's algorithm. In Dijkstra's algorithm, upon finishing a vertex, the algorithm does a decrease key on all adjacent vertices. You should use the location index for an O(1) way of finding the location of the vertex in the priority queue. Do not turn Dijkstra into an $O(n^2)$ algorithm by doing a linear search through the priority queue to find a vertex!

This project has no report. However, you must document amply and appropriately.

What to Turn in: You must turn in a single zipped file containing your source code in one of the allowable languages (C++, C#, Java). There is no report.

Grading will be according to the following rubric:

Node and Edge classes	0 Classes missing	5 Node and edge class with errors	10 Node and edge classes correctly implemented	
Graph Class	0 No Graph class	5 Graph class has errors	10 Graph class correctly implemented	
Program reads input correctly	0 Does not read input	5 Graph size is hard coded	10 Reads input correctly	15 Reads input of any length
Heap implemented correctly	0 No heap	5 Heap uses O(n) search to increase keys	10 Heap uses lookup table correctly.	
Prim output correct	0 No Prim output	5 Prim output with errors	15 Prim output with no errors	
Dijkstra output correct	0 No Dijkstra output	5 Dijsktra output with errors	15 Dijkstra output with no errors	