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.

   Solution: Build a table $T[]$ where $T[i]$ stores the minimize the sum of placement and access cost assuming we have servers $S_i, \ldots, S_n$ and we place a copy at S_i. So, $T[n]$ is c_n. Now to compute $T[i]$, we need to place a copy at S_i. Suppose the optimal solution for the problem considered by $T[i]$ places the next copy at $T[k]$ $(k > i)$ – then we need to pay for the access cost of $(k - (i+1)) + \ldots + 1 = \frac{(k-i)(k-i-1)}{2}$. Therefore,

$$T[i] = c_i + \min_{k=i+1,\ldots,n} \left(\frac{(k-i)(k-i-1)}{2} + T[k] \right).$$

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.

   Solution: Have a table $T[]$, where $T[i]$ tells you where the part of the string $s[i...n]$ can be reconstituted (so, the table entry is true or false). Clearly,

$$T[i] = \vee_{j=i,\ldots,n}(\text{dict}(s[i...j]) \vee T[j+1]).$$

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.

   Solution: Note that there are only 8 ways in which you can tile a particular column – call these ways $W_1, \ldots, W_8$ (3 ways in which you can put two pebbles, 4 for 1 pebble, and 1 for 0 pebble). Call two arrangements W_i, W_j compatible if placing pebbles like W_i and W_j in two adjacent columns (with W_i being on the left) does not violate any rules. Now have a table $T[i, c]$ which tells you the optimal placement for columns c till n provided the configuration in the column c is W_i. Now,

$$T[i, c] = \text{value}(W_i) + \max_j T[j, c + 1],$$

   where the maximum is taken over those configurations W_j which are compatible with W_i.

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.

   Solution: Suppose $w_1, \ldots, w_n$ are arranged in lexicographic ordering. Build a table $T[]$, where $T[i, j]$ gives the cost of the optimal tree for the words $w_i, \ldots, w_j$. If $i = j$, then $T[i, i] = f_i$. For $T[i, j]$, consider the optimal tree. If the root is w_r, then we have $w_i, \ldots, w_{r-1}$ in the left sub-tree and $w_{r+1}, \ldots, w_j$ in the right subtree. Further while computing the cost of the overall tree for $T[i, j]$ we need to account for the fact that the depth of the nodes (other than root node) increases by 1. So,

$$T[i, j] = (f_i + \cdots + f_j) + \max_{r=i,\ldots,j} (T[i, r - 1] + T[r + 1, j]).$$

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 $\sum_i a_i$.

Solution: Build a table $T[i, s_1, s_2]$, which stores a boolean value – this value is true if it is possible to partition $a_i, \ldots, a_n$ into 3 parts such that the first part adds up to s_1 and the second part adds up to s_2. Now, you can easily check the following recurrence (write the base cases yourself):

$$T[i, s_1, s_2] = \text{OR}(T[i+1, s_1, s_2], T[i+1, s_1 - a_i, s_2], T[i+1, s_1, s_2 - a_i]).$$

The three options correspond to the three options for 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.

   Solution: Build a table $T[i, s]$ which tells the optimal solution for month i till n given that you have s trucks at the beginning of month i (before you place any order for this month). Observe that $s \leq S$, and so, this table has nS entries. There are two cases – (i) $s \geq d_i$, and (ii) $s < d_i$. In the first case, we need to place an order (of at least $d_i - s$ trucks), and the remaining number of trucks can be at most S. Therefore, if s' denotes the number of trucks remaining after servicing the order for month i, then if we pay $s'C$ storage cost. Therefore,

$$T[i, s] = K + \max_{s'=0}^{S} s' \cdot C + T[i+1, s'].$$

   If $s > d_i$, we need not place an order in this month (why?), and so,

$$T[i, s] = (s - d_i) \cdot C + T[i+1, s - d_i].$$