INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Stamp / Signature of the Invigilator

EXAMINATION (End Semester)										**SEMESTER (Spring)**			
Roll Number										**Section**		**Name**	
Subject Number	C	S	2	1	0	0	3	**Subject Name**		*Algorithms – I*			
Department / Center of the Student											**Additional sheets**		

Important Instructions and Guidelines for Students

1. You must occupy your seat as per the Examination Schedule/Sitting Plan.
2. Do not keep mobile phones or any similar electronic gadgets with you even in the switched off mode.
3. Loose papers, class notes, books or any such materials must not be in your possession, even if they are irrelevant to the subject you are taking examination.
4. Data book, codes, graph papers, relevant standard tables/charts or any other materials are allowed only when instructed by the paper-setter.
5. Use of instrument box, pencil box and non-programmable calculator is allowed during the examination. However, exchange of these items or any other papers (including question papers) is not permitted.
6. Write on both sides of the answer script and do not tear off any page. **Use last page(s) of the answer script for rough work.** Report to the invigilator if the answer script has torn or distorted page(s).
7. It is your responsibility to ensure that you have signed the Attendance Sheet. Keep your Admit Card/Identity Card on the desk for checking by the invigilator.
8. You may leave the examination hall for wash room or for drinking water for a very short period. Record your absence from the Examination Hall in the register provided. Smoking and the consumption of any kind of beverages are strictly prohibited inside the Examination Hall.
9. Do not leave the Examination Hall without submitting your answer script to the invigilator. **In any case, you are not allowed to take away the answer script with you.** After the completion of the examination, do not leave the seat until the invigilators collect all the answer scripts.
10. During the examination, either inside or outside the Examination Hall, gathering information from any kind of sources or exchanging information with others or any such attempt will be treated as '**unfair means**'. Do not adopt unfair means and do not indulge in unseemly behavior.

Violation of any of the above instructions may lead to severe punishment.

Signature of the Student

To be filled in by the examiner

Question Number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Total**
Marks Obtained											

Marks obtained (in words)	**Signature of the Examiner**	**Signature of the Scrutineer**

CS21003 Algorithms–I, Spring 2018–2019

End-Semester Test

25–April–2019 NR112/211/212/311/312, 02:00pm–05:00pm Maximum marks: 60

Instructions

- Write your answers in the question paper itself. Be brief and precise. Answer all questions.
- If you use any algorithm/result/formula covered in the class, just mention it, do not elaborate. For example, if Adrab's algorithm is covered in the class, just mention that by running Adrab's algorithm on input A, you get some output B. You are not asked to elaborate the steps of Adrab's algorithm. If your task is to generate the input A for Adrab's algorithm and/or to convert its output B to C, using algorithms not covered in the class, write the steps how you do that.
- Do not use plain English to write the steps of your algorithms. Write readable pseudocodes. Only if it is necessary, justify the steps of your pseudocodes in English.

Questions start from the next page.

1. You are given two binary search trees T_1 and T_2 storing m and n keys, respectively. There may be repetitions of keys between the two trees. Your task is to merge the input trees into a single binary search tree T storing the union of the keys from T_1 and T_2 (repeated keys should appear only once). Neither of T_1, T_2, T is needed to be height-balanced. Either propose a worst-case $O(m+n)$-time algorithm to solve this problem, or prove that no such algorithm can be designed. **(10)**

Solution An algorithm with the desired time complexity can be designed.

1. Convert T_1 to a right-skew tree by applying rotations.
2. Convert T_2 to a right-skew tree by applying rotations.
3. At this point, T_1 and T_2 are essentially linked lists with respect to their right-child pointers. Merge the two lists. If a key appears in both the trees, include it only once in the merged list.

Step 1 takes $O(m)$ time, Step 2 $O(n)$ time, and Step 3 $O(m+n)$ time.

2. You are given an array A of n integers, and an integer k in the range $2 \leqslant k \leqslant n$. Your task is to determine whether A contains some element(s) with at least k occurrences in A. For instance, $k = 2$ corresponds to the problem of determining whether A contains duplicate(s), whereas $k = \lfloor n/2 \rfloor + 1$ deals with the problem of finding whether A contains a majority element.

