Problem Set 4 (due Tuesday, October 24)

(The solutions to the exercises from the book are taken from those provided by the authors.)

1. (5 + 5 = 10 points) Uniqueness of MSTs when all weights are distinct

- Suppose T_1 and T_2 are distinct minimum spanning trees for graph G. Let (u, v) be the lightest edge (smallest weight edge) among all edges that are in T_1 and but not in T_2. Let (x, y) be any edge that is in T_2 and not in T_1. Show that $w(x, y) \geq w(u, v)$.

  Answer: Suppose we add edge (x, y) to T_1. This creates a cycle C. By the MST property, it follows that the weight of every edge in C is at most $w(x, y)$. Furthermore, at least one edge in C is in T_1 and not in T_2, because otherwise T_2 has a cycle. Let this edge be (a, b). Thus, it follows that $w(x, y) \geq w(a, b)$. Since (u, v) is the lightest edge that is in T_1 and not in T_2, we have $w(u, v) \leq w(a, b)$. This implies that $w(x, y) \geq w(u, v)$, thus proving the desired claim.

- Prove that if the weights on the edges of a connected, undirected graph are distinct, then there is a unique minimum spanning tree.

  Answer: Let, if possible, there be two distinct minimum spanning trees T_1 and T_2. We will derive a contradiction. Let (u, v) be the smallest weight edge in T_1 that is not in T_2, and let (u', v') be the smallest weight edge in T_2 that is not in T_1. Since all the edges weights are distinct, we have either $w(u, v) < w(u', v')$ or $w(u', v') < w(u, v)$. Without loss of generality, assume the former. Now suppose we add the edge (u, v) to T_2. This forms a cycle, say C, in T_2. Cycle C contains at least one edge that is in T_2 and not in T_1, because otherwise T_2 has a cycle. Let this edge be (x, y). By the definition of (u', v'), we have $w(u', v') \geq w(x, y)$. But $w(u', v') > w(u, v)$, implying that $w(x, y) > w(u, v)$, violating the MST property of T_2. This yields a contradiction.

2. (6 + 6 = 12 points) Bottleneck MSTs

(a) This is false. Let G have vertices v_1, v_2, v_3, v_4, with edges between each pair of vertices, and with the weight on the edge from v_i to v_j equal to $i+j$. Then every tree has a bottleneck edge of weight at least 5, so the tree consisting of a path through vertices v_3, v_2, v_1, v_4 is a minimum bottleneck tree. It is not a spanning tree, however, since its total weight is greater than that of the tree with edges from v_1 to every other vertex.

(b) This is true. Suppose that T is a minimum spanning tree of G, and T' is a spanning tree with a lighter bottleneck edge. Thus, T contains an edge e that is heavier than every edge in T', it forms a cycle C on which it is the heaviest edge (since all other edges in C belong to T'). By the Cycle Property, then e does not belong to any minimum spanning tree, contradicting the fact that it is in T and T is a minimum spanning tree.

3. (2 + 10 = 12 points) Nesting boxes

We are given a set of n empty boxes $B_i = (h_i, w_i, \ell_i)$, $1 \leq i \leq n$, where h_i, w_i, and ℓ_i denote the height, width, and length of box B_i. Let us say that box B_i *nests* within box B_j if and only if box B_i can be oriented so that it is *strictly smaller* than box B_j in each corresponding dimension. For example, even though the box $B_i = (5, 1, 1)$ is taller than the box $B_j = (2, 6, 3)$, B_i can be placed inside B_j by orienting it as a $(1, 5, 1)$.

(a) Describe an $O(1)$ time algorithm to determine whether one box nests within another.

