COL 702

Homework II

Due on Sept. 26, 2021

Justify your answers with proper reasonings/proofs.

1. You are given a directed graph G and two vertices s and t in G. Each edge of G is associated with a cost $c(e)$ that may be negative; however there is no negative cycle in G. Give an efficient algorithm to compute the total number of shortest s-t paths in G. Note that you are not supposed to output the set of such paths, but just a count of how many different shortest paths are there from s to t.

   Solution We first compute the shortest path distances from s to every vertex v using Bellman Ford algorithm – let $D[v]$ denote this quanity. Let E' be the set of directed edges $e = (x, y)$ such that $D[y] = D[x] + c_{(x,y)}$.

   We first claim that if P is a shortest path from s to t, then all edges in P belong to E'. This is easy to see since for every edge (x, y) on this path, the portion of P from s to x has length $D[x]$, and similarly for y. So the length of the edge e is $D[y] - D[x]$.

   Conversely, we argue that any path from s to a vertex v using edges in E' only must be a shortest path. Indeed, if such a path is $v_0 = s, v_1, \ldots, v_k = v$, then each edge (v_i, v_{i+1}) on this path has length $D[v_{i+1}] - D[v_i]$. Summing over all edges, we see that the length of this path is $D[v] - S[s] = D[v]$. Thus, for each vertex v, we have to find the number of paths from s to v in the graph $G' = (V, E')$.

   We do this as follows: let $N_k[v]$ denote the number of paths from s to v in G' using exactly k edges. Now observe that $N_0[v]$ is 1 if $v = s$, 0 otherwise. Now, for any vertex v, a path from s to v must have the second last vertex from $\Gamma^-(v)$, where $\Gamma^-(v)$ denotes the set of in-neightbours of v. Therefore,

   $$N_k[v] = \sum_{u \in \Gamma^-(v)} N_{k-1}(u).$$

   We stop when $k = n$, and output $N_n[v]$ for every vertex v. Let us now see the overall time taken by this procedure. Finding $D[v]$ for every vertex takes $O(nm)$ time, and so this is also the time to construct G'. After this time taken to construct $N_k[v]$ for all vertices (from $N_{k-1}[u]$) is $\sum_v degree(v) \leq O(m)$. Therefore, the overall time is $O(mn)$.

2. You are given a directed graph G where all edge lengths are positive except for one edge. Given a source vertex s, give $O(m \log n)$ time algorithm for finding a shortest path from a vertex s to a vertex t. Now assume there are a constant number of edges in G which have negative weights (rest have positive weights). Give an $O(m \log n)$ time algorithm to find a shortest path from s to t.

   Solution Let R denote the edges $e_1, \ldots, e_k$ with negative length, and let $e_i = (u_i, v_i)$. Let H be the graph obtained by removing the edges in R. Let $D[v]$ denote the shortest

path from s to v in H. Also, let $D_i[v]$ denote the shortest path from v_i to v in H. Computing these take $O(km \log n)$ time, which is $O(m \log n)$ since k is a constant.

Now suppose P is a shortest path from s to t in G and say it contains the edges $e_{i_1}, e_{i_2}, \ldots, e_{i_r}$ from E' in this order as go from s to t. Then it is easy to check that the length of the path P is equal to

$$D[u_{i_1}] + c_{e_{i_1}} + D_{i_1}[u_{i_2}] + l_{e_{i_2}} + D_{i_2}[u_{i_3}] + \cdots + D_{i_r}[t].$$

Thus, we if knew the edges $e_{i_1}, e_{i_2}, \ldots, e_{i_r}$, then we can find the length of P in $O(1)$ time. So, we just try all such choices of $e_{i_1}, \ldots, e_{i_r}$ – there are at most $2^k \cdot k!$ such choices (there are 2^k subsets of edges in E' and then we try all orderings of this subset), which is a constant. So, in $O(m \log n)$ time, we can find the length of the path P.

3. The second minimal spanning tree is one that is distinct from the minimal spanning tree (has to differ by at least one edge) and is an MST if the original tree is ignored (they may even have the same weight). Design an efficient algorithm to determine the second MST.

   Solution The main observation is that we can assume that the second MST will differ from the MST in only one edge.

   Indeed, suppose it differs in two edges: let T denote the MST and T' denote the second MST. Let e and f be edges T which are not there in T'. Now consider adding e to the T'. This will create a cycle C and this cycle must have an edge which e' which is at least as expensive as e and not in T (why?). Then we can remove e' and add e to T' to get a tree of the same or better cost.

   Now we do not know the identity of the edge in which the two trees differ. But we can guess the edge e' which is in $T' \setminus T$. When we add this to T, this will create a cycle. We must drop the edge (other than e') in this cycle which has the highest cost. Running time: $O(mn)$.