(a) Propose a worst-case $\mathrm{O}(n \log n)$-time algorithm to solve the given problem. **(4)**

Solution Sort A by an optimal sorting algorithm (like merge sort or heap sort). Make a single pass through A to count how many times each element appears.

(b) Propose an expected $\mathrm{O}(n)$-time algorithm to solve the given problem. **(6)**

Solution Use an augmented hash table H that stores, in addition to the keys, how many times each key is inserted. If an element is inserted in H for the first time, the count against that element is set to 1. Later, if the same key is tried to be inserted multiple times, each time we make a successful search for the key in H, and increment the count by one. The hash function you use should map keys to indices in the hash table (but should be independent of the counts).

We insert the elements of A one by one in the augmented hash table (after it is initialized to empty). If the incremented count during an insertion ever reaches the value k, return *Yes*. If all insertions result in counts $< k$, return *No*.

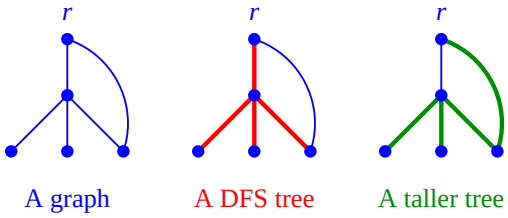
3. You are given a connected undirected graph $G = (V, E)$, and a vertex $r \in V$. The following two parts deal with algorithms for finding spanning trees of G, rooted at r and having specific properties. Prove/Disprove whether the algorithms mentioned below will work always.

(a) **Property** : T should have the minimum possible height among all spanning trees of G, rooted at r.
Algorithm : Run a BFS traversal starting from r, and return the BFS tree. **(5)**

Solution *True.* A BFS traversal from r gives the shortest distance (in terms of numbers of edges) from r to all vertices in G. The height of a tree is defined to be the largest of these shortest distances from r. That is, the height of the BFS tree is the distance of the farthest node from r.

(b) **Property** : T should have the maximum possible height among all spanning trees of G, rooted at r.
Algorithm : Run a DFS traversal starting from r, and return the DFS tree. **(5)**

Solution *False.* A counterexample is provided below. The DFS traversal chooses neighbors from left to right.



Notice that the taller tree can be discovered by another DFS traversal from r. The height of the DFS tree may depend upon the order of choosing the neighbors. That is, the given algorithm is not guaranteed to work always.

4. Let $G = (V, E)$ be a connected undirected graph. A subset $F \subseteq E$ is called a set of *feedback edges* if every cycle of G has one or more edges in F. Clearly, if we remove from G all the edges belonging to a set of feedback edges, the graph becomes acyclic.

(a) Propose an efficient algorithm to compute a set F of feedback edges, containing as few edges (of G) as possible. Mention the running time of your algorithm. **(3 + 1)**

Solution Let T be any spanning tree of G. Take as F the set of all non-tree edges. If we remove any edge e from F and add e to T, a cycle will be formed, implying that $F \setminus \{e\}$ is not a set of feedback edges. Since $(V, E \setminus F)$ is a tree (and so acyclic), F is minimal. Since every spanning tree of G contains exactly $|V| - 1$ edges, every minimal set F of feedback edges is also minimum (in terms of the number of edges).

An efficient algorithm to compute a spanning tree of G is to run a BFS/DFS traversal on G. The running time for this is $O(|V| + |E|) = O(|E|)$ (since G is connected). Preparing the set F consisting of all the non-tree edges (corresponding to the BFS/DFS tree) can also be done in the same time.

(b) In this part, suppose that each edge $e \in E$ has a real-valued cost $c(e)$. Propose an efficient algorithm to compute a set F of feedback edges, such that the total cost $\sum_{e \in F} c(e)$ of F is as small as possible. Mention the running time of your algorithm. **(5 + 1)**

Solution Compute a maximum spanning tree T of G. This can be done by modifying Kruskal's algorithm which sorts the edges in the descending order of their weights. Take as F the set of all the edges of G, that are not in T.

