Name: _____

1. (3 points) Several of your friends have decided to hike the Appalachian Trail next summer. They want to hike as much of the trail as possible per day, but do not want to hike in the dark. On a map, they've identified a large set of good spots for camping, and are considering the following system for deciding when to stop each day. Each time they come to a potential camp site, they determine whether they can make it to the next one before nightfall. If they can make it, then they keep hiking; otherwise, they stop. They believe this system is optimal in that it minimizes the number of camping stops they will have to make. Show that their algorithm has the greedy choice property.

You may consider the trail to be a line segment of length L, assume that your friends can hike d miles per day irrespective of terrain and weather, and that the potential campsites are located at distances $x_1, x_2, \ldots, x_n$ miles from the start of the trail. In addition, you may assume that the campsites are never more than d miles apart, that the first campsite is d or fewer miles from the start of the trail, and the n^{th} campsite is d or fewer miles from the end of the trail.

Solution

For this problem, you need to demonstrate that you understand what the greedy choice property is — that there is an optimal solution that includes the greedy choice — and that you can use the exchange method to prove that the algorithm has the greedy choice property. The exchange method has two major components: describing the general form of an optimal (but not necessarily greedy) solution and demonstrating that the optimal solution can be modified to include the greedy choice.

We will say a set of campsites is *feasible* if consecutive sites are within distance d of each other, the first campsite is within d of the start of the trail, and the last campsite is within distance d of the end of the trail. Let S denote an optimal solution to the problem. Then S is a feasible set of campsites, $S = \{i_1, i_2, \ldots, i_k\}$, such that there is no feasible set with fewer than k campsites.

Suppose S does not use the greedy choice, so the hikers could have gone farther than the first chosen campsite, i_1, on the first day. Let campsite j be the greedy choice for the first day, that is, the furthest campsite they could have reached. It must be that $x_{i_1} < x_j \leq d$. Let $S' = \{j, i_2, i_3, \ldots, i_k\}$. S' only differs from S in the selection of the first campsite. Since campsite j is within d miles of the start of the trail and $x_{s_2} - x_j < x_{s_2} - x_{s_1} < d$, S' is a feasible set of campsites. Note that, in this last inequality, $x_{s_2} > x_j$ since, if it were not, we could produce a solution with $k - 1$ campsites, contradicting the assumption that S was optimal. Since S' is a feasible set of size k, it is optimal, and we have proven that the algorithm has the greedy choice property.

(continued on other side)

2. (3 points) Consider the following, incomplete c-table to compute the LCS length for the sequences ACGAA and GCGTCA:

		G	C	G	T	C	A
	0	0	0	0	0	0	0
A	0	***0***	0	0	0	0	1
C	0	0	***1***	**1**	**1**	1	1
G	0	1	**1**	***2***	**2**	2	2
A	0	1	1	*2*	***2***	***2***	**3**
A	0	1	1	2	**2**	**2**	***3***

Solution

(a) Complete the c-table. Show all work.

The bold elements are the values that had to be computed. The values are computed starting at the top, left-most unknown value, working left to right to complete the row, then moving to the next row, again starting at the left-most element. To compute an individual element, say $c[i, j]$ we look at the corresponding letters x_i and y_j of the sequences. If $x_i = j_j$, then we wet $c[i, j] = c[i - 1, j - 1] + 1$; otherwise, set set $c[i, j]$ to the larger of $c[i - 1, j]$ and $c[i, j - 1]$.

(b) Reconstruct an LCS using the completed table. On the table, indicate the path used to reconstruct the sequence.

Starting with $c[5, 6]$, we reconstruct the LCS $\langle C, G, A\rangle$. A possible path used to reconstruct the sequence is indicated by the numbers in *italics*.

Note that you could reconstruct the same LCS starting at $c[4, 6]$.

3. (4 points) Let $G = (V, E)$ be a directed graph, and let $m = |E|$ and $n = |V|$. For all edges $(u, v) \in E$, let $w(u, v) \geq 0$ be real-valued edge weights. If the number of edges is large, specifically if $m = \Theta(n^2)$, then a min-heap is *not* the most efficient data structure to use with Dijkstra's algorithm; in fact, it would be better to use a simple n-long array of pointers to the vertex objects. *Explain all answers!*

Solution

(a) Using the array of pointers to vertices, what is the running time of EXTRACT-MIN, the procedure which extracts the minimum-weight vertex from the array?

The EXTRACT-MIN function finds the vertex with lowest path weight, returns a pointer to the vertex, and removes the vertex from the data structure. We have to scan the entire n-long array, accessing $v.d$ for each vertex. This is a $\Theta(n)$ operation.

(b) Using the array of pointers to vertices, what is the running time of RELAX, the procedure which adjusts the path weights of vertices when necessary?

Using an array, there is no min-heap property to be maintained, so reducing the path weight of a vertex is a $\Theta(1)$ operation, and so the entire RELAX function is $\Theta(1)$.

(c) What is the running time of Dijkstra's algorithm using the array of pointers to vertices and assuming that $m = \Theta(n^2)$?

The running time of Dijkstra's algorithm is

$$O(m \cdot \text{Cost of RELAX} + n \cdot \text{Cost of EXTRACT-MIN}),$$

so using the array data structure and assuming $m = \Theta(n^2)$, the running time is

$$O(m \cdot 1 + n \cdot n) = O(m + n^2) = O(n^2 + n^2) = O(n^2).$$

(d) What is the running time of Dijkstra's algorithm using a min-heap, assuming that $m = \Theta(n^2)$. Justify the conclusion that the array of pointers is better in this case.

The usual running time for Dijkstra's algorithm is $O((m + n) \lg n)$, which, assuming $m = \Theta(n^2)$, becomes $O(n^2 \lg n)$, which is larger than the Big-Oh bound for the array version.