HOMEWORK 8

Due Friday, November 12, 2010 by 1:15pm in class

Please submit each problem separately, i.e. each problem should begin on a new page and only the pages for one problem should be stapled together. Failure to do so might result in some problem(s) not being graded.
For general homework policies and our suggestions, please see the policy document.
Do not turn in Q 3.

---

1. (40 points) In this problem, you will tackle the so called the *bottleneck path problem*. Like the shortest path problem, in this problem you are also given a directed graph $G = (V, E)$ and for every edge $e \in E$, we are given its length $\ell_e \geq 0$. The problem differs from the shortest path problem in its objective. For the bottleneck path problem, given two nodes $s, t \in V$, you need to find a path with the shortest longest edge. More precisely, given a path P connecting s and t in G, it's bottleneck is defined as $b(P) = \max_{e \in P} \ell_e$. The goal is to output an $s - t$ path P^* that minimizes $b(P)$ over all $s - t$ paths P.

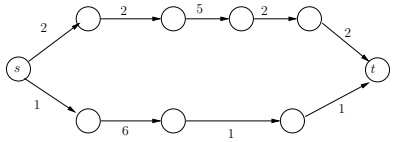

Figure 1: In the graph above, the bottom path is the shortest path (with length 9) while the top path has the minimum bottleneck (of 5).

Present an algorithm that solves the problem above and prove its correctness. (Justify why your algorithm runs in polynomial time: a detailed analysis of its run time is not required.)

Hint: Try to modify Dijkstra's algorithm.

2. (45 + 15 = 60 points) In class we have seem algorithms that compute the MST in time $O(m \log n)$. This is under the assumption that the graph is given it its adjacency list representation. This of course is not as best as possible[1].

   In this problem you will explore how to implement Prim's MST algorithm if the graph is given in its adjacency matrix form. (The adjacency matrix for a weighted graph is the natural generalization of the unweighted one we have seen in class earlier. In particular, the entry for (i, j) in the matrix is a ∞ if (i, j) is not an edge, otherwise it is $c_{(i,j)}$.)

   (a) Show how to implement Prim's algorithm in $O(n^2)$ time if the input graph is given as an adjacency matrix.

[1]The best known running time for an MST algorithm is $O(m \cdot \alpha(n, m))$, where $\alpha(\cdot, \cdot)$ is the inverse Ackermann function and is a a very slooooooooowly growing function– for all practical input values, the func tion value is smaller than (say) 5.

Hint: Look at the implementation of Prim's algorithm in the book and try to think how having an adjacency matrix might help you maintain the critical quantity you need to keep track of (for each vertex). You will have to maintain some auxiliary data structure(s) (but simpler one(s) than the one used in the book's implementation).

(b) Argue that your algorithm above is the best possible: i.e., *no* algorithm that solves the MST problem when the input graph is given in the adjacency matrix format can have a faster asymptotic running time.

Hint: Recall that the running time of an algorithm has to include the time it takes to write its output and the time to read its input.

3. (**DO NOT hand this problem in**) Exercise 8 in Chapter 4.