**CSL 356 Analysis and Design of Algorithms**

Minor II, Sem I 2013-14, Max 40, Time 1 hr

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

**Note** (i) Write your answers in the blank sheets 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) Unless stated otherwise, 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. Write answers to the following problems (no proof or algorithm required)

   (a) Let $T$ be an MST of a graph $G = (V, E)$. Let $G' = (V, E')$ be a subgraph of $G$ , i.e. $E' \subset E$. Then $T \cap E'$ is an MST of $G'$. (**Yes/No**) No (even if it remains connected) **(3 )**

   (b) Let $T$ be a minimum spanning tree output by the basic greedy algorithm where the weight of a edge $e_i$ is given by $w_i$. Suppose we change the weights to $w'_i = a \cdot w_i + b$ where $a, b > 0$ are reals numbers. Does $T$ still remain the minimum spanning tree with the modified weights ? (**Yes/No**) Yes - multiplying with a positive number does not change the MST since every spanning tree changes by the same factor. Adding a fixed number (even negative) also keeps the relative weight the same because all maximal subsets have the same rank. **(3 )**

   (c) In a job scheduling problem with deadlines, where each job takes unit time, can you schedule all the following 12 jobs without missing their respective deadlines ? **(2+3 )**
   (i) (Deadlines): 6, 3, 8, 4, 3, 9, 10, 7, 4, 3, 2, 3 (**Yes/No**) No

   (ii) If the deadlines of a given set of $n$ jobs are integers in $[1, 2n]$, what is the best time complexity for verifying if all the jobs can be scheduled within their respective deadlines.
   Sort the deadlines in increasing order - sorting can be done in $O(n)$ time. The $i$-th job (in sorted order) can be scheduled without penalty if and only if the deadline is $\geq i$.

   (d) Consider the family of hash functions $h_a(x) : x \to ((a \cdot x) \mathrm{mod} N) \mathrm{mod} m$ $a \neq 0$ where $a, x \in \{0, 1, \ldots N-1\}$? Here $N$ is prime and denotes the size of universe and $m$ is the size of the hash table. **(1+3 )**
   (i) What is the total number of hash functions ?
   $N - 1$.

   (ii) Is the family Universal ?(Note that this different from the one done in class) (**Yes/No**) Yes. If $h_a(x) = h_a(y), x \neq y$ then $a(x - y) \equiv_N t$ where $t$ is a multiple of $m$, namely, $\{0, +m, -m, +2m - 2m \ldots + (N-1)/m \cdot m\}$. For each $t$, there is a distinct solution $a \equiv (x-y)^{-1} \cdot t$. Therefore there are at most $2(N-1)/m + 1$ solutions. In conjunction with part (i), it is 2-universal.
   **If the first part is incorrect then no credit for the second part**.

2. We have a set of $n$ variables $x_1, x_2 \ldots x_n$ and a set of $k$ relations of the form $x_i = x_j$ or $x_\ell \neq x_t$. We want to determine if this is consistent. For example, $\{x_1 = x_2,\ x_1 = x_3, x_2 \neq x_3\}$ is not consistent. Whereas, $\{x_1 = x_2$,$x_2 = x_3,\ x_1 \neq x_4\}$ is consistent. **(2+8 )**
(i) What is a lower bound for the running time for this problem ?

It is $\Omega(k)$ from the size of input - marks has been deducted for use of $O$ instead of $\Omega$.
I have also accepted $\Omega(n+k)$ as a valid lower bound since the union find approach has a dependence on $n$ unless implemented very carefully.
(ii) Design an efficient algorithm for this problem that is within a factor of $O(\log n)$ of the lower bound, and the closer, the better.

The equalities can be thought of as equivalence classes. We can model it as a graph $G = (V, E)$ where each vertex corresponds to a variable and we have an edge $(v_i, v_j)$ corresponding to an equality $x_i = x_j$. So the connected components correspond to the equivalence classes. The set of inequalities is consistent iff no two vertex in a connected component say $u, v$ are related by $u \neq v$. Corresponding to each such inequality, we *Find* if $C(u) \neq C(v)$ where $C()$ is the component number.
This can be done using appropriate data structures.
(i) We compute the connected components (using DFS or BFS) in $O(n + k)$ time (the maximum number of edges is $k$). Then corresponding to each component we label the vertices - doable in $O(n)$ time and then subsequently test the inequalities in additional $O(k)$ time. So total is $O(n + k)$ that matches the lower bound.
*To get rid of n from the running time, you have to first use hashing to reduce te range of labels from* $[1..n]$ *to* $[1..2k]$. *This can be done in* $O(k)$ *expected time using universal hashing. Without this step, you will incur an additional* $O(n)$ *for initialization of the Union Find or the solution given above.*
(ii) You can use Union-Find to do this although the ordering of the relations is important - must process the "=" before $\neq$. This gives $O((n + k) \log^* n)$ using the fastest heuristics or even $O((n + k)logn)$ using the simpler ones. I have awarded about 6 marks for this version and 4-5 marks for the $O((n + k) \log n)$

3. Given a $4 \times n$ board (4 rows, $n$ columns) with a positive number on each square, we want to place **upto** $2n$ coins so as to maximize the sum of the numbers in the squares where the coins are placed. Additionally we cannot have two coins in vertically or horizontally adjacent squares (diagonally no restriction).
Design an $O(n)$ algorithm for this problem based on Dynamic Programming. Only Partial credit for polynomial time algorithm that is not $O(n)$. **(15 )**

**Formal definition of subproblems, Dynamic programming recurrence with justification**

Each column can contain at most two coins because of the adjacency restriction. Each possible placement can be thought of as a binary string, i.e., 1001 implies that coins are placed in rows 1 and 4. The feasible options are 0000, 1010, 0101, 1001, 1000, 0100, 0010, 0001, namely 8 (some constant $C$). The optimum solution can be constructed as choosing one of the $C$ options in the last column and then placing the remaining coins in columns 1 to $n-1$ such that column $n-1$ is *compatible* with the last column. Let $P(i, j)$ denote the optimal solution for $i..n$ columns beginning with pattern $j$, where $1 \leq i \leq n$ and $j \leq C$. Then

$$P(i, j) = \max_{j' \in A(j)} \{P(i + 1, j') + C_i(j')\}$$

where $A(j)$ denotes the columns compatible with $j$ and $C_i(j')$ is the profit corresponding the the pattern $j'$ in $i$-th column. Clearly $P(n, j)$ is fixed and we need the solution of $\max_j P(1, j)$ where

$j \leq C$.

**Time and space analysis** The number of columns compatible with a fixed (feasible) column is $\leq C$. We compute the entries of the $n \times C$ table starting with the last row and compute row-wise in decreasing order of rows. Each entry takes $O(C)$ steps to be computed, so the overall computation takes $O(nC^2)$ which is $O(n)$ since $C$ is fixed. Similarly space is also $O(n)$.

Note that the optimal solution as well as the actual placements can be computed simultaneously.

Full credit has been awarded only when the DP recurrence has been written correctly capturing the constraints of placement considering the different cases.

About 10 marks have been awarded for coming up with the right approach, even if the DP recurrence was not worked out precisely.