4. You are given two arrays A and B, each of length n, containing integers. Give an efficient algorithm to find a permutation σ of $\{1, 2, \ldots, n\}$ such that the following quantity is minimized:

$$\sum_{i=1}^{n} |A[i] - B[\sigma(i)]|.$$

   Solution We can assume A is sorted in decreasing order (otherwise we will just adjust the permutation σ accordingly). Now, we also arrange B in decreasing order – call this permutation σ. We now show that this algorithm is optimal.

   We first show that there is an optimal algorithm which pairs $A[1]$ with $B[\sigma(1)]$, and the rest will follow by induction (as done in class). So let us prove this claim. Consider an optimal solution, and suppose it pairs $(A[1], B[i])$, where $i \neq \sigma(1)$, and say $A[j]$ is paired with $B[\sigma(1)]$. Now consider

$$|A[1] - B[i]| + |A[j] - B[\sigma(1)]|.$$

There are 4 possibilities for the signs above. Check that in each of these cases, the above is at least

$$|A[1] - B[\sigma(1)]| + |A[i] - B[j]|.$$

This shows that we can only improve the solution by pairing $(A[1], B[\sigma(1)])$ and $(A[i], B[j])$.

5. You are given a positive integer N. You want to reach N by starting from 1, and performing one of the following two operations at each step: (i) increment the current number by 1, or (ii) double the current number. For example, if $N = 10$, you can start with 1, and then go in the sequence $1, 2, 4, 8, 9, 10$, or $1, 2, 4, 5, 10$. Give an efficient algorithm, which given the number N finds the minimum number of such operations to go from 1 to N.

   Solution We use a recursive algorithm. If N is even, we half it and recursively solve for $N/2$, and at the last step we use doubling. If N is odd, we first subtract 1, recursively solve the problem and then add 1 at the last step.

   To prove correctness we use induction on N. If N is odd, the last step must be increase by 1, and by induction, the algorithm is optimal for $N - 1$. So the greedy algorithm is optimal for N as well. Now suppose N is even and consider an optimal algorithm for N. First assume that the last step done by this algorithm is doubling $N/2$ to N. Since the greedy algorithm also has this step as the last step, and by induction hypothesis, the greedy algorithm takes optimal number of steps to reach $N/2$, we see that the greedy algorithm is optimal for N as well.

   So the more interesting case is when the optimal algorithm goes from $N - 1$ to N as the last step. Now, the optimal way of reaching N_1, by induction hypothesis, is given by the greedy algorithm. But the greedy algorithm for $N - 1$ has as its last two steps (note that $N - 1$ is odd), doubling of $(N - 2)/2$ followed by increment by 1. Thus, we see that the optimal algorithm for N has as its last three steps going from $(N - 2)/2$ to $N - 1$ and then incrementing twice. But we can replace these three steps by two steps: first double $(N - 2)/2$ and then increment once. This is a contradiction, and so this case cannot happen. This proves that the greedy algorithm is optimal.

6. You are given an array A of length N (numbered $A[1], \ldots, A[N]$). The entries in the array are integers (positive or negative). A subarray of the array A is of the form $(A[i], A[i + 1], \ldots, A[j])$ for some $i \leq j$. Two subarrays are said to be disjoint if they do not have any common array element. A sub-array is said to be positive if the total sum of the values in it is positive. Now, given the array A, we would like to find the minimum number of mutually disjoint subarrays which cover all the **positive** elements in it. For example, if the array is $3, -5, 7, -4, 1, -8, 3, -7, 5, -9, 5, -2, 4$, one solution is to have three sub-arrays: $[3, -5, 7, -4, 1], [3, -7, 5], [5, -2, 4]$. Consider the following greedy algorithm for solving this problem: let $A[i]$ be the first positive integer from left. Starting from $A[i]$ find the largest index j such that the sub-array $A[i], \ldots, A[j]$ is positive. Take this sub-array and then solve the remaining problem for $A[j+1], \ldots, A[n]$ using the same algorithm. Is this algorithm optimal ? Either prove or disprove it.

Solution This statement is false. Indeed consider the following input: 1, -L, L, -2, 1, -2, 1, -2, 1, ..., -2, 1, where L is a very large positive number. The greedy algorithm will first form the sub-array $[1, -L, L]$ and after which each 1 will be a new sub-array. So if the pattern $-2, 1$ is repeated N times, the greedy algorithm uses $N+1$ subarrays. But it is easy to check that the optimal solution needs to use only 2 sub-arrays: [1] and $[L, -2, 1, -2, 1, \ldots, 1]$