# Tutorial 4 *By Vanshaj Sharma*

1.

**3-PARTITION.** Given a set of $n$ numbers $A = \{a_1, a_2, \cdots a_n\}$, does there exists a partition of in to three disjoint sets $A_1, A_2, A_3$ such that

$$\sum_{a_i \in A_1} a_i = \sum_{a_j \in A_2} a_j = \sum_{a_k \in A_3} a_k$$

**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):

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

The three options correspond to the three options for $a_i$.

2.

Dictionary You are given a string of n characters s, 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 1 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.** Let $T[i]$ be 1 if string $s[i \ldots n]$ can be reconstituted as a sequence of valid words else 0. Then,

$$T[i] = \max_{j=i\ldots n}(dict(i \ldots j) = 1 \wedge T[j+1] = 1)$$

The size of the DP table is $1 \times (n+1)$. Filling each entry takes $O(n)$ int the worst case. Therefore time complexity is $O(n^2)$.

3. Suppose we want to replicate a file over a collection of n servers, labeled S1, S2, ..., Sn.

To place a copy of the file at server Si results in a placement cost of ci , for an integer ci > 0.

Now, if a user requests the file from server Si , and no copy of the file is present at Si , then the servers Si+1, Si+2, Si+3, . . . are searched in order until a copy of the file is finally found, say at server Sj , where j > i.

This results in an access cost of j − i. (Note that the lower-indexed servers Si−1, Si−2, ... are not consulted in this search.) The access cost is 0 if Si holds a copy of the file.

We will require that a copy of the file be placed at server Sn, so that all such searches will terminate, at the latest, at Sn.

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 Si with i = 1, 2, ..., n − 1, of whether to place a copy of the file at Si or not. (Recall that a copy is always placed at Sn.)

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.

**Subproblems.** $T(i)$ = The minimum cost solution assuming we are placing a copy at $S_i$ and we only have the servers $S_i, S_{i+1}, \cdots S_n$, for all $i = 1, 2, \cdots n$.

**Recurrence.**

$$T((n) = c_n)$$

$$T(i) = c_i + \min_{i+1 \le k \le n} \left( \frac{(k-i-1)(k-i)}{2} + T(k) \right), \forall i < n$$

Why is the recurrence above correct ? Suppose you have placed a copy at $S_i$ (that is required by the definition of $T(i)$) - you pay $c_i$. Now, imagine that I tell you that the next copy is placed at server $S_k, i+1 \le k \le n$. Then what is the cost of this configuration ? Well, you need to pay access cost for serving $i+1, i+2, \cdots k-1$ with the copy at $S_k$ which is

$$(k-i-1) + (k-i-2) + \cdots + 1$$

We do not need to pay for access at $i$ since we have already placed a copy at $S_i$. So the other cost we pay is given by the optimal solution to the subproblem assuming $S_k$ has a copy and only servers from $k$ to $n$. The only catch is - we do not know the correct index $k$. So we try all and take the one which gives the minimum total cost.