Answer: In order to determine whether B_i nests in B_j, we run the following algorithm. We sort the three dimensions of B_i and B_j in nonincreasing order. Let (x_i, y_i, z_i) and (x_j, y_j, z_j) be the two vectors obtained. We thus have $x_i \geq y_i \geq z_i$ and $x_j \geq y_j \geq z_j$. We claim that B_i nests in B_j if and only if $x_i < x_j$, $y_i < y_j$, and $z_i < z_j$. To see this claim, we first note that if $x_i < x_j$, $y_i < y_j$, and $z_i < z_j$, then clearly B_i nests in B_j. For the other direction, we need to prove that if B_i nests in B_j, then $x_i < x_j$, $y_i < y_j$, and $z_i < z_j$. If not, then either $x_i \geq x_j$, $y_i \geq y_j$, or $z_i \geq z_j$. If $x_i \geq x_j$, then $x_i \geq y_j$ and $x_i \geq z_j$. This contradicts the fact that B_i nests in B_j. Similarly, we can derive contradictions under the assumption $y_i \geq y_j$ or $z_i \geq z_j$. Thus, we have an $O(1)$ procedure to determine whether a given box B_i nests in another box B_j.

(b) We would like to compute the maximum length k of a sequence of boxes $B_{i_1}, \ldots, B_{i_k}$ such that B_{i_j} nests inside $B_{i_{j+1}}$, $1 \leq j < k$. Design and analyze a polynomial-time algorithm for the problem. (*Hint:* One can use either dynamic programming or express the problem as a shortest-path problem in a suitably defined graph.)

Answer: We construct a graph $G = (V, E)$ as follows. The set V is the set of n boxes. We have an edge from box B_i to box B_j if and only if B_i nests in B_j.

The above process of constructing the graph takes $O(n^2)$ time since we have to check for every pair of boxes. We claim that G is acyclic. Our proof is by contradiction. Let, if possible, $B_1, B_2, \ldots, B_k, B_1$ form a cycle in G. Looking at the x-dimensions of each of the boxes, we get $x_1 < x_2 < \ldots < x_k < x_1$, a contradiction.

We next claim that any valid sequence of "nestable" boxes is a path in G and vice versa. To see this, we first note that if $B_{i_1}, \ldots, B_{i_k}$ is a valid sequence of nestable boxes then B_{i_j} nests inside $B_{i_{j+1}}$, $1 \leq j < k$. Therefore, $B_{i_1} \rightarrow \ldots \rightarrow B_{i_k}$ is a path in G. Similarly, if $B_{i_1} \rightarrow \ldots \rightarrow B_{i_k}$ in G, then by our choice of edges, B_{i_j} nests inside $B_{i_{j+1}}$, $1 \leq j < k$. Therefore, $B_{i_1}, \ldots, B_{i_k}$ is a valid sequence of nestable boxes. We have thus reduced the problem of finding the maximum length sequence of nestable boxes to the problem of finding the longest path in G.

In order to find the longest path in G, we augment the graph slightly to obtain $G' = (V', E')$ by adding two new vertices s and t, and adding edges from s to every vertex (box) in V and an edge from every vertex in V to t. Thus, V' equals $V \cup \{s, t\}$ and E' equals $E \cup \{(s, v), (v, t) : v \in V\}$. Hence, $|V'|$ equals $n + 2$ and $|E'|$ is $|E| + 2n = O(n^2)$. We assign a weight of -1 to every edge in E'.

It is easy to see that any path p in G from a vertex u to v corresponds to a path p' in G' which consists of the edge (s, u) followed by the path p to v and then the edge (v, t). And vice versa. Therefore, the problem of finding the longest path in G is equivalent to finding the shortest path in G' (since the weights are all -1). Also, there are no negative weight cycles since there are no cycles at all. We can compute the shortest path from s to t in $O(n^3)$

time using the Bellman-Ford algorithm we studied in class. In fact, for dags, one can actually solve the problem faster in $O(n^2)$ time. This path yields a sequence of vertices in V, which is the longest sequence of nestable boxes.

