Major Exam (CSL 356/ COL 351)

Read the instructions carefully:

- In all your algorithms, you must give argument for correctness, and mention the running time.
- The algorithm should be explained in English (you can use pseudocode, but it should be easy to understand). Do NOT use examples/figures to explain your algorithm. We will NOT read any such explanation. If using dynamic programming, you must mention the meaning of table entries and recurrence in English.
- The proofs should be brief and statements in the proof should follow a logical sequence. Do NOT use examples or special cases to explain the proof. We will NOT read any such part of the proof.
- For NP-completeness reduction, you must use any of the following NP-complete problems: 3-SAT, Vertex Cover, Independent Set, Clique, Subset Sum, Hamiltonian Cycle.
- Each question carries 10 marks.

1. You are given two strings $S = s_1, \ldots, s_n$ and $T = t_1, \ldots, t_m$. Assume that no character appears in both the strings. You are also given a cost $C(s_i, t_j)$ for every pair of characters in S and T. You would like to merge the two strings into a new string Z of length $|S| + |T|$ such that the order of the characters in S remains unchanged in Z and similarly for T. The cost we pay is the following : whenever we have two characters s_i, t_j adjacent in Z we pay $C(s_i, t_j)$. The goal is to find a string Z which minimizes the total cost. Give an efficient algorithm for this problem.

   Example : Suppose $S = ABAC, T = 1241$. Now, we can merge them as $AB1A24C1$ or $A1B241AC$ because ordering of the characters in S or T is not changed in the merged string. However, $AB124C1A$ is not valid because the ordering of characters in S is not preserved. Now the cost of the string $AB1A24C1$ is $C(B,1) + C(A,1) + C(A,2) + C(C,4) + C(C,1)$.

2. You have a collection of n software applications $\{1, \ldots, n\}$ running on an old computer system. You would like to port these to a new system. If you move application i to the new system, you will get benefit of b_i – the quantity b_i can be a positive or a negative integer. If it is positive, it means that you get an advantage of b_i by moving the application to the new system, whereas if it is negative, then it means that the old system was better suited for this application. Further, for every pair of applications i and j, you are given a non-negative integer x_{ij}, which denotes the expense of moving exactly one of these applications to the new system (and keeping the other one in the older system). Give an efficient algorithm which outputs the set of applications which should be moved to the new system such that the net benefit is maximized – note that if you move a subset S of applications to the new system, then the net benefit can be written as

$$\sum_{i \in S} b_i - \sum_{i \in S, j \notin S} x_{ij}.$$

3. You are given a directed graph G with edge capacities (which are positive integers), a source s and a sink t. You are also given a maximum s-t flow in the graph, i.e., for every edge e, you know f_e, where f is a max-flow from s to t. Now, we change the capacity of an edge e from c_e to $c_e - 1$ (the capacity of other edges remains unchanged). Give an algorithm to find a max-flow from s to t in this new graph. The running time of your algorithm should be linear in the size of the graph.

① Use dp, dp[i][j][k] answer for s[:i], t[:j] if

k==0 → beginning with s, k==1 beginning with t.

$$dp[i][j][0] = \min(dp[i+1][j][0] + C(s_i, t_j) + dp[i+1][j][1])$$

$$dp[i][j][1] = \min(dp[i][j+1][1] + C(s_i, t_j) + dp[i][j+1][0])$$

O(mn)

4. Recall the fractional knapsack problem: you are given a knapsack of size B and n items. The i^{th} item has size s_i and profit p_i. You would like to select a subset of these items such that their total size is at most B and their total profit is maximised. In the fractional knapsack problem, the items are divisible, and so you can take a fraction of an item. If you take fraction f_i of item i, its size will be $f_i s_i$, and the profit would be $f_i p_i$. Give a linear (in number of items) time algorithm to find the optimal solution. You can assume that all floating point arithmetic operations (e.g., addition, division, comparison, multiplication) up to arbitrary precision can be done in constant time.

5. We say that a set S of vertices in an undirected graph G form 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 at least k. Prove that this problem is NP-complete.

6. Consider the optimization version of the subset-sum problem: you are given a set of numbers $x_1, \ldots, x_n$, and a number B. A subset S of these numbers is said to be **valid** if the total sum of the numbers in S is at most B, i.e., $\sum_{x_i \in S} x_i \leq B$. You would like to find a valid set such that the total sum of numbers in S is maximized.
Example. If the numbers are $2, 8, 4$, and B $= 11$, then the optimal solution is the subset $S = \{2, 8\}$.

   (a) Consider the following algorithm for this problem:

   Initialize S to emptyset and T to 0 (T is equal to the sum of the numbers in S).
   For $i = 1, \ldots, n$
      if $T + x_i \leq B$, then add x_i to S and update T to $T + x_i$.
   Output S.

   Give an example in which the total sum of the set S returned by this algorithm is less than half the total sum of the optimal solution.

   (b) Give a polynomial-time approximation algorithm for this problem with the following guarantee: It returns a valid set S whose total sum is at least half as large as the maximum total sum of any valid set. You should prove why your algorithm has this property.