Name: ______________________________

1. (3 points) Professor Gecko has always dreamed of inline-skating across North Dakota. The professor can carry two liters of water and can skate m miles before running out of water. He will start with two full liters of water, and he has a map that shows all the places along his route at which he can refill his water and the distances between these locations. The professor's goal is to minimize the number of water stops along his route, and he believes the following greedy algorithm will do so: whenever he reaches a water stop, he checks to see if he can make it to the next stop before running out of water; if so, he continues to the next stop; otherwise, he stops and fills his water bottle. Prove that this algorithm has the greedy choice property.

You may consider Professor Gecko's route to be a line segment of length L. Assume that the water stops are located at distances $x_1, x_2, \ldots, x_n$ miles from the start of the route, they are never more than m miles apart, that the first stop is m or fewer miles from the start of the route, and the n^{th} stop is m or fewer miles from the end of the route.

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 water stops is *feasible* if consecutive stops are within distance m of each other, the first stop is within m of the start of the route, and the last water stop is within distance m of the end of the route. Let S denote an optimal solution to the problem. Then S is a feasible set of stops, $S = \{i_1, i_2, \ldots, i_k\}$, such that there is no feasible set with fewer than k stops.

Suppose S does not use the greedy choice, so Prof. Gecko could have gone farther than the first chosen stop, i_1, without running out of water. Let stop j be the greedy choice for the first stop, that is, the furthest stop he could have reached. It must be that $x_{i_1} < x_j \le m$. Let $S' = \{j, i_2, i_3, \ldots, i_k\}$. S' only differs from S in the selection of the first stop. Since stop j is within m miles of the start of the route and $x_{s_2} - x_j < x_{s_2} - x_{s_1} < m$, S' is a feasible set of stops. Note that, in this last inequality, $x_{s_2} > x_j$ since, if it were not, we could produce a solution with $k-1$ stops, 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 ACGTCA:

		A	C	G	T	C	A
	0	0	0	0	0	0	0
A	0	*1*	1	1	1	1	1
C	0	1	***2***	**2**	**2**	2	2
G	0	1	**2**	***3***	**3**	3	3
A	0	1	2	*3*	***3***	***3***	**4**
A	0	1	2	3	**3**	**3**	***4***

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 $\langle A, C, G, A \rangle$. 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.