Major Exam (COL 702)

Read the instructions carefully:

- You need to justify correctness and running time of each algorithm.
- If using dynamic programming, you must explain the meaning of table entries, and explain the order of computing them.
- No argument should use examples – they will be ignored.
- You can assume any result proved during the lectures (but cannot assume any other result which is in the book or elsewhere).

1. For each of these problems, state whether they are true or false, and give a justification for your answer. There is no negative marking, but you will get marks only if the justification is correct.

   (i) (**2 marks**) If a decision problem in $\mathcal{P}$ is NP-complete, then $\mathcal{P} = NP$. Recall that $\mathcal{P}$ denotes the decision problems which have polynomial time algorithms.

   Solution: This is true. Suppose $L \in \mathcal{P}$ is NP-complete. Now, let L' be any problem in NP. Then, by definition of NP-completeness, $L' \leq_p L$. Therefore, if L has a polynomial time algorithm, then L' also has a polynomial time algorithm. This implies that all problems in NP have polynomial time algorithms.

   (ii) (**2 marks**) Let G be a connected undirected graph. We say that a vertex x is a safe vertex in G if removing x does not disconnect G. A vertex x is a safe in G if and only if the DFS tree formed by running DFS from x results in a DFS tree such that the root (i.e., x) has only one child.

   Solution: This is true. We have to prove both directions. Suppose x has only one child in the DFS tree T. Then if we remove x from T, T stays connected, and since T is only a subgraph of G, G will also stay connected when x is removed. Conversely, suppose x has two or more children. Because there are no edge of G is a cross edge with respect to T, once we remove x, the children of x will lie in different connected components.

   (iii) (**2 marks**) If $f(n) = O(g(n))$ then $2^{f(n)}$ is $2^{O(g(n))}$.

   Solution: This is also true. If $f(n) = O(g(n))$, then $f(n) \leq cg(n)$ for some constant c. Therefore, $2^{f(n)} \leq 2^{cg(n)}$.

2. (**5 marks**) You are given a set S of $n+1$ distinct numbers (which may not be integers). You can assume that S is given as an array (need not be sorted). You are also given an unsorted array A of size n containing exactly n out of the $n+1$ numbers in S. Give an $O(n)$ time divide and conquer algorithm to find the number from S which is not in A. The only operation allowed on numbers in S (or A) is comparison (you are NOT allowed to perform addition, subtraction, multiplication, etc. on these numbers). **Solution:** We use divide and conquer strategy. First find the median of A in $O(n)$ time. Call this x. Let A_L be the numbers in A which are less than or equal to x and S_L be the numbers less than or equal to x in S. Define A_R and S_R similarly. Now, either $|A_L| = |S_L| - 1$, or $|A_R| = |S_R| - 1$. Depending on the case, we proceed recursively. Since finding median takes $O(n)$ time, we get the recurrence $T(n) = T(n/2) + O(n)$, whose solution is $T(n) = O(n)$. If you used randomized version (like quick-sort), that is ok, but get partial marks.

3. A shuffle of two strings X and Y is formed by interspersing the characters into a new string, keeping the characters of X and Y in the same order. For example, the strings PRODGYRNAMAMMIINCG and DYPRONGARMAMMICING are both shuffles of DYNAMIC and PROGRAMMING.

   (a) (**5 marks**) Given 3 strings $A[1..n], B[1..m], C[1..(n{+}m)]$, give a dynamic programming algorithm to determine whether C is a shuffle of A and B (the algorithm outputs a boolean value only).

   Solution: We maintain a table $T[i,j]$ which is true if $C[1..(i+j)]$ is a shuffle of $A[1..i]$ and $B[1..j]$; otherwise it is false. The recurrence is as follows: if $C[i+j]$ is equal to both $A[i]$ and $B[j]$, then $T[i,j]$ is true if either $T[i,j-1]$ or $T[i-1,j]$ is true; otherwise it is false (many of you have made a mistake, you are saying that if $C[i+1]$ is equal to $A[i]$, then set it to $T[i-1,j]$, this is wrong). Otherwise if $C[i+j]$ is equal to $A[i]$, then this table entry is $T[i-1,j]$. If $C[i+j]$ is equal to $B[j]$ only, then this table entry is $T[i,j-1]$. If none of these cases happen, then $T[i,j]$ is false.

   The for loop now has to go from $i = 1$ to n, and $j = 1$ to n.

   (b) (**4 marks**) A smooth shuffle of X and Y is a shuffle of X and Y that never uses more than two consecutive symbols of either string. For example, PRDOYGNARAMMMIICNG is a smooth shuffle of the strings DYNAMIC and PROGRAMMING, but DYPRONGARMAMMICING is not. Describe and analyze an algorithm to decide, given three strings $X[1..n], Y[1..m], Z[1..(n+m)]$, whether Z is a smooth shuffle of X and Y.

