CSL356 Sept 22,23,29

TUTORIAL SHEET 7

1. [**KT-Chapter6**] Suppose you are given a directed graph $G = (V, E)$ with length l_e on edges (which could be negative), and a sink vertex t. Assume you are also given finite values $d(v)$ for all the vertices $v \in V$. Someone claims that for each node $v \in V$, the quantity $d(v)$ is the cost of the minimum-cost path from node v to t. (i) Give a linear time algorithm which verifies whether the claim is correct, (ii) Assuming that all the distances $d(v)$ are correct, and that all $d(v)$ values are finite, you now need to compute distances to a different sink vertex t'. Give an $O(m \log n)$ time algorithm for computing these distances $d'(v)$ for all the vertices $v \in V$.

   Solution: (i) First of all, we must have $d(v) \leq d(w) + l(v, w)$ for every edge (v, w) — indeed, this says that one way of going from v to t is first go to w and then go to t. Assume this condition holds for every edge e. The first observation is that $d(v)$ is at most the length of shortest path from v to t. We can show this as follows: consider the shortest path P from v to t and then add up the above inequality for all edges in this path.

   Now, consider the edges which lie on a shortest path from any of the vertices to t. On such an edge e, if the values $d()$ are indeed correct, then we must have $d(v) = d(w) + l(v, w)$)(why ?). So, we consider all edges for which equality holds – such edges must form a connected graph. Now show that if P is a path in this connected graph from v to t, then $d(v)$ is equal to the length of this path (again, by adding up the equations for every edge). And so, from the previous paragraph, it follows that $d(v)$ is equal to the length of the shortest path from v to t.

   (ii) We would like to run Dijkstra because Dijkstra takes $O(m \log n)$ time. But, we need all edge lengths to be non-negative. For this, we define a new length of edge $e = (v, w)$ as $l'_e = l_e + d(w) - d(v)$. As noted above, $l'_e \geq 0$. Also, argue that for any vertex v, a shortest path with respect to l_e is also a shortest path with respect to l'_e and vice versa.

2. [**Dasgupta, Papadimitriou, Vazirani -Chapter6**]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,

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

3. **[Dasgupta, Papadimitriou, Vazirani -Chapter6]** 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):

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

The three options correspond to the three options for a_i.

4. 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.