CSC263H1: Problem Set 8

Due Tuesday ~~December 3 before 10pm~~

December 4 before 10pm

General instructions

Please read the following instructions carefully before starting the problem set. They contain important information about general problem set expectations, problem set submission instructions, and reminders of course policies.

- Your problem sets are graded on both correctness and clarity of communication. Solutions that are technically correct but poorly written will not receive full marks. Please read over your solutions carefully before submitting them.
- Each problem set may be completed in groups of up to three. If you are working in a group for this problem set, please consult the articles on collaboration and plagiarism on posted on quercus.
- Solutions must be typeset electronically, and submitted as a PDF with the correct filename. **Handwritten submissions will receive a grade of ZERO.**

  The required filename for this problem set is **problem_set8.pdf**.
- Problem sets must be submitted online through CrowdMark. If you haven't used CrowdMark before, give yourself plenty of time to figure it out, and ask for help if you need it! If you are working with a partner, you must form a group on CrowdMark, and make one submission per group. "I didn't know how to use CrowdMark" is not a valid excuse for submitting late work.
- Your submitted file(s) should not be larger than 9MB. You might exceed this limit if you use a word processor like Microsoft Word to create a PDF; if it does, you should look into PDF compression tools to make your PDF smaller, although please make sure that your PDF is still legible before submitting!
- Submissions must be made *before* the due date on CrowdMark.
- The work you submit must be that of your group; you may not use or copy from the work of other groups, or external sources like websites or textbooks.

1. **[12 marks] Bipartite Graphs:** A graph $G = (V, E)$ is bipartite if and only if it is possible to partition its vertices into two sets V_1 and V_2, so that all edges have one endpoint in each set. It is important to recognize when a graph is bipartite since there are many fast graph algorithms for this class of graphs.

   (a) Show that a graph is bipartite if and only if it contains no odd cycle. *(Note the if and only if means you need prove both ways)*

   (b) Does this correctness condition hold for a graph that is not connected? You can explain through an example

   (c) Using a modified version of Depth-First search, design a function BIPARTITE(G) that, given a connected graph G in terms of its adjacency-list, returns TRUE if G is bipartite and FALSE otherwise. Show the run-time complexity of your algorithm as well.

2. **[15 marks] Complete Graphs:** A *complete graph* K_n is the undirected graph on n vertices in which every pair of vertices is joined by an edge.

   (a) What is the running time of the breadth first search algorithm when the input graph is K_n?

   (b) Modify BFS to obtain an algorithm which, given an undirected graph G with n vertices and m edges, determines whether it is connected in $O(m+n)$ time, but whose running time for K_n is only $\Theta(n)$.

   (c) Find a *disconnected graph* on n vertices for which the running time of the algorithm from ~~(a)~~ (b) is $\Theta(n^2)$, or explain why no such graph exists.

3. **[0 marks] Spanning:**

   This extra question is for your own edification. If you attempted it, we encourage you to upload your solution, for our data, but it will not be marked.

   Consider a weighted, undirected graph $G = (V, E)$, where all edge weights are distinct:

   (a) Define the term minimum spanning tree M of G, and prove that G has exactly one minimum spanning tree. (*Remember that no two edges have same weight*)

   (b) Do Prim's and Kruskal's algorithms still work if the edge weights are allowed to be negative?

   (c) Suppose that we have already computed a minimum-weight spanning tree M in a weighted, undirected graph G. Now suppose we want to either
   (i) increase the weight of one of the edges in M or
   (ii) decrease the weight of one of the edges of G not in M.
   In either case M may no longer be of minimum weight, and we may have to compute a new minimum-weight spanning tree. Describe linear-time (O(n + m)) algorithms for each of these two problems. For (i), assume that the edges not in M are sorted by weight. Describe your algorithms in high-level terms, no pseudocode.