CS3230 Design and Analysis of Algorithms September 6, 2019

Solution *Sketches* to Problem Set 4

Name: Divesh Aggarwal

Problem 4-1 (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 $k = 1, b_1 = 1, b_2 = 3$ 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.

Solution: Let $T = 8$, and $b_1 = 1$, $b_2 = 4$, and $b_3 = 6$. Then the greedy algorithm above gives $c = 3$ coins ($8 = 6 + 1 + 1$), while the smallest solution is 2 coins ($8 = 4 + 4$). $\square$

(b) Give a dynamic programming algorithm for solving this algorithm. Prove the correctness of your algorithm and analyze the running time.

Solution: Iterate from $i = 0$ to T. Compute the smallest number of coins $c[i]$ for providing a change equaling i cents using the following recurrence:

$$c[i] = \begin{cases} 0, & \text{if } i = 0 \\ 1 + \min_{j : i - b[j] \geq 0} c[i - b[j]], & \text{if } i > 0 \;. \end{cases}$$

The running time is $O(kT)$. $\square$

(c) Now assume that $b_i = 3^{i-1}$ for $1 \leq i \leq k$. Use Local Swap to argue that in this case, the greedy algorithm given solve the correct answer. How does the running time of this algorithm compare with the running time of part (b).

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

Solution: Consider any optimal solution that differs from the solution given by the greedy algorithm, and let the coins chosen be $v_1 \geq v_2 \geq \cdots v_n$. Let the greedy algorithm choose coins $u_1 \geq u_2 \geq \cdots \geq u_m$. Note that each $u_i, v_i \in \{3^0, \ldots, 3^{k-1}\}$. Consider the smallest index i such that $v_i < u_i$. There exists such an index because of the assumption that the optimal solution differs from the greedy solution, which always chooses the largest possible coin at each step. Thus $v_i = 3^j$ and $u_i = 3^\ell$ for $j \leq \ell - 1$. This implies that $v_i + \cdots + v_n \geq 3^\ell$. We know that $v_i, \ldots, v_n \leq 3^j$. Thus, we claim that there are at least three coins in $\{v_i, \ldots, v_n\}$ of value 3^t for some $t \leq j$. This is because

$$2(3^0 + \ldots + 3^j) = 3^{j+1} - 1 < 3^\ell \ .$$

These three coins of value 3^t can be replaced by a single coin of value 3^{t+1} thus contradicting the optimality of the solution.

The running time of this algorithm is $O(T/3^{k-1} + k)$. $\square$

Problem 4-2 (Fill in the Matrix)

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.

Solution: If the greedy algorithm does not output "failure", then clearly the matrix computed is a feasible solution.

Now assume that there are feasible solutions, but the greedy algorithm outputs "Failure". Let i be the maximum taken over all feasible solutions such that the first $i-1$ rows of the solution are identical to that computed by the greedy algorithm. This implies that there exists a feasible solution with the first $i-1$ rows same as that computed by the greedy algorithm, but there is no solution that has the first i rows same as the greedy solution. Note that the greedy algorithm cannot fail when filling the i-th row if the first $i-1$ rows have been filled "correctly".

Let A' be the feasible solution with maximum number of entries in the ith row same as the greedy solution A and the first $i-1$ rows identical to A.

By the maximality of i, and since the number of 1s in the ith row is same in A and A', there exist j, k such that $A[i][j] = 1$, $A'[i][j] = 0$ and $A[i][k] = 0$, $A'[i][k] = 1$. Note that $A[i][j] = 1$ and $A[i][k] = 0$ implies $B[j] \geq B[k]$. Since $A'[i][j] = 0$, and $A'[i][k] = 1$, and A' is a feasible solution, this implies there are strictly more 1s in $A'[i+1, \ldots, n][j]$ than $A'[i+1, \ldots, n][j]$. This implies that there exists some $\ell \in \{i+1, \dots, n\}$ such that $A'[\ell][j] = 1$ and $A'[\ell][k] = 0$. Thus we can construct a matrix A'' as:

$$A''[a][b] = \begin{cases} 0, & \text{if } (a,b) = (i,k) \text{ or } (\ell, j) \\ 1, & \text{if } (a,b) = (i,j) \text{ or } (\ell, k) \\ A'[a][b], & \text{otherwise .} \end{cases}$$

It is easy to verify that A'' is a feasible solution which is identical to A' in the first $i-1$ rows, and has 2 more entries than A' that are identical to A in the ith row, thus contradicting the optimality of A'. $\square$

Problem 4-3 (Uninterrupted Health Insurance)

You live in a country with very high medical costs. You also anticipate undergoing a super-expensive medical treatment for the time period $[S, F)$. Luckily, as long as you have at least one job, you will have a cheap health insurance which will fully cover your expensive treatment. You are also very smart, so you get n job offers, where the i-th job is for the period $[s_i, f_i)$, so that the union of $[s_i, f_i)$ fully covers the desired interval $[S, F)$. For simplicity, assume the jobs are already sorted by their starting times, so that $s_1 \leq s_2 \leq \ldots \leq s_n$.

