CS3230 Design and Analysis of Algorithms September 6, 2020

Problem Set 3

Lecturer: Divesh Aggarwal

Instruction

This tutorial contains two kinds of questions - questions marked with an asterisk (*) are part of the assignment, and you need to submit your solutions to these questions; solutions to unmarked questions need not be submitted, but you should still attempt them. Your assignment solution will be graded. Submit your assignment solution by uploading to the LumiNUS. Solutions typeset using LaTeXare encouraged but not necessary. You may also submit scanned handwritten solutions, but they must be clearly readable. Solutions that are not readable may receive only partial marks. Late submissions will receive no marks. This assignment solution will be due by 11:59 pm, September 19, 2020 (Saturday).

Problem 3-1 (Minimize the number of coins)

Assume you live in a country, which has k different currency coins having values $b_1 = 1 < b_2 < \ldots < b_k$ cents. You just bought an amazing algorithms textbook costing $T > 0$ cents. Assume that you have an unlimited number of coins of each of the k values, and you want to provide *exact* change equaling T cents using the minimal number of coins c. For example, if $k = 2, b_1 = 1, b_2 = 3$ and $T = 5$, the optimal solution is $3 + 1 + 1 = 5$, using only $c = 3$ coins, since the other solution $1 + 1 + 1 + 1 + 1$ uses 5 coins.

Consider the following greedy algorithm for solving this problem: "Find the largest valued coin whose value v is less than or equal to T. Add this coin to your solution and recurse on $(T - v)$."

(a) Give an example with $k = 3$ (i.e., values of T, b_1, b_2, b_3) for which the above mentioned algorithm does not give the optimal solution.

(b) Now assume that $b_i = 3^{i-1}$ for $1 \le i \le k$. Use Local Swap to argue that in this case, the greedy algorithm given solve the correct answer.

(**Hint**: First show that $2 \cdot (b_1 + \ldots + b_{i-1}) < b_i$. How can you use this inequality?)

Problem 3-2 (*Fill in the Matrix) 20 Points

Assume that you are given as input two arrays $R[1 \ldots n]$ and $C[1 \ldots n]$ containing integers from $\{0, 1, \ldots, n\}$. You need to construct an $n \times n$ matrix A that contains entries 0 or 1 such that for all i, $R[i]$ is the number of 1s in i-th row of A, and $C[i]$ is the number of 1s in i-th column of A. If no such construction of A is possible, then your algorithm should output "Failure". For example, if $R[i] = C[i] = 1$ for all i, then one valid matrix A would the the identity matrix (but any so called "permutation matrix" will also work). On the other hand, if $\sum_i R[i] \neq \sum_j C[j]$, then no such matrix A can exist (as we must have $\sum_i R[i] = \sum_{i,j} A[i,j] = \sum_j C[j]$).

Consider the following greedy algorithm that does the following. It first checks whether $\sum_i R[i] = \sum_j C[j]$, and if not, then it outputs "Failure". Otherwise, it constructs A one row at a time. Assume inductively that the first $i-1$ rows of A have been constructed. Let $\mathsf{Curr}[j]$ be the number of 1s in the j-th column in the first $i-1$ rows of A. Now, sort the values $B[j] = C[j] - \mathsf{Curr}[j]$, and consider $R[i]$ columns j with $R[i]$ largest values $B[j]$. If $B[j] = 0$ for any of these $R[i]$ columns, the algorithm outputs "Failure". Otherwise, these $R[i]$ "largest columns" are assigned 1s in row i of A, and the rest of the columns are assigned 0s. That is, the columns that still needs the most 1s are given 1s.

For example, if $R[i] = C[i] = i$ for all i, then the greedy algorithm will output the unique matrix A where $A[i,j] = 1$ iff $i + j \geq n + 1$ (when, as you can check, is the only feasible solution).

Formally prove that this algorithm is correct using the Local Swap argument.

Problem 3-3 (Partially-Sorted)

Given an integer k, We say that an array $A[1..n]$ is *partially-sorted* if it can be divided into n/k blocks, each of size k (you may assume that n is always multiples of k), such that the elements in each block are larger than the elements in earlier blocks, and smaller than elements in later blocks. The elements within each block need not be sorted.
For example, the following array is partially sorted for $k = 3$:

1	4	2	7	6	8	10	11	9	15	13	16

(a) Prove that any comparison-based partially-sorting algorithm requires $\Omega(n \log(n/k))$ comparisons in the worst case by using the decision tree method.

(b) Give an algorithm that completely sorts an already *partially-sorted* array in $O(n \log k)$ time.

Problem 3-4 (*A long travel) 25 Points

You want to travel on a straight line from from city A to city B which is N miles away from A. For concreteness, imagine a line with A being at 0 and B being at N. Each day you can travel at most d miles (where $0 < d < N$), after which you need to stay at an expensive hotel. There are n such hotels between 0 and N, located at points $0 < a_1 < a_2 < \ldots < a_n = N$ (the last hotel is in B). Luckily, you know that $|a_{i+1} - a_i| \leq d$ for any i (with $a_0 = 0$), so that you can at least travel to the next hotel in one day. Your goal is to complete your travel in the smallest number of days (so that you do not pay a fortune for the hotels).

