Department of Computer Science and Engineering
End Semester Examination

Course No: CS 213 Course Name: Data Structures and Algorithms
Date: 21/11/2020 Time: 8-00 to 12-30 Marks:50

---

1. The set $\mathcal{S}$ of all non-empty bit strings that do not contain two consecutive occurrences of 1 can be defined as follows:

(i) The strings 0, 1, 01 are in $\mathcal{S}$.

(ii) If s is any string in $\mathcal{S}$ then $s.0$ is a string in $\mathcal{S}$.

(iii) If s is any string in $\mathcal{S}$ then $s.01$ is a string in $\mathcal{S}$.

Here . denotes concatenation. Consider the functions f and g defined as follows:

a)	$f(0) = 1$	$f(1) =$ undefined	$f(01) =$ undefined
b)	$g(1) = 0$	$g(01) = 00$	$g(0) =$ undefined
c)	$f(s.0) = g(s).0$	if $g(s)$ is defined	$= s.1$ otherwise
d)	$f(s.01) = g(s).01$	if $g(s)$ is defined	= undefined otherwise
e)	$g(s.0) = f(s).0$	if $f(s)$ is defined	= undefined otherwise
f)	$g(s.01) = f(s).01$	if $f(s)$ is defined	$= s.0.0$ otherwise

(i) Prove that for all $n \geq 1$, there is exactly one string s_1 of length n in $\mathcal{S}$ such that $f(s_1)$ is undefined and exactly one string s_2 of length n such that $g(s_2)$ is undefined. Describe these strings by giving the positions at which they contain a 1, in terms of n. (3)

(ii) Let s be any string in $\mathcal{S}$ for which $f(s)$ is defined. Prove that $g(f(s)) = s$. Also, if $g(s)$ is defined then prove that $f(g(s)) = s$. (4)

(iii) Let s_2 be the string of length n in $\mathcal{S}$ such that $g(s_2)$ is not defined. Prove that the sequence $s_2, f(s_2), f(f(s_2)), \ldots$ contains all strings in $\mathcal{S}$ of length n, without repetition, and this sequence terminates with the string s_1 for which $f(s_1)$ is not defined. Given a string s of length n in $\mathcal{S}$, give an $O(n)$ time algorithm to find the index at which it occurs in the sequence defined above. (You may need to use the Fibonacci numbers). (6)

2. (i) Consider an array A of size n that contains integers. It is required to fill in the array with arbitrary integer values, possibly negative. There are m constraints given by triples (i, j, C_{ij}), which indicates that the sum of the integers in the subarray $A[i] + \cdots + A[j]$ is at most C_{ij}. It can be assumed that $0 \leq i \leq j < n$, and C_{ij} is an integer. Describe an $O(n + m)$ time algorithm to determine whether the given constraints force the sum of all entries in the array to be bounded, or the sum of all entries can be made arbitrarily large, but still satisfying the given constraints. If the sum is bounded, find the maximum possible value of the sum. (5)

(ii) Suppose now there is an additional constraint that each element in the array must be non-negative and have value at most M. Describe an $O(n+m)$ time algorithm to find the maximum possible sum of the whole array, subject to the given constraints. Assume $C_{ij} \geq 0$, so it is possible to satisfy all constraints, (6)

3. (i) Prove that in any binary tree with $n \geq 1$ nodes, there exists a node with at least $n/3$ and at most $(2n+1)/3$ descendants. Give an example of a binary tree with n nodes, such that every node has either $< (n+1)/3$ or $> 2n/3$ descendants, for infinitely many n. (4)

(ii) A data structure is required to perform the following operations on a fixed binary tree T. The nodes of T are numbered 0 to $n-1$, and each node stores an arbitrary integer. The operations to be performed are to set and get the value stored in any specified node i, and to find a node that is a local minimum. A node is a local minimum if the value stored in it is at most the values in its parent and children, if they exist. Using the previous property of a binary tree, show how to implement this so that set/get take $O(1)$ time and a local minimum can be found in $O(\log n)$ time. You can take $O(n \log n)$ time for pre-processing the tree, though it is possible to do it in $O(n)$ time. (6)

4. (i) An undirected graph is said to *nearly bipartite* if there exists a node v such that the graph obtained by deleting the node v (and edges incident with it) is bipartite. Equivalently, there exists a node v such that every odd cycle in the graph contains v. Describe an $O(n+m)$ time algorithm to test whether a given undirected graph is nearly bipartite. Do this in several steps, each of which takes $O(n+m)$ time. First find a chordless odd cycle, that is an odd cycle in which no two non-consecutive vertices are adjacent. Consider the graph obtained by deleting all nodes in the odd cycle. It must be bipartite. Now consider the edges between the odd cycle and a component of the remaining graph. Consider what restriction it imposes on the node in the odd cycle that must be deleted to make the graph bipartite. Check if there is a node that satisfies the restrictions for all components of the remaining graph. Show how each step can be done in $O(n+m)$ time. (8)
NOTE: You may find a published paper that gives an algorithm for this problem, but you will not get any marks for just copying it. You must follow the steps as described, explain them, and show how each of them can be implemented in $O(n+m)$ time.

(ii) A nearly bipartite graph cannot have two disjoint odd cycles. Give the smallest possible example of an undirected graph without two disjoint odd cycles which is not nearly bipartite. (2)

(iii) For all $k \geq 1$, give an example of an undirected graph that does not contain two disjoint odd cycles, but it cannot be made bipartite by deleting k or fewer nodes. Prove that your graph has the required properties. Hint: Consider an $n \times n$ grid graph for $n > k$. This graph is bipartite, now add a suitable set of edges to it, to get the required example. (6)