INDIAN STATISTICAL INSTITUTE

Class Test – II

M Tech (CS) – I Year, 2018-2019 (Semester – II)

Design and Analysis of Algorithms

Date: 15.02.2019 Maximum Marks: 30 Duration : 1.0 hour

Note: The question paper is of 40 marks. Answer as much as you can, but the maximum you can score is 30. Answer a question within its allotted box. Give time complexity analysis for any algorithm you design.

Course: (M Tech/JRF/PLP) _____

Name: _____ **Roll Number:** _____

(Q1) You are given a sequence $\mathcal{X}$ of n real numbers. Design and analyze an efficient algorithm to find a contiguous subsequence $\mathcal{X}_{ij} = \mathcal{X}(i) \ldots \mathcal{X}(j)$ of $\mathcal{X}$ for which the sum of elements in the subsequence is maximized. (Note that $3, -4, 9$ is a contiguous subsequence of $\{1, -2, 3, -4, 9\}$.) [10]

First, note that if all numbers were positive, the entire array is the answer. A negative number throws up options. You may gain or lose if you include it. When do you gain and when do you lose? Think it over.

A quick look at the characteristics of the problem shows that it admits overlapping subproblems and the optimal substructure property. Thus, it is a fit candidate for a dynamic programming solution.

Let $S(j)$ denote the maximum sum of the elements in a contiguous subsequence ending at the j-th location of the array $\mathcal{X}$. So, finding $S(n)$ would give us the desired solution. Now,

$$S(j) = max\{S(j-1) + \mathcal{X}(j), \mathcal{X}(j)\}\ 1 \leq j \leq n;$$

Clearly, this requires $O(n)$ time.

(Q2) Given an undirected graph $G = (V, E)$, design and analyze an efficient algorithm to determine if G contains a cycle of odd length. A cycle of odd length is called an odd cycle. [10]

Use either DFS or BFS.

In DFS, as you go down exploring the DFS tree, label the vertices alternately 0 and 1, i.e. if the parent is 0, mark the child as 1, and vice-versa. Now when you encounter a back edge, just check if it is between vertices of the same label. If they are between the same label, you have an odd cycle. If you do not encounter any such back edge, then there is no odd cycle.

To make the answer complete, prove the following easy statement: *There is an odd cycle in G if and only if there is a back edge between vertices of the same label.*

If you are using BFS, work with the *levels* and argue on a *cross-edge*, i.e., an edge which is not a *tree edge*.

(Q3) Let $G = (V, E)$ be a directed graph in which $E = E_1 \cup E_2$ and $E_1 \cap E_2 = \phi$. The edges in E_1 have weights greater than or equal to zero and the edges in E_2 have weights less than zero. Now to find out the single source shortest path, we do the following. Let w_e be the minimum weight of all edges in E_2. Surely $w_e < 0$. Now we add $|w_e|$ to all the edge weights to make them non-negative. Now we run Dijkstra's shortest path algorithm on this graph with the new edge weights (i.e., all edge weights increased by $|w_e|$) to get the desired shortest path.

Can we do so? Prove or disprove the above statement. [10]

This is a **wrong** statement and cannot be proved.
If we add the same amount of weight to each edge to make it positive, and then run the Dijkstra's algorithm, it does not find the shortest path in the original graph. This is so because paths that use more edges (which could have been negative edges) will be rejected in favour of paths having more weight using fewer edges in the new graph. It is easy to construct such an example.

(Q4) Let $G = (V, E)$ be a strongly connected directed graph with positive edge weights, and $v_0 \in V$ be a particular node. Design and analyse an efficient algorithm to find shortest paths between all pairs of vertices in V that go through v_0. Prove the correctness of your algorithm. [10]

Using the single source shortest path algorithm twice — once on G and once on G^R, the reversed graph with source as v_0. Let us look at the path between u and v through v_0 as $P_1 : u \rightsquigarrow v_0$ and $P_2 : v_0 \rightsquigarrow v$. P_2 can be found out from the shortest path on G and P_1 can be found out from the shortest path on G^R. So, we need to sew together paths of the form u to v_0 and v_0 to v to get paths from u to v that go through v_0 for all pairs $u, v \in V$. The time complexity is $O(|E| \log |V| + |V|^2)$ for running Dijkstra twice and finding the required paths for all pairs $u, v \in V$.