Basic Algorithms CSCI-UA.0310

Practice Test (with Solutions)

Problem 1 (10+10 points)

(a) Describe a point set with n points that is the worst-case input for the Jarvis' algorithm that finds the convex hull of the given input points.
Note that the Jarvis' algorithm was discussed in the recitation lecture.

(b) Let $G = (V, E)$ be a directed graph without any cycles. How many strongly connected components can G have? Justify your answer.

(a) The running time of the Jarvis' algorithm on a given input of n points is $O(nh)$, where h is the number of points on the boundary of the convex hull of the given n points. Note that $3 \leq h \leq n$. Thus, the running time becomes the worst if $h = n$, or in other words, if all the n points lie on the boundary of their convex hull.

(b) Refer to the last claim on page 101 of the lecture notes: A strongly connected component is by definition connected, and connected directed graphs with more than one vertex have cycles. Therefore, if the graph G has no cycles, each of its strongly connected components must only consist of one vertex. This means that G has $|V|$ strongly connected components.

Problem 2 (20 points)

Consider the array $A[1 \dots n]$ of n non-negative integers. There is a frog on the first index of the array. In each step, if the frog is positioned on the i^{th} index, then it can hop to any of the indices $i, \dots, i + A[i]$ (so the frog can at most hop to the index $i + A[i]$).
Develop a greedy algorithm to determine the minimum number of hops necessary so that the frog can reach the last index, i.e., the n^{th} index. You must fully explain your algorithm.
Your algorithm must return -1 if the frog cannot reach the last index.

This problem is totally similar to HW7 P2. You can simply use the idea of that problem to get full mark.
Here is another direct approach:
In each step of the algorithm, if the frog stands on the index i, it must hop to the admissible index j among $i, \dots, i + A[i]$ which gives it the farthest admissible position in the next step. In other words, the frog must hop to the index j among $i, \dots, i + A[i]$ for which $j + A[j]$ is maximized. Note that when the frog stands on the index j, then the farthest admissible position it can hop to will be $j + A[j]$, which is why the frog has to choose the index j for which $j + A[j]$ is maximized.
If $A[j] = 0$, this means that no subsequent indices will be reachable. In this case, the algorithm must return -1 indicating that the frog cannot reach the last index.

Problem 3 (20 points)

Given a directed weighted graph G with no cycles, develop an algorithm to find the weights of all longest paths from one source vertex s to the other vertices in G in $O(|V| + |E|)$ time.
A longest path from u to v is a path of maximum weight from u to v.

This problem is totally similar to Problem 6 of the Additional Practice Problems. The same solutions works here, except that we are looking for the longest paths instead of the shortest paths. Thus, only min must be replaced by max in the solution.

Problem 4 (20 points)

Consider a directed graph $G = (V, E)$ where each edge is colored in either red or blue. Develop an $O(|V|+|E|)$ time algorithm that, given vertices u and v in G, determines if there exists a walk from u to v that uses at least one blue edge (note that this walk may have repeated vertices). Justify the correctness of your algorithm and analyze the running time.
(Hint: Try to construct a new graph using an idea similar to HW9 P4.)

The idea is similar to HW9 P4 with some modifications in constructing the new graph. We have to construct a new graph G' which is not colored, and a BFS (or DFS) on G' determines if there is a walk from u to v in the input graph G with at least one blue edge.
For this, let $n = |V|$, and label the n vertices of G by $v_1, \ldots, v_n$. Construct two graphs G_1 and G_2 each having a copy of the n vertices $v_1, \ldots, v_n$. Thus, G_1 and G_2 have altogether $2n$ vertices combined, and each has a copy of the vertices v_i's.
Take all the red edges of G and draw them as the edges of G_1, and also draw them as the edges of G_2. Thus, G_1 and G_2 each contains all the red edges of G. Now, if G has a blue edge (v_i, v_j), take the copy of v_i from G_1 and take the copy of v_j from G_2 and draw a directed edge among them. Also, do the same thing reversely, i.e., take the copy of v_i from G_2 and take the copy of v_j from G_1 and draw a directed edge among them. Thus, each blue edge of G accounts for two edges running between G_1 and G_2; one from G_1 to G_2, and one from G_2 to G_1. Let G' be the combination of G_1 and G_2 constructed this way. G' has $2n$ vertices and twice the number of edges as G (since each blue edge of G is drawn twice running between G_1 and G_2, and G_1 and G_2 each has one copy of each red edge of G. Thus, G' has two copies of each edge of G). If we want to find a walk from u to v in G with at least one blue edge, we can simply run BFS (or DFS) in G' to check if there is a path from the copy of u in G_1 to the copy of v in G_2. The reason is that if we want to start from a vertex in G_1 and end up in a vertex in G_2, we need to run from G_1 to G_2 at least once. Traversing from G_1 to G_2 is only possible by using a blue edge due to the construction.
Note that the algorithm, i.e., running BFS (or DFS) on G' from the copy of u in G_1, only takes $O(2|V| + 2|E|) = O(|V| + |E|)$ time (since G' has $2|V|$ vertices and $2|E|$ edges).

Problem 5 (20 points)

Consider the n points $p_1, p_2, \ldots, p_n$ in the plane. Each point p_i has the xy-coordinates $p_i = (x_i, y_i)$. The cost of connecting two points $p_i = (x_i, y_i)$ and $p_j = (x_j, y_j)$ is defined as

$$|x_i - x_j| + |y_i - y_j|.$$

For example, the cost of connecting $p_i = (1, 5)$ and $p_j = (3, 4)$ is

$$|1 - 3| + |5 - 4| = 3.$$

We want to make all the points connected such that there is a path between any pair of points. Develop an algorithm that returns the minimum cost to make all the points connected. Your algorithm must run in $O(n^2)$ time. Explain all the steps of your algorithm.

The problem just asks for a minimum spanning tree (MST) connecting all the points. For this, represent each point p_i by a vertex and draw all possible pairwise edges among these n points. This way, the complete undirected graph $G = (V, E)$ obtained has n vertices and $n(n-1)/2 = \Theta(n^2)$ edges (refer to page 104 of the lecture notes). If the cost of connecting the points p_i to p_j is assigned to the edge p_ip_j, then G becomes a weighted graph. This means that we can simply run the Prim's algorithm to obtain a minimum spanning tree of G. The weight of the minimum spanning tree is the minimum cost to connect all the points (vertices). The graph has $\Theta(n^2)$ edges, and computing the weight of each edge only takes $O(1)$ time. Thus, it takes $\Theta(n^2)$ time to compute the weights of all the edges. If we run the Prim's algorithm using Fibonacci heaps, as seen in the lecture (the same running time obtained for the Dijkstra's algorithm on page 155 of the lecture notes), the total running time of the Prim's algorithm becomes $O(|E| + |V| \log |V|)$. Since $|V| = n$ and $|E| = \Theta(n^2)$, this gives the total running time as $O(n^2 + n \log n) = O(n^2)$.

Remark: You do not need to mention the bound obtained by using the Fibonacci heaps to get full mark. A solution with either of the Kruskal's algorithm or the Prim's algorithm with the version that uses regular heaps with worse running time, instead of the Fibonacci heaps, will be given full mark for this problem, as well.