Again, F is a minimal set of feedback edges. Since $\sum_{e \in E} c(e) = \sum_{e \in T} c(e) + \sum_{e \in F} c(e)$ is constant for a given graph G, minimizing $\sum_{e \in F} c(e)$ is equivalent to maximizing $\sum_{e \in T} c(e)$, and the correctness of the algorithm is established.

The running time is $O(|E| \log |E|) = O(|E| \log |V|)$.

5. Let $G = (V, E)$ be a directed graph with each edge $e \in E$ carrying a positive weight $c(e)$. You run the Floyd–Warshall algorithm to compute and store in an $n \times n$ matrix D the shortest distances between every pair of vertices. Then, a new vertex w joins the graph with a specified set of weighted edges from w to (some of) the existing vertices in V and from (some of) the existing vertices in V to w. Assume that positive weights are specified for all the new edges. You need to update D to an $(n+1) \times (n+1)$ matrix D' storing the new shortest distances between the vertices in $V \cup \{w\}$. Propose an $O(|V|^2)$-time algorithm to solve this problem. **(10)**

Solution As usual, assume that $V = \{0, 1, 2, \ldots, n-1\}$, and $w = n$. Perform the following steps.

1. Copy D to the top-left $n \times n$ block of D'.
2. For each $u \in V$, set $D'(u, w) = \min(D'(u, v) + c(v, w))$, where the minimum is taken over all the new edges (v, w) added to G. (The empty minimum is taken to be $+\infty$.)
3. For each $u \in V$, set $D'(w, u) = \min(c(w, v) + D'(v, u))$, where the minimum is taken over all the new edges (w, v) added to G. (The empty minimum is taken to be $+\infty$.)
4. Set $D'(w, w) = 0$.
5. Run one iteration of the Floyd–Warshall algorithm on D' (to allow $w = n$ as an intermediate vertex).

6. Let S and T be strings (over the same alphabet) of lengths n and m, respectively. Assume that $m \leqslant n$. For $r \geqslant 0$, let T^r denote the r-fold concatenation of T with itself. For example, T^0 is the empty string and $T^1 = T$ for any T, $(aba)^4 = abaabaabaaba$, and $(aa)^3 = aaaaaa$. Your task is to find the largest $r \geqslant 0$ such that T^r is a substring of S. Propose a worst-case $O(n)$-time algorithm to solve this problem. **(10)**

Solution The steps of an $O(n)$-time algorithm are given below.

1. Run the KMP algorithm to find all the matches of T in S. From the output of the KMP algorithm, generate an array $M[0 \ldots n-m]$ such that $M[i] = 1$ if T matches $S[i \ldots i+m-1]$, and $M[i] = 0$ otherwise. The KMP algorithm itself may supply M as its output.
2. Initialize an array R to $R[i] = 0$ for all $i \in [0, n-m]$. The array element $R[i]$ is meant to store the largest r such that a match of T^r begins from $S[i]$.
3. For $i = 0, 1, 2, \ldots, n-m$ in that sequence, repeat:
   If $(M[i] = 1)$ and $(R[i] = 0)$, carry out the following steps (else do nothing).
   (a) Initialize $r = 0$ and $k = i$.
   (b) While $(k \leqslant n-m)$ and $(M[k] = 1)$, increment r, and add m to k.
   (c) Reset $k = i$.
   (d) While $(r > 0)$, set $R[k] = r$, decrement r, and add m to k.
4. Compute and return the maximum of $R[0 \ldots n-m]$. If needed, you can additionally return all the positions in R, that store this maximum value.

Step 1 takes $O(n+m) = O(n)$ time. Step 2 requires $O(n-m) = O(n)$ time. The condition $R[i] = 0$ in Step 3 implies that if $R[i]$ is already set to a value $\geqslant 1$, the sub-steps (a)–(d) are not carried out. That is, each $R[i]$ is set only once (in Step 3(d)) or skipped (if a match of T does not begin from $S[i]$). To sum up, Step 3 too runs in $O(n-m) = O(n)$ time. Finally, Step 4 too can be implemented in $O(n-m) = O(n)$ time.