Of course, one possible solution is to take all the jobs (it's OK to have multiple jobs at the same time). However, being a naturally lazy person, you would like to take the smallest number of jobs which would cover the desired interval $[S, T)$.

(a) Design an efficient $O(n)$ greedy algorithm for this problem. For partial credit, give $O(n^2)$ algorithm. Do not forget to carefully argue the correctness of your algorithm.

Solution: Here is the insight which will give us our greedy algorithm. Let A be the set of all jobs that start before S and let $a \in A$ be the job in A that ends the latest. Then any solution to the Health Insurance problem must have at least one job in A. (Otherwise your not insured at the beginning of your treatment.) And if the solution has a job in A then we can always replace that job with a and we still have a valid solution which as good as the original. What's more we can completely remove any other jobs in A from the solution since they are completely covered by job a anyway. This operation can only improve our solution. So basically what i just argued is that given any optimal solution Z to the Health Insurance problem. We can always find an equivalent solution to Z that has only job a from the set A.

So that is going to be our greedy choice. In other words, on the highest level the algorithm does the following.

1. Find $[s_a, f_a) = a \in A$.
2. Add a to the solution, set $S = f_a$ and repeat until $S \geq F$.

To prove that this algorithm is correct we use induction on how many jobs are in the optimal solution. For the base case we argue that if the optimal solution has only one job then the algorithm works. Well, from the discussion above we know that one such optimal solution is the set $\{a\}$ which is exactly what the greedy algorithm outputs. So that case is done. Now we assume that the algorithm can find optimal solutions up to size $i-1$.

Now we consider problems who's optimal solution has i jobs. Let $a = [s_a, f_a)$ and let T be an optimal solution that contains a. (We know such a T exists by the above argument.) Then the crucial step is to argue that by removing a from T we get an optimal solution to the subproblem $[f_a, F)$. If we can show this we are done because by assumption our algorithm solves that subproblem optimally and also adds a to that solution so it produces something that is just as good as T for problem $[S, F)$.

Suppose for a moment that $T' = T\{a\}$ is not optimal for $[f_a, F)$ and let T'' be the a better (optimal) solution for that segment. Then we could solve the entire problem $[S, F)$ with $T'' \cup \{a\}$ which is strictly a better solution then T. That contradicts the optimality of T though so it must be that T' is also optimal for $[f_a, F)$. This concludes the proof that our greedy algorithm produces optimal solutions.

Here is the pseudocode for the greedy algorithm. It takes in parameters S, F and array J of length n. The entries of J are pairs of integers where $J[i].s$ is the start time for i-th job and $J[i].f$ is it's end time. Note that J is sorted by start times. We also assume that there is always at least one valid solution for the given inputs.

```
HealthInsurrance($S, F, J, n$)
    $i = 1$; $T = \emptyset$
    While $S < F$
        $max = i$
        While $(J[i].s \leq S) \wedge (i \leq n)$
            If $J[i].f > J[max].f$ Then
                $max = i$
            $i = i + 1$
        $T = T \cup i$
        $S = J[i].f_i$
    Return $T$
```

To compute the runtime of this algorithm it suffices to see that the if-statement is run at most once for every value of $i \in \{1 \dots n\}$ since i ever changed by being incremented by 1. So the runtime of the total algorithm is $O(n)$. □

(b) Assume you can hold at most two jobs at the same time (e.g., each job only has a day shift and a night shift, and you have to be at the job). Does your solution from part (a) still work in this case? If yes, prove it. Else, give a counter-example.

Solution: The solution from part (a) still works. This follows from the claim that no optimal solution to the problem in part (a) will ever have more then 2 jobs overlapping each other at a time.

We prove this claim by contradiction. Suppose we have an optimal solution T that contains three jobs x, y and z which over lap each other at some point such that x starts first, then y and finally z. Also let job $q \in \{x, y, z\}$ be the job that ends the latest. Then I claim that we can simply remove any the jobs in $\{y, z\} \setminus \{q\}$ from T since those jobs are completely covered by $\{x, q\}$. But that means we can remove at least one node from T and still get a valid solution. This contradicts the optimality of T so we can conclude that no triple of jobs in T overlap on a single point. $\square$