Consider the following greedy algorithm: "Each day, starting at the current hotel a_i, travel to the furthest hotel a_j s.t. $|a_j - a_i| \leq d$, until eventually $a_n = N$ is reached". I.e., if several hotels are within reach in one day from your current position, go to the one closest to your destination.

(a) (10 Points) Formally *prove* that this algorithm is correct if staying at any hotel costs the same amount of money c.

Suppose that you still want to travel from A to B, but now that a night stay at the i-th hotel has a costs c_i dollars (i.e. different hotels charge differently). Your goal is to minimize your total expenditure at the hotels.

(b) (5 Points) Give an example of values of $N, d, n, a_1, \ldots, a_n, c_1, \ldots, c_n$ for which the greedy algorithm from the previous problem does not work. Clearly state the answer given by the greedy algorithm and the correct answer for your algorithm.

(c) (5 Points) Let $C(k)$ be the minimum total expenditure at the hotels for traveling a distance of k miles from the starting point A. Give a recurrence relation that computes $C(k)$ in terms of the input parameters.

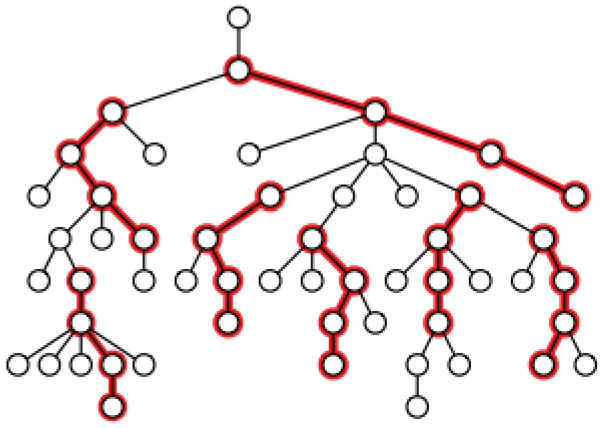
(d) (5 Points) Use the recurrence in part (c) to give a dynamic programming algorithm to compute $C(N)$. Analyze the running time of your algorithm.

Problem 3-5 (Paths on Tree)

Given a rooted tree T with n vertices and an integer $0 < k \leq n-1$. A *path* of length k is defined as follows: choose a starting vertex, travel towards the root (in other words, keep visiting the parent vertex) until $k+1$ vertices (including the starting one) have been visited, the $k+1$ visited vertices form a path. Two paths are disjoint if they don't share any vertex.

(a) Describe and analyse a greedy algorithm to compute the maximum number of disjoint paths in T, where each path has length k. Do **not** assume that T is a binary tree. For example, given the tree below as input and $k = 3$, your algorithm should output 8 as the solution.

Figure 1: Seven disjoint paths of length $k = 3$. This is **not** the largest set of disjoint paths.



(b) Now suppose each vertex in T has an associated reward, and your goal is to maximize the total reward of the vertices in your paths, instead of the total number of paths. Show that your greedy algorithm does **not** always return the optimal reward. [Hint: Give a counter example.]

(c) Describe an efficient algorithm to compute the maximum possible reward as described in part (b). Prove that your algorithm is correct.

Problem 3-6 (Don't put too much weight, please!)

Jack loves hiking and wants to always try different hiking trails. He has a map with n different landmarks labeled $1, \ldots, n$. Since there will be no water available during the paths between landmarks, the map also carries information about the amount of water $w[i, j]$ you need to carry to survive while taking the direct road from landmark i to j (without any intermediate landmarks in between), for $1 \leq i, j \leq n$.

On a given day, Jack may start at any landmark i and end at any landmark j for $1 \leq i, j \leq n$. He does not mind to visit some intermediate landmarks on the way, as long as the maximum weight that he needs to carry during the entire journey to be as little as possible. You may assume that there is plenty of water to fill from at each of the landmarks.

Assume you are given an edge-weighted graph $G = (V, E)$, where the set of vertices correspond to the n landmarks and the weight of the edge from i to j denotes the amount of water needed when taking the direct road from i to j.

(a) Assume that G is an *undirected tree*, i.e., the graph is an undirected connected graph with no cycles. Design an $O(n^2)$ algorithm to help Jack determine the smallest possible maximum weight $s[i, j]$ that he needs to carry when going from i to j, for all pairs of vertices i, j. i.e. your algorithm should correctly compute all values in the 2-dimensional array s.

(b) Now give an $O(n^2)$ algorithm to solve the problem of part (a) for any connected undirected graph G, and not necessarily a tree. Make sure you prove the correctness of your algorithm, and not just the run-time.