Avi Hirsch

Design and Analysis of Algorithms

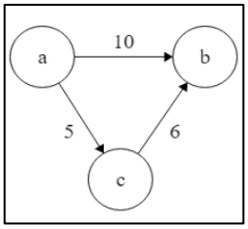
HW #10

1. True. A MST must exist that includes e* because this is the first edge that Kruskal's algorithm removes from its set of edges to check for inclusion in its MST; thus, e* will always be in the MST formed by Kruskal's algorithm.

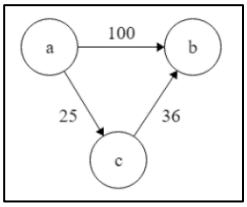
2. (a) True. Every set of ordered positive distinct integers will maintain the same order when every integer in the set is squared. Thus, Kruskal's algorithm, which forms a MST in order of edges (from smallest to greatest), will create the same MST for a graph if every edge's weight is squared. Thus, a MST remains a valid MST if all edge weights are squared.

   (b) False. Proof by counterexample:

   Digraph A, in which the shortest path a->b is edge (a, b):



   Digraph A*, in which the edges of A have been squared:
   The shortest path from a->b is now (a, c), (c, b) = 61.
   Thus, squaring the edges of a digraph
   could change the shortest path.



7. The optimal algorithm is the greedy algorithm whose schedule takes jobs in order of finishing time, **from longest to shortest**, ignoring preprocessing time.
   This is the case because total preprocessing time will be the same regardless of the order of the jobs, since preprocessing times cannot overlap. Thus, we can discount preprocessing time. To finish all the jobs as early as possible, the last job must finish as early as possible; thus, it must be the job with the shortest finishing time. Similarly, the job before that must finish as soon as possible, etc., until the first job, which will have the longest finishing time (since it starts first).
   To prove that this is optimal, we must show that there is no schedule that will finish earlier than this. In other words, given an alternative schedule S that is different from ours, ours will always finish as soon if not sooner than S. S, since it is different from our schedule, must have at least two jobs that are not in order of longest to shortest finishing time—the first job will have a shorter finishing time than the second.
   For example, say job #1 will take 1 second to process and 1 second of finishing time; job #2 will take 1 second to process and 3 seconds of finishing time. S will take job #1 and then job #2, for a total of 5 seconds; our algorithm would reverse the order, taking a total of 4 seconds.
   This specific example can be generalized to all cases; for any two jobs, beginning with the one with a shorter finishing time will be *at best* as good as reversing the order, but sometimes worse. Swapping the two jobs so that the job with a shorter finishing time is later will *never* increase the completion time of the schedule. (Because the finishing times overlap, the later, shorter finishing time will always start as soon as it can, no later than when the jobs were flipped.)