COL 351

TUTORIAL 6

1. **[KT-Chapter 6]** Suppose we want to replicate a file over a collection of n servers, labeled $S_1, S_2, ..., S_n$. To place a copy of the file at server S_i results in a placement cost of c_i, for an integer $c_i > 0$. Now, if a user requests the file from server S_i, and no copy of the file is present at S_i, then the servers $S_{i+1}, S_{i+2}, S_{i+3}, \ldots$ are searched in order until a copy of the file is finally found, say at server S_j, where $j > i$. This results in an access cost of $j - i$. (Note that the lower-indexed servers $S_{i-1}, S_{i-2}, ...$ are not consulted in this search.) The access cost is 0 if S_i holds a copy of the file. We will require that a copy of the file be placed at server S_n, so that all such searches will terminate, at the latest, at S_n. We would like to place copies of the files at the servers so as to minimize the sum of placement and access costs. Formally, we say that a configuration is a choice, for each server S_i with $i = 1, 2, ..., n-1$, of whether to place a copy of the file at S_i or not. (Recall that a copy is always placed at S_n.) The total cost of a configuration is the sum of all placement costs for servers with a copy of the file, plus the sum of all access costs associated with all n servers. Give a polynomial-time algorithm to find a configuration of minimum total cost.

2. **[Dasgupta, Papadimitriou, Vazirani -Chapter 6]** You are given a string of n characters $s[1...n]$, which you believe to be a corrupted text document in which all punctuation has vanished (so that it looks something like "itwasthebestoftimes..."). You wish to reconstruct the document using a dictionary, which is available in the form of a Boolean function dict(): for any string w, dict(w) outputs true if w is a valid word false otherwise. Give a dynamic programming algorithm that determines whether the string $s[]$ can be reconstituted as a sequence of valid words. The running time should be at most $O(n^2)$, assuming each call to dict() takes unit time.

3. **[KT-Chapter 6]** We are given a checkerboard which has 4 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) so as to maximize the sum of the integers in the squares that are covered by pebbles. There is one constraint: for a placement of pebbles to be legal, no two of them can be on horizontally or vertically adjacent squares (diagonal adjacency is fine). Give an $O(n)$ time algorithm to find an optimal placement of the pebbles.

4. **[Dasgupta, Papadimitriou, Vazirani -Chapter 6]**Suppose you are given n words $w_1, \ldots, w_n$ and you are given the frequencies $f_1, \ldots, f_n$ of these words. You would like to arrange them in a binary search tree (using lexicographic ordering) such that the quantity $\sum_{i=1}^{n} f_i h_i$ is minimized, where h_i denotes the depth of the node for word w_i in this tree. Give an efficient algorithm to find the optimal tree.

5. [**Dasgupta, Papadimitriou, Vazirani -Chapter 6**] Consider the following 3-PARTITION problem. Given integers $a_1, \ldots, a_n$, we want to determine whether it is possible to partition of $\{1, \ldots, n\}$ into three disjoint subsets I, J, K such that

$$\sum_{i \in I} a_i = \sum_{j \in J} a_j = \sum_{k \in K} a_k = \frac{1}{3} \sum_{l=1}^{n} a_l.$$

For example, for input $(1, 2, 3, 4, 4, 5, 8)$ the answer is yes, because there is the partition $(1, 8), (4, 5), (2, 3, 4)$. On the other hand, for input $(2, 2, 3, 5)$ the answer is no. Devise and analyze a dynamic programming algorithm for 3-PARTITION that runs in time polynomial in n and in $\sum_i a_i$.

6. [**KT-Chapter 6**] 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 quantity 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 out 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: it costs K for every order placed.
   - 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.