CSL 630 Analysis and Design of Algorithms

Minor II, Sem I 2014 -15 Max 40, Time 1 hr

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

Note (i) The answers must be written in the space provided under each question including the back of the sheet. Do all roughwork in separately provided paper.

(ii) Write your answers neatly and precisely. You won't get a second chance to explain what you have written.

(iii) Every algorithm must be accompanied by proof of correctness and a formal analysis of running time and space bound. Feel free to quote any result from the lectures without proof - for any anything new, you must prove it first.

1. A *bottleneck* spanning tree (BST) minimimizes the maximum weight edge among all spanning trees of a weighted undirected graph $G = (V, E)$. The *value* 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+8+2)**
   (a) Design a linear time algorithm to determine if the BST has value $\leq b$ 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 $E' \subset E$.
   (b) Design an efficient, preferably linear time algorithm for finding a BST.

   Let the number of vertices be n and number of edges be m. We can pick the median weight among m elements in $O(m)$ steps and let this weight be denoted by a_1. Run the previous algorithm using $b = a_1$. If the answer is Yes then the value is $\leq a_1$. Else the value is larger. If it is smaller try with $b =$ the $m/4$-th weight else with $b = 3/4m$-th weight. Thus we are doing a binary search to find the value of the BST, where in stage i, we are constructing a spanning tree of m_i edges, $m_i \leq m$.

   To analyze the running time, first note that we will try to construct a spanning tree only if $m_i \geq n-1$ otherwise, we know that it cannot be connected. Since the spanning tree algorithm (DFS or BFS) is invoked at most $O(\log m) = O(\log n)$ times, the running time is at most $O((m+n)\log n)$.

   To improve this analysis, whenever there is a **No** answer, we retain the forest constructed so far and run the spanning tree treating the forests as a *super vertex*. By careful implementation, we can merge vertices into a single vertex in time proportional to the number of edges involved in each stage. Rename all the vertices in a spanning tree and then relabel the end-points of all the edges (this is not union find since we are merging the whole spanning tree). This way we do not repeat the work on the edges and the total time is $T(m) = T(m/2) + O(m) = O(m/2 + m/4 + \ldots = O(m)$.
   (c) Which of the following statements are true ? The weight of a BST is the sum of weights of all the edges.
   Negative 1 for each wrong tick.

   (a) For all graphs, the weights of BST and MST are equal.

   (b) Every MST is also a BST.

   (c) Every BST is also an MST.

   (d) There is no relation between the weights of MST and BST.

   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.

2. In a knapsack problem with n items, the profits are integers in the range $[1 \ldots n^2 \log n]$ but the capacity of the knapsack may be very large (larger than a polynomial). Design an efficient polynomial time algorithm for this version of the knapsack problem. **(15)**

Let p_i, w_i denote the profits and weights of the i-th item for $1 \leq i \leq n$. Let $P = \sum_i p_i \leq n^3 \log n$ which is a polynomial. Let $Q(i,j)$ = minimum weight of items among $x_1, x_2 \ldots x_i$ to obtain a profit of j and ∞ otherwise (if we cannot obtain a profit j). Then the solution of the knapsack problem is $argmax_j\{Q(n,j) \leq B\}$, where B is the capacity of knapsack. We define a recurrence as follows

$$Q(i,j) = \min\{Q(i-1,j), w_i + Q(i-1, j-p_i)$$

The base case $Q(1,j) \quad j \leq P$ is easy to compute. Thus this can be solved using DP using a $n \times P$ table where each entry can be computed in $O(1)$ time in increasing order of rows. The space bound can be improved to $O(P)$ since we need only the previous row.

3. Show that by using universal hash functions, we can hash any subset S of n elements in a table of size αn^2 elements for an appropriate constant α such that there are no collisions between the elements of S.
   Hint: It suffices to show that the probability of no collisions is greater than some constant, say 1/10.
   (10)

   For elements $x, y \in S$, the universal hash function guarantees that the probablity that x, y collide for a random choice of a hash function is $\frac{c}{m}$ where c is a constant and m is the size of the table. So the expected number of collisions for all the $\binom{n}{2}$ pairs of S is $\frac{cn\cdot(n-1)}{2m}$. For $m \geq cn^2$ this is $< 1/2$. So, from Markov's inequality, the probability that it is < 1 is greater than $1/2$. So with probability at least $1/2$ there are no collisions.
   We can choose a random hash function and if there is any collision then we can choose independently another random hash function - the expected number of repetitions is 2.