CSCI 33500 - Spring 2016 -
Homework #4
Graphs - Prim's Algorithm

Due in class Thursday April 21st

The goal of this homework is to understand and implement a working version of Prim's algorithm for finding a Minimum Spanning Tree in a weighted, undirected, and connected graph. The algorithm's description can be found in chapter 9.5.1 / page 414 of the book. The secondary goal of this homework is to compare the two classic graph representations: the adjacency matrix and adjacency list.

1 Practice

Below is a adjacency matrix representation of a graph. An X in a cell means no connection between those vertices. The value in a cell is the weight associated with that edge.

	A	B	C	D	E	F	G	H
A	X	2	5	7	X	5	2	9
B	2	X	4	5	9	1	8	2
C	5	4	X	5	3	X	4	10
D	7	5	5	X	6	7	9	2
E	X	9	3	6	X	X	X	X
F	5	1	X	7	X	X	10	10
G	2	8	4	9	X	10	X	3
H	9	2	10	2	X	10	3	X

- Give the equivalent adjacency *list* representation of the above graph. Don't forget the edge weights.
- Draw the graph and compute the minimum spanning tree using Prim's algorithm, drawing the graph after each stage of the algorithm.

2 Implementation

- Write a procedure `gen_random_graph` that generates a weighted, undirected random graph.

  Your procedure should:

  - Have for input an integer N as the number of vertices in the graph.
  - Assign random weights on each edges ranging from 1 to 10. Not all possible edges are present though! As in the above example, represented an absent edge by an X.
  - Return a pair `(M,L)`, with M and L respectively being the matrix and list representation of the (same) random graph you are generating.
  - Read section 9.1.1 Representation of Graph on page 380 of the book for insight on how to create these data structures.
  - Use non-digit characters as vertex names, to avoid confusion with edge weights.
  - Bonus: guarantee that the graph is connected.

- Write two procedures `solve_mst_prim_matrix` and `solve_mst_prim_list` that respectively take as input an adjacency matrix and an adjacency list, and return as output a minimum spanning tree, using Prim's algorithm. The input and output should be the same data structure as what you created in `gen_random_graph`. If the graph is not fully connected, the procedures should return an empty graph.
  - It makes senses to write accompanying procedures, for example `get_neighbors_matrix(vertex, graph)`, `get_neighbors_list(vertex, graph)`, or `get_smallest_edge(graph)`. Be sure to document any of these procedures properly.

3 Experiment & Analysis

In your `main()` call, run the following experiments:

- Generate a random graph of size 5 and display both matrix and list representations in the terminal.
- Compute the minimum spanning tree for each representation using the procedures you wrote, and display both results in the terminal.

- Generate a 1000 random graphs of size 100 (both matrix and list representation). Compare the total runtime of finding the minimum spanning tree on these graphs using adjacency matrixes vs adjacency lists. Do not take into account non-connected graphs (which should return nothing).

- Use `std::chrono::high_resolution_clock` from the `<chrono>` header included in C++11 to compare runtime.

4 How to submit

Your homework should be zipped in a single file containing your code, and a PDF readme documenting your code, your answers, and the analysis of the experiments. Name your zip HW4_STUDENTNAME.zip, and email it to `fg297@hunter.cuny.edu` by noon on Thursday April 21st.