COL 351

TUTORIAL SHEET 7

1. Suppose you own two stores, A and B. On each day you can be either at A or B. If you are currently at store A (or B) then moving to store B the next day (or A) will cost C amount of money. For each day i, $i = 1, \ldots, n$, we are also given the profits $P^A(i)$ and $P^B(i)$ that you will make if you are store A or B on day i respectively. Give a schedule which tells where you should be on each day so that the overall money earned (profit minus the cost of moving between the stores) is maximized.

   Solution: We make two tables T^A and T^B. The table $T^A[i]$ denotes the maximum profit from days $i, \ldots, n$ provided we start from store A on day i. Define $T^B[i]$ similarly. Clearly, the desired output is $\max(T^A[1], T^B[1])$. Base case is also easy $T^A[n] = P^A(n), T^B[n] = P^B(n)$. Now check that the following recurrence holds:

   $$T^A[i] = \max\left(T^A[i+1] + P^A(i), T^B[i+1] + P^A(i) - C\right).$$

   A similar recurrence can be computed for $T^B[i]$. Now these table entries can be computed from $i = n$ to 1.

2. Given a tree T where vertices have weights, an independent set is a subset of vertices such that there is no edge joining any two vertices in this set. Give an efficient algorithm to find an independent set of maximum total weight.

   Solution: We create two tables $T_1[v]$ and $T_2[v]$, where v is a vertex. $T_1[v]$ denotes the maximum weighted independent set for the sub-tree rooted below v when v is not included in the independent set and $T_2[v]$ is the same quantity when v is included in the independent set. As base cases, when v is a leaf, $T_1[v] = 0, T_2[v] = w_v$. Now suppose v is not a leaf node and has children $u_1, \ldots, u_k$. Then it is easy to see that

   $$T_2[v] = w_v + \sum_{i=1}^{k} T_1[u_i],$$

   and

   $$T_1[v] = \sum_{i=1}^{k} \max(T_1[u_i], T_2[u_i]).$$

   In which order should we compute these table entries ? Postorder traversal.

3. Given a tree $T = (V, E)$, where each vertex $v \in V$ has a weight w_v. Give a polynomial time algorithm to find the smallest weight subset of vertices whose removal results in a tree with exactly K leaves.

   Solution: Build a table $A[v, k]$ which gives the smallest weight subset of vertices which need to be removed from the subtree rooted below v such that it has k leaves.

Note that if the subtree below v, denoted by $T(v)$, has less than k leaves, then this entry is undefined. Leaf nodes form the base case – do it yourself. Now consider a node v and suppose it has children $w_1, \ldots, w_j$. Now, we need to figure out how many leaves we want in each of the subtrees $T(w_i)$. So for this, we run another dynamic program. Build a table $B[i, k']$ which tells us the smallest weight subset of vertices we need to remove from $T(w_1), \ldots, T(w_i)$ such that they have k' leaves (in total). So, $B[1, k']$ is same as $A[w_1, k']$. Now observe that

$$B[i, k'] = \min_{k''=0}^{k'} (B[i-1, k''] + A[w_i, k' - k'']).$$

Finally, $A[v, k] = B[j, k]$. Thus, we can fill in the table A using post-order traversal.

4. You are given N boxes, where box i has height h_i, width w_i and length l_i. Give an algorithm for finding a stacking of a subset of boxes of maximum total height : box i can be stacked on top of box j if $w_i < w_j$ and $l_i < l_j$.

   Solution: Arrange the boxes in decreasing order of w_i values. Let $T[i]$ denote the maximum height that can be attached from the first i boxes provided box i is on top. Clearly $T[1] = h_1$. Now, consider $i > 1$. Consider the optimal solution achieving $T[i]$ and let j be the box below i. Then note that all the boxes in the range $j+1, ..., i-1$ cannot appear below j. Therefore,

   $$T[i] = h_i + \max_{j \leq i : w_j < w_i, l_j < l_i} T[j].$$

   This gives an $O(n^2)$ time algorithm.

5. A *bitonic* sequence of numbers $x_1, x_2, x_3 \ldots x_k$ is such that there exists an i, $1 \leq i \leq k$ such that $x_1, x_2 \ldots x_i$ is an increasing sequence and $x_i, x_{i+1} \ldots x_n$ is a decreasing sequence. It is possible that one of the sequences is empty, i.e., strictly increasing (decreasing) sequences are also considered bitonic. For example 3, 6, 7 , 5, 1 is a bitonic sequence where 7 is the discriminating number.
   Given a sequence of n numbers, design an efficient algorithm to find the longest *bitonic subsequence*. In 2, 4, 3 , 1, -10, 20, 8, the reader can verify that 2,3,20, 8 is such a sequence of length 4.

   Solution: We have two tables $T_1[i]$ and $T_2[i]$. $T_1[i]$ is the maximum length of a bitonic sequence length which ends at $A[i]$ and has the property that it is only an increasing sequence. $T_2[i]$ is the maximum length of a bitonic sequence which ends at $A[i]$ and has the property it contains both an increasing and decreasing sequence. Base case is easy. Now, $T_1[i]$ can be computed by considering all values $j < i$ such that $A[j] < A[i]$ and adding 1 to $T_1[j]$, i.e.,

   $$T_1[i] = \max_{j < i : A[j] < A[i]} T_1[j] + 1.$$

   For $T_2[i]$, we need to guess the previous term in this sequence. Hence,

   $$T_2[i] = \max_{j < i : A[j] > A[i]} \max(T_1[j], T_2[j]) + 1.$$

6. Given a convex n-gon (number of vertices is n), we want to triangulate it by adding diagonals. Recall that $n-3$ diagonals are required to triangulate. The *cost* of triangulation is the sum of the lengths of the diagonals added. For example, in a parallelogram, we will choose the shorter diagonal for minimizing cost. Design an efficient algorithm to find the minimum cost diagonalization of a given n-gon.

   Solution: This is like the chain matrix multiplication. Label the vertices $v_1, \ldots, v_n$ in cyclic order. Let $T[i, j]$ be the optimal triangulation for the polygon formed by $v_i, \ldots, v_j$. Now, we give recurrence for $T[i, j]$: consider the polygon P formed by $v_i, \ldots, v_j$. In a triangulation of P, let the edge (v_i, v_j) be part of triangle given by v_k. Then the remaining triangulation can be thought of as triangulation of $v_i, ..., v_k$ and $v_k, \ldots, v_j$. Thus,

$$T[i, j] = \max_{k:i<k<j} T[i, k] + T[k, j] + l(v_i, v_j, v_k),$$

   where $l(v_i, v_j, v_k)$ is the total length of all the edges in the triangle (v_i, v_j, v_k).