COL 702 Advanced Data Structures and Algorithms

Minor 2, Sem I 2018-19, Max 40, Time 1 hr

Name _____ **Entry No.** _____ **Group**

Note Write in the space provided below the question including back of the page.

Every algorithm must be accompanied by a proof of correctness, time and space complexity. You can however quote any result covered in the lectures without proof.

1. In a skip list $\mathcal{L}$ having n elements, while searching for an element E, we are given a *finger* element $F \in \mathcal{L}$ which is within distance d from E_s, where $E_s \in L$ is a successor of E - it is possible that $E = E_s$.
   For example in the list $3, 4, 7, 14, 19, 21, 25, 30$, and $E = 23$, the finger $F = 7$ is within distance $d = 4$ from $E_s = 25$.
   How can you use the finger so that the expected search time is a function of d ? Describe the procedure and the analysis. Note that only F is given but d is not known. In the standard analysis $d \leq n$. **(10 marks)**

   Although d is not known, we will be able to achieve an $O(\log d)$ bound using the following procedure. Starting with the finger F at the bottom-most level (we are give a pointer) we will look for E_s by examining the next element $succ(F)$ from our current position. If $succ(F) > E$ then we are done.

   Otherwise, we continue until we find an element larger then E or we find an uplink - let us denote such an element by F_1 which is the finger in level 1. No. of expected steps to find an uplink is 2 since every element is moved up with probability 1/2. We repeat the same process in any level till we succeed in finding a $succ(F_i) \geq E$. From this point we climb down like a normal skip tree traversal and locate E.

   So there are two phases - one upwards and the other downwards. In each level, we spend expected $O(1)$ time, so the total search time depends on the number of levels we must climb. For this, think about the part of the skip list between F and E_s that are separated by distance d. Using the same arguments as in class, we can easily argue that we will climb expected $O(\log d)$ levels. You may also focus on the truncated skip list between F and E_s and ignore everything else.

   Some answers described an $O(d)$ sequential search solution for which very little marks were given.

2. Let $G = (V, E)$ be a weighted graph where the weights can be arbitrary real number (i.e. both positive or negative). **(3+7 marks)**
   (i) Can you use the generic-greedy algorithm to find the Minimal Spanning Tree ? You can assume that the graph is connected.

   If there are no negative weights, we apply greedy directly. If there are negative weights, then let w_m denote the least weight and subtract it from all the weights. This makes all weights positive and does not alter the relative rankings of the spanning trees as all of them have the same rank.

   Very little credit was given if the answer did not address this crucial restriction of the matroid theorem. We had addressed the minimization problem by reducing it to maximization but it still worked with non-negative weights

   (ii) Let $w_1, w_2 \ldots w_{n-1}$ be a sequence of numbers in increasing order or weights that corresponds to the edges of the MST $\mathcal{T}_1$. Consider another sequence $w'_1, w'_2 \ldots w'_{n-1}$ of weights in increasing order that corresponds to another MST $\mathcal{T}_2$. Here $\mathcal{T}_1 \neq \mathcal{T}_2$, i.e., all edges are not same but $\sum_i w_i = \sum w'_i$ since both are MSTs. Is it true that the sequences w_i and w'_i are equal for all i or they can be different ? If true, then prove it, else provide a counterexample.

   Suppose the sequences are not identical, then consider the least i such that $w_i \neq w'_i$ and wlog, assume

that $w_i > w'_i$. Consider the set of edges A such these edges have weights $\leq w'_i$. The rank of A is at least i from $\mathcal{T}_2$ - therefore we can find an edge $e \in A$ that doesn't induce a cycle with $e_1, e_2 \ldots e_{i-1}$.

Then add e to the $\mathcal{T}_1$ that will induce a unique cycle and discard the heaviest edge in the cycle, say e'. It follows that

Claim: $e' \in \{e_i, e_{i+1} \ldots e_{n-1}\}$ and so the weight of the new tree $\mathcal{T}_1 \cup \{e\} - e'$ is less than the MST $\mathcal{T}$ - a contradiction.

Most answers correctly identified that the sequences would be identical but the proofs had major fallacies, even after they started on the right track of proving by contradiction. Not enough attention was given to - when you add an edge what are the edges that it could induce a cycle with. This was critical for the proof to work. For such answers, at most 50% marks were awarded.

3. For a given set of universal hash functions, what is the expected number of pairwise collisions when we hash any subset S of n elements in a table of size $cn \log n$ elements ?
Note that if three elements hash to the same locations, they would be counted as 3 pairs. **(5 marks)**

   Since the probability that a pair collides is $\leq 1/m$ from the definition of universal hash function where m is the size of the table, the expected number of pairs that collide is $\binom{n}{2} \cdot \frac{1}{cn \log n} = O(n/\log n)$.

4. Given a set of n jobs J_i $1 \leq i \leq n$ with (integral) deadlines d_i find out if all of them can be scheduled within their deadlines. The running times T_i for the jobs are known to be integers in the range $1..n$. Analyze the efficiency of your algorithm. **(5 marks)**

   Sort the jobs in increasing order of their deadlines let this sorted list be $J'_1, J'_2 \ldots J'_n$. Then add the running time in this order, i.e., $Y_i = T_1 + T_2 \ldots T_i$. Then we can schedule all the jobs within their deadlines iff $d_i \leq Y_i$ for all i where d_i is according to the sorted list.

   To prove it, notice that if the condition is satisfied all jobs before J'_i can be scheduled leaving suffcient time for J'_i. If this condition is not satisfied, consider, the smallest such j where it fails. Since the total requirement for processing before d_j is more than d_j, then there is no feasible schedule.

   If d_i are also integers in polynomial range, then total running time is $O(n)$ else $O(n \log n)$.

   Most answers lacked a proper correctness proof of the scheduling algorithm. You have to justify that when the answer is YES/NO then there is a feasible/no-feasible schedule.

5. A *bottleneck* spanning tree (BST) minimizes the maximum weight edge among all spanning trees of a weighted undirected graph $G = (V, E)$. The *cost* of BST $= \min_{T \in \mathcal{T}}(\max_{e \in T}\{weight(e)\})$ where $\mathcal{T}$ is the set of all spanning trees of G. **(5,5 marks)**

   (a) Which of the following statements are true ? The weight of a BST is the sum of weights of all the edges. (Proof not required - write True or False only)
   Negative 1 for each wrong tick.

   (a) For all graphs, the total weights of BST and MST are equal. (False)
   (b) Every MST is also a BST. (True)
   (c) Every BST is also an MST. (False)
   (d) There is no relation between the weights of MST and BST. (False)

   Every MST is also a BST since, the largest weight in a MST cannot be larger than BST. If you consider all edges less than or equal to the value of of BST, that is also a matroid and a greedy construction will yield an MST.

   (b) Design a linear time algorithm to determine if the BST has cost $\leq b$ for a given b.

   Let E' be the set of edges whose weights are $\leq b$. Construct a spanning tree (using DFS/BFS) on

E'. If it can be done then value is less than b else (if it is not connected) then the value is larger. The time taken is clearly $O(|E| + |E'| + |V|)$ where $E' \subset E$.

Some answers assumed that the BST was given - only the graph is given, otherwise it is a meaningless problem. On the other extreme some answers actually tried to construct a BST which is an overkill. The present problem is a decision problem, we only want a YES/NO answer whether there is a BST with value $\leq b$.