COL351: Analysis and Design of Algorithms **Fall 2024**

Tutorial Sheet 9

Announced on: Sept 26 (Thurs)

- This tutorial sheet contains problems in dynamic programming (DP). When presenting your solutions, please define the DP table clearly. Don't forget to write the base case. The correctness and running time arguments can be brief (1-2 sentences).
- Problems marked with ($\star$) will not be asked in the tutorial quiz.

1. The SUBSET SUM problem is defined as follows: We are given as input n nonnegative integers $a_1, \ldots, a_n$ and a target integer T. The goal is to decide whether there exists a subset $S \subseteq \{a_1, a_2, \ldots, a_n\}$ such that the entries of S add up to exactly T. If the answer is YES, your algorithm must return the set S. Design an $\mathcal{O}(nT)$ time and $\mathcal{O}(T)$ space algorithm for this problem. Briefly justify your algorithm's correctness, running time, and space requirements.

2. We are given a checkerboard that has four rows and n columns and has an integer written in each square. We are also given a set of $2n$ pebbles, and we want to place some or all of these on the checkerboard (each pebble can be placed on exactly one square) to maximize the sum of the integers in the squares that are covered by pebbles. For a placement of pebbles to be valid, no two of them can be on horizontally or vertically adjacent squares (diagonal adjacency is allowed). Give an $\mathcal{O}(n)$ time algorithm to find an optimal placement of the pebbles.

3. Let A be an $n \times n$ bit matrix, that is, every entry $A[i, j]$ is either 0 (white) or 1 (black). A submatrix $A[i_1 : i_2, j_1 : j_2]$ is a *black square* of size s if (a) $s = i_2 - i_1 + 1 = j_2 - j_1 + 1$ and (b) $A[i, j] = 1$ for all $i_1 \leqslant i \leqslant i_2$ and $j_1 \leqslant j \leqslant j_2$. Design an $\mathcal{O}(n^2)$ time algorithm to find the largest black square submatrix of A.

4. Consider the following inventory problem: You are running a store that sells some large product (let us assume you sell trucks), and predictions tell you the number of sales to expect over the next n months. Let d_i denote the number of sales you expect in month i. We will assume that all sales happen at the beginning of the month, and trucks that are not sold are stored until the beginning of the next month. You can store at most S trucks, and it costs C to store a single truck for a month. You receive shipments of trucks by placing orders for them, and there is a fixed ordering fee of K each time you place an order (regardless of the number of trucks you order). You start with no trucks. The problem is to design an algorithm that decides how to place orders so that you satisfy all the demands d_i and minimize the costs. In summary:

- There are two parts to the cost. First, storage: It costs C for every truck on hand that is not needed that month. Second, ordering fees: every order placed costs K.
- In each month, you need enough trucks to satisfy the demand d_i, but the amount left over after satisfying the demand for the month should not exceed the inventory limit S.

Give an algorithm that solves this problem in time that is polynomial in n and S.

5. This problem describes four generalizations of the knapsack problem. In each, the input consists of item values $v_1, v_2, \ldots, v_n$, item sizes $s_1, s_2, \ldots, s_n$, and additional problem-specific data (all positive integers). Which of these generalizations can be solved by dynamic programming in time polynomial in the number n of items and the largest number M that appears in the input? Mention all options that apply and provide a brief justification for your selection.

   a) Given a positive integer capacity C, compute a subset of items with the maximum possible total value subject to having total size *exactly* C. (If no such set exists, the algorithm should correctly detect that fact.)

   b) Given a positive integer capacity C and an item budget $k \in \{1, 2, \ldots, n\}$, compute a subset of items with the maximum possible total value subject to having total size at most C and *at most k items*.

   c) Given capacities C_1 and C_2 of two knapsacks, compute disjoint subsets S_1, S_2 of items with the maximum possible total value $\sum_{i \in S_1} v_i + \sum_{i \in S_2} v_i$ subject to the knapsack capacities $\sum_{i \in S_1} s_i \leqslant C_1$ and $\sum_{i \in S_2} s_i \leqslant C_2$.

   d) Given capacities $C_1, C_2, \ldots, C_m$ of m knapsacks, where m could be as large as n, compute disjoint subsets $S_1, S_2, \ldots, S_m$ of items with the maximum possible total value $\sum_{i \in S_1} v_i + \sum_{i \in S_2} v_i + \cdots + \sum_{i \in S_m} v_i$, subject to knapsack capacities $\sum_{i \in S_1} s_i \leqslant C_1$, $\sum_{i \in S_2} s_i \leqslant C_2, \ldots,$ $\sum_{i \in S_m} s_i \leqslant C_m$.

6. The following problems all take as input two strings X and Y, with lengths m and n, over some alphabet Σ. Which of them can be solved in $\mathcal{O}(mn)$ time? Mention all options that apply and provide a brief justification for your selection.

   a) Consider the variation of sequence alignment in which, instead of a single gap penalty α_{gap}, you are given two positive numbers a and b. The penalty for inserting $k \geqslant 1$ gaps in a row is now defined as $ak + b$, rather than $k \cdot \alpha_{\text{gap}}$. The other penalties (for mismatching two symbols) are defined as before. The goal is to compute the minimum possible penalty of an alignment under this new cost model.

   b) Compute the length of the longest common subsequence of X and Y. (A *subsequence* need not comprise consecutive symbols. For example, the longest common subsequence of "abcdef" and "afebcd" is "abcd".)

   c) Assume that X and Y have the same length n. Determine whether there exists a permutation f, mapping each $i \in \{1, 2, \ldots, n\}$ to a distinct value $f(i) \in \{1, 2, \ldots, n\}$, such that

$X_i = Y_{f(i)}$ for every $i \in \{1, 2, \ldots, n\}$.

d) Compute the length of the longest common substring of X and Y. (A *substring* is a subsequence comprising consecutive symbols. For example, "bcd" is a substring of "abcdef", while "bdf" is not.)

7. ($\star$) We are given a set of points $P = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ in the two-dimensional Euclidean plane, with $x_1 < x_2 < \cdots < x_n$. We will use p_i to denote the point (x_i, y_i).

Given a line L defined by the equation $y = ax + b$, we say that the error of L with respect to P is the sum of its squared "distances" to the points in P, i.e.,

$$\texttt{error}(L, P) = \sum_{i=1}^{n} (ax_i + b - y_i)^2.$$

Using calculus, it can be shown that the line of best fit is given by:

$$a = \frac{n \sum_i x_i y_i - (\sum_i x_i)(\sum_i y_i)}{n \sum_i x_i^2 - (\sum_i x_i)^2} \text{ and } b = \frac{\sum_i y_i - a \sum_i x_i}{n}.$$

Our goal in this problem is twofold: First, we want to partition P into some number of *segments*. A segment is a subset of P that represents a contiguous set of x-coordinates; that is, it is a subset of the form $\{p_i, p_{i+1}, \ldots, p_{j-1}, p_j\}$ for some indices $i \leqslant j$. Second, for each segment S in our partition of P, we compute the line minimizing the error with respect to the points in S, according to the formulas above.

The penalty of a partition is defined to be a sum of the following terms.

a) The number of segments into which we partition P, times a fixed, given multiplier $C > 0$.

b) For each segment, the error value of the optimal line through that segment.

Design an efficient algorithm to find a partition of minimum penalty.