4. (13 points) Word segmentation

The key observation to make in this problem is that if the segmentation $y_1 y_2 ... y_n$ is an optimal one for the string y, then the segmentation $y_1 y_2 ... y_n$ would be an optimal segmentation for the prefix of y that excludes y_n (because otherwise we could substitute the optimal solution for the prefix in the original problem and get a better solution).

Given this observation, we design the subproblems as follows. Let $\textsc{Opt}(i)$ be the score of the best segmentation of the prefix consisting of the first i characters of y. We claim that the recurrence

$$\textsc{Opt}(i) = \max_{j \leq i} \textsc{Opt}(j-1) + \textsc{Quality}(j...n).$$

would give us the correct optimal segmentation where $\textsc{Quality}(a...b)$ means the quality of the word that is formed by the characters starting from position a and ending in position b). Notice that the desired solution is $\textsc{Opt}(n)$.

We prove the correctness of the above formula by induction on the index i. The base case is trivial, since there is only one word with one letter.

For the inductive step, assume that we know that Opt function as written above finds the optimum solution for the indices less than i, and we want to show that the value $\textsc{Opt}(i)$ is the optimum cost of any segmentation for the prefix of y up to the i-th character. We consider the last word in the optimal segmentation of this prefix. Let's assume it starts at index $j \leq i$. Then according to our key observation above, the prefix containing only the first $j-1$ characters characters must also be optimal. But according to our induction hypothesis, $\textsc{Opt}(j)$ will yield us the value of the aforementioned optimal segmentation. Therefore the optimal cost $\textsc{Opt}(i)$ would be equal to $\textsc{Opt}(j)$ plus the cost of the last word.

But notice that our above recurrence exactly does this calculation for each possibility of the last word. Therefore our recurrence will correctly find the cost of the optimal segmentation.

As for the running time, a simple implementation (direct evaluation of the above formula starting at index 1 until n, where n is the number of characters in the input string) will yield a quadratic algorithm.

5. (13 points) Defending against robot attacks

(a) Change x_4 to 2 in the given example. Then this algorithm would activate EMP at times 2 and 4, for a total of 4 destroyed; but activating at times 3 and 4 as before still gets 5.

(b) Let $\textsc{Opt}(j)$ be the maximum number of robots that can be destroyed for the instance of the problem just on $x_1 x_2 ... x_j$. Clearly if the input ends at x_j, there is no reason not to activate the EMP then (you're not saving it for anything), so the choice is just when to last activate it before step j. Thus $\textsc{Opt}(j)$ is the best for these choices over all i:

$$\textsc{Opt}(j) = \max_{0 \le i < j} [\textsc{Opt}(i) + \min(x_j, f(j-i))]$$

where $\textsc{Opt}(0)=0$. The full algorithm is just

Set $\textsc{Opt}(0)=0$
For i=1,2,...,n
    Compute $\textsc{Opt}(j)$ using the recurrence
Endfor
Return $\textsc{Opt}(n)$

The running time is $O(n)$ per iteration, for a total of $O(n^2)$.

An alternative solution would define $\textsc{Opt}'(j,k)$ to be the best solution for steps j through n, given that the EMP in step j has already been charging for k steps. the optimal way to solve this subproblem would be to either activate the EMP in step j or not, and $\textsc{Opt}'(j,k)$ is just the better of those choices:

$$\textsc{Opt}'(j,k) = \max(\min(x_j, f(k)) + \textsc{Opt}'(j+1,1), \textsc{Opt}'(j+1,k+1)).$$

We initialize $\textsc{Opt}'(n,k) = \min(x_n, f(k))$ for all k, and the full algorithm is

Set $\textsc{Opt}'(n,k) = \min(x_n, f(k))$ for all k
For j = n-1, n-2, ...,1
    For k = 1,2,....j
        Compute $\textsc{Opt}'(j,k)$ using the recurrence
    End For
End For
Return $\textsc{Opt}'$(1,1)

The running time is O(1) per entry of $\textsc{Opt}'$, for a total of $O(n^2)$.