   Solution: We make two tables $T[i,j]$ and $S[i,j]$ where $T[i,j]$ is true if and only if $C[1..(i+j)]$ is a smooth shuffle of $A[1..i]$ and $B[1..j]$ with the condition that $C[i+j]$ must match with $A[i]$. Similarly, $S[i,j]$ is true if and only if $C[1..(i+j)]$ is a smooth shuffle of $A[1..i]$ and $B[1..j]$ with the condition that $C[i+j]$ must match with $B[j]$. Our final answer would be true if either $T[n,m]$ or $S[n,m]$ is true. Let us see how to fill the table entries. Consider $T[i,j]$ (the case of $S[i,j]$ is similar). If $C[i+j]$ is not equal to $A[i]$, then $T[i,j]$ is false. So assume $C[i+j] = A[i]$. If $C[i+j-1]$ is not equal to $A[i-1]$, then $T[i,j] = S[i-1,j]$. If $C[i+j-1] = A[i-1]$, then $T[i,j]$ is true if either $S[i-1,j]$ or $S[i-2,j]$ are true.

   For loop should be shown as well.

4. (**5 marks**) A town has f families, and k clubs. The i^{th} club needs a_i members, but can take at most 3 members from any family. Further, each person can belong to at most 1 club. Assume that the j^{th} family has b_j members. Show how the problem of assigning people to clubs while satisfying the above constraints can be expressed as a maximum flow problem.

   Solution: We make a graph with a vertex v_j for each family j and a vertex w_i for each club i. We also add special vertices s, t. We draw a directed edge (v_j, w_i) for all families j and clubs i, of capacity 3. For each family j, we have an edge (s, v_j) of capacity a_i, and for each club i, we have an edge (w_i, t) of capacity a_i. Finally, we find a max flow from s to t and check if this max-flow has value $\sum_i a_i$ – if not, there is no solution.

   First suppose there is a solution which assigns a_i people to each club i. Let x_{ij} be the number of members of family j assigned to club i, and let x_j denote $\sum_i x_{ij}$. Then we send x_j flow on (s, v_j) edge, x_{ij} flow on (v_j, w_i) edge and a_i flow on (w_i, t) edge. This forms a flow of value of $\sum_i a_i$.

   Conversely, suppose there is a flow of value $\sum_i a_i$. It must be saturating all the edges (w_i, t), i.e., sending a_i amount of flow on this edge. Now, we use integrality of max-flow (this is important, most students have not mentioned this), and so the flow on each (v_j, w_i) edge is an integer between 0 and 3. Further the flow an any edge (s, v_j) is at most b_j, and the flow conservation at v_j shows how the family members of family j are split across clubs.

5. (**5 marks**) We say that a set S of vertices in an undirected graph G forms a **near-clique** if there are edges between every pair of vertices in S, except perhaps for one pair – so a near-clique on k vertices will have either $\binom{k}{2}$ edges (in which case, it will be a clique) or $(\binom{k}{2} - 1)$ edges. Given a graph G and parameter $k > 0$, we would like to decide if the graph has a near-clique of size k – call this the NEAR-CLIQUE problem. Prove that the CLIQUE problem is polynomial time reducible to the NEAR-CLIQUE problem (recall that in the CLIQUE problem, we are given a graph G and a parameter k, and would like to decide if G has a clique of size k).

   Solution: Consider an instance of the clique problem – G, k. We produce an instance of the near-clique problem as follows: we add two new vertices a_0 and a_1 to the graph G. Further, we connect all vertices in G to these two vertices by edges (but we do not have an edge between a and b) – call this graph G'. The instance for the near-clique problem is $G', k+2$. Suppose G has a clique of size k. Then the vertices in these clique along with a_0 and a_1 form a near clique of size $k+2$ in G'. Conversely, suppose there is a near clique of size $k+2$ in G' – call these vertices S. There is a subset S' of size $k+1$ of S such that we have edge between every pair of vertices in S'. S' cannot contain both a and b – and so, at least k of the vertices in S' are also present in G, and so, we have a clique of size k in G.

   Many students reduced in the other direction, i.e., from near clique to clique, this is wrong.

6. (**5 marks**) You are given two sets X and Y of n positive integers each. You are asked to arrange the elements in each of the sets X and Y in some order. Let x_i be the i^{th} element of X in this order, and define y_i similarly. Your goal is to arrange them such that $\Pi_{i=1}^{n} x_i^{y_i} = x_1^{y_1} \times x_2^{y_2} \times \cdots \times x_n^{y_n}$ is maximized. Give an efficient greedy algorithm to solve this problem.

   Solution: First sort them in decreasing order. So assume $x_1 \geq x_2 \geq \ldots \geq x_n$ and $y_1 \geq y_2 \geq \ldots \geq y_n$. Now pair up x_i with y_i. In order to prove optimality, we want to argue that there is an optimal solution which pairs x_1 with y_1.

   So let O be an optimal solution and suppose it pairs (x_1, y_k) and (x_j, y_1), where $k \neq 1$. Now

$$x_1^{y_k} x_j^{y_1} = x_1^{y_k} x_j^{y_1 - y_k} x_j^{y_k} \leq x_1^{y_k} x_1^{y_1 - y_k} x_j^{y_k} = x_1^{y_1} x_j^{y_k}.$$

   Thus by pairing (x_1, y_1), we can only increase the value of the product, and so, there is an optimal solution which forms the pair (x_1, y_1). Now proceed by induction for the rest of the argument.