Homework 12, Basic Algorithms, Spring 2014.
Due date: Tuesday, April 29.

Problems 8.13, 8.24 from the text and problems 1–4 below.
For problem 8.24 you need to argue that your algorithm is correct as well as analyzing its run time.

Honors Section. Problem 8.27 from the text and problem 5 below.

1. Consider the following shortest path problem. The input consists of a directed graph $G = (V, E)$ in adjacency list format, together with non-negative edge lengths $l(u, v)$ for each edge $(u, v) \in E$, a specified vertex $r$, along with two subsets of $V$: $S$, a set of supermarkets, and $W$, a set of winestores. The task, is to find, for vertex $r \in V$, the length of a shortest path from $r$ to some $x \in S$, followed by some $y \in W$, and then back to $r$.

Show how to solve this problem in time $O((V + E) \log V)$ by constructing a suitable graph $H$ and running Dijkstra’s algorithm on $H$. Be precise when specifying $H$, and remember to explain the running time of your algorithm.

2. The evacuation problem.
The input comprises a directed graph $G = (V, E)$ in adjacency list format, along with two non-negative edge lengths $l(u, v)$ and $b(u, v)$ for each edge $(u, v) \in E$; in addition, there is a special vertex $p$, the chemical plant, and a second designated vertex $s$, the mayor’s house. The mayor has asked the emergency services to plan evacuation routes from $s$ to each vertex $v$ in the event that the chemical plant catches fire. Such a fire would spread from $u$ to $v$ along edge $(u, v)$ in time $b(u, v)$. An evacuation route may only go through vertices that are not on fire, but among all such routes should be the shortest possible (measured in terms of $l$ which is measured in the same time units as $b$).

Show how to modify and/or use Dijkstra’s algorithm in order to compute the lengths of the evacuation routes from $s$ in time $O((E + V) \log V)$.

3.a. This problem concerns the implementation of a Priority Queue $Q$ with key values in the range $[0, m - 1]$. $n$ insertions are performed initially; for convenience, these $n$ items are named $1, \ldots, n$. These insertions are followed by $m$ ReduceKey operations and up to $n$ DeleteMins. Also the following guarantee is made: once an item with key value $k$ has been removed from $Q$ (by a DeleteMin), every subsequent ReduceKey operation, ReduceKey($i, l$), will have $l \geq k$; i.e. subsequently, all key values in $Q$ remain greater than or equal to $k$.

Show how to implement this Priority Queue so that the initial $n$ insertions take $\Theta(n)$ time, each ReduceKey takes $\Theta(1)$ time, and the up to $n$ DeleteMins take $O(m + n)$ time.
Hint. i. Use an array of $m$ buckets to store the items in $Q$, and crosslink the items in $Q$ with an array QLoc[1 : n] of pointers into $Q$: QLoc($i$) is the pointer to item $i$ if it is in $Q$ and is nil otherwise.
ii. Having removed an item with key $k$ (by a DeleteMin), how can you use your knowledge of $k$ to speed-up the next DeleteMin?

b. Consider Dijkstra algorithm, and suppose that the edge lengths are all integers and further you are guaranteed that every shortest path has length at most $m - 1$. Show how to
implement Dijkstra’s algorithm to run in time $O(E + V)$.
(Be careful: non-shortest paths may have length $m$ or greater.)

c. Why does this approach not yield an $O(E + V)$ time implementation of Prim’s algorithm in the event that all edge lengths are integers in the range $[0, m - 1]$?

4. Let $G = (E, V)$ be an undirected graph. Suppose that the edge lengths are all distinct. Show that the minimum spanning tree is unique.

5. Let $G = (V, E)$ be an undirected graph. Two vertices are said to be biconnected if they lie on the same simple cycle. It is a fact that biconnectivity is an equivalence relation (you need not prove this).

This problem concerns an alternative solution to problem 8.27. The task is to identify for each edge $e \in T$ an edge $e' \neq e$ to replace $e$ in the event that $e$ is removed from $G$, where $T_e = T - \{e\} \cup \{e'\}$ is required to be a minimum spanning tree of $G' = (V, E - \{e\})$.

Notice that, when added to $T$, $e'$ forms a cycle on which $e$ lies. Every other edge on this cycle will either have $e'$ as a repair edge, or an even lower cost edge as a repair edge.

This suggests the following approach to computing the repair edges. Process the edges not in $T$ in increasing cost order. Keep track of the biconnected components (as sets of vertices) formed by the repair edges found so far. Then what identifies a new repair edge and for which edges is it a repair edge?

Flesh out the above outline. Show that its running is the time for $O(n)$ unions and $O(m)$ finds, plus the time to sort the $m$ edges using their costs as the key, plus $O(E + V)$ time.