College of Computer & Information Science — CS5800
Northeastern University — Algorithms
Fall 2017 — 11 December 2017

Practice Problems

Please note that this set of problems should not be viewed as a close representative of the final. Please use this material as a supplement to the rest of your preparation for the final. Best wishes for all your finals!

Problem 1. Sorting a partially sorted array

We are given an array A with n distinct integers, which is partially sorted; each element in A is within distance k of its position in the sorted order (increasing order). Formally, the input array A satisfies the following condition: For $i \in [1, n]$, the index of the element $A[i]$ in *sorted* order is in the range $[\max\{1, i-k\}, \min\{n, i+k\}]$.

Suppose we know k. Design an $O(n \lg k)$ time algorithm to solve the problem. Briefly justify the worst-case running time of your algorithm.

Answer: There are several approaches that work. Here is one. In the first phase, sort the first group of k elements (positions 1 through k). In the second phase, sort the second group of k elements (positions $k+1$ through $2k$) and merge with the sorted first group. In the ith phase, sort the ith group of k elements (positions $k(i-1)+1$ through ki) and merge with the sorted $(i-1)$th group. The last group may have fewer than k elements.

Since there are at most $\lceil n/k \rceil$ groups, and each sort takes $\Theta(k \log k)$ time and each merge takes $\Theta(k)$ time, we obtain a running time of $\Theta((n/k)k \log k + (n/k)k) = \Theta(n \log k)$.

Problem 2. Longest common substring

A string $X[1 \ldots k]$ is said to be a *substring* of $Y[1 \ldots n]$ if there exists $1 \le i \le n-k$ such that $Z[j] = Y[i+j]$ for all $1 \le j \le k$. In other words X is a substring of Y if there exist (possibly empty) strings A and B such that Y is a concatenation of A, Z, and B in that order.

Given two strings X and Y, we say that Z is a common substring of X and Y if Z is a substring of X and Z is a substring of Y.

Design an efficient algorithm for determining the length of a longest common substring of given strings X and Y of length m and n, respectively. State the worst case running time of your algorithm.

Answer: For each $1 \le i \le m$ and $1 \le j \le n$, let $L[i, j]$ denote the length of the longest common substring of $X[i \ldots m]$ and $Y[j \ldots n]$ that starts at $X[i]$ and $Y[j]$.

For the base case, we define $L[m+1, n+1] = 0$. For $1 \le i \le m$ and $1 \le j \le n$, if $X[i] = Y[j]$, we have $L[i, j] = L[i+1, j+1] + 1$; otherwise $L[i, j] = 0$. Using two for-loops, i from m to 1 and j from n to 1, we compute $L[i, j]$ for all i, j.

We return the maximum, over all i and j, of $L[i, j]$ (again using two for-loops).

The running time is $\Theta(mn)$.

Problem 3. Scheduling activities in two lecture halls

We are given a set S of n activites that we would like to schedule among two lecture halls. For each activity, we have a start time s_i and a finish time f_i. Two activities that overlap in time may not be scheduled in the same lecture hall; they may, however, be scheduled separately in the two lecture halls.

Design a greedy algorithm to select a maximum-size set of activities from S that may be scheduled among the two lecture halls.

Answer: This can be solved by a simple extension of the greedy algorithm for activity scheduling.

Maintain a current set X of unprocessed activities and set Y of selected activities. Initially, X is S and Y is $\emptyset$. Repeatedly: (i) find an activity i in X with earliest finish time; (ii) if i can be scheduled in any of the two lecture halls, add i to Y and place it in the lecture hall with largest finish time of a scheduled activity; (iii) delete i from X.

Return Y.

Problem 4. Kruskal and arbitrary MST

Let G be an undirected connected graph with positive weights (not necessarily distinct) on edges. Let T be an arbitrary minimum spanning tree of G. Show that there exists a way to execute Kruskal's algorithm such that it returns T as a result.

Answer: While running Kruskal, whenever we need to break ties among edges with the same weight, give preference to edges from T. We can prove by induction on the edges in order of appearance in Kruskal that the forest being maintained is a subset of T. Initially, it is empty. For the induction step, if $e \in T$ is being considered when our current forest is F, then clearly it will be added since what we have presently is part of T and adding this edge cannot create a cycle. If $e \notin T$ is being considered, then we now prove it forms a cycle with F. By our choice of ordering, every edge in T not in F has weight greater than $w(e)$. If e does not form a cycle with F, then the cycle it forms with T includes an edge of weight greater than $w(e)$; replacing this edge with e leads to a tree lighter than T, a contradiction.

Problem 5. Crucial edges in flow network

We say that an edge e is crucial in a given flow network G if e belongs to some minimum capacity cut in G. Give a polynomial time algorithm to determine if a given edge e is crucial in a given flow network G.

Answer: If e is crucial in G, then decreasing its capacity by 1 will decrease the min-cut by 1, and hence the max-flow by 1. If e is not crucial in G, then decreasing its capacity by 1 will have no impact on the min-cut and max-flow since e is not in any of the min-cuts.

So an algorithm to check if e is crucial is to (i) first calculate max-flow in G, (ii) reduce capacity of e by 1 and recompute max-flow; (ii) return YES if max-flow value decreases, otherwise return NO.

Problem 6. Division of computational tasks

Your company has n different computational tasks $T_1, T_2, ..., T_n$ that it needs to execute. There are 2 computers c_1 and c_2 that can execute these computational tasks. Each of the tasks $T_1, ..., T_n$ can be executed by any of the 2 computers c_1 or c_2.

You know that the costs of the computations are given by (for $1 \leq i \leq n$):

- The cost of executing task T_i by c_1 is $a_i > 0$.
- The cost of executing task T_i by c_2 is $b_i > 0$.
- If T_i and T_j are executed by 2 different computers (clearly $i \neq j$), there is an additional cost of d_{ij}. (The additional cost is the same whether T_i is executed by c_1 and T_j by c_2 or T_i is executed by c_2 and T_j by c_1.)

Suggest an algorithm to find the best way to divide the tasks between the two computers.

Answer: See Solution in "December 8's Review".

Problem 7. k-Densest Subgraph Problem

In the k-Densest Subgraph Problem, you are given an unweighted undirected graph G and an integer p, and the goal is to determine whether there exists a subset S of k vertices in G such that the total number of edges in G with both end-points in S is at least p.

Show that the k-Densest Subgraph Problem is NP-complete. (*Hint:* Give a reduction from the Maximum Clique Problem.)

Answer: Membership of the problem in NP is straightforward: a certificate is simply a subset S with k vertices and at least p edges among vertices in S. We can check in linear time if the number of vertices and edges are indeed as desired.

The reduction from Maximum Clique proceeds as follows. For the Maximum Clique problem we are given a graph G and an integer ℓ and asked if there exists a clique of size at least ℓ. We construct instance of k-Densest Subgraph: the input graph is G, k equals ℓ and p equals $\ell(\ell-1)/2$, which is the number of edges in a clique of size ℓ.

If G has a clique of size ℓ, then the answer to the k-Densest Subgraph problem is YES. For the other direction, note that the only way the answer to the given instance of the k-Densest Subgraph problem is YES is if there is a set S with ℓ vertices and $\ell(\ell-1)/2$ edges; but this set has to form a clique of size ℓ. Clearly, the reduction is polynomial-time.