CSC373 Summer '22
Tutorial 3

June 2, 2022

Q1 Coin change

Consider the problem of making change with fewest possible coins.

Input: A positive integer A, and positive integer "denominations" $d[1] < d[2] < \ldots < d[m]$.

Output: A list of "coins" $c[1 \ldots n]$ is feasible when each $c[i]$ is in d, repeated coins are allowed (i.e. it is possible that $c[i] = c[j]$ with $i \neq j$), and $\sum_{i=1}^{n} c[i] = A$. Output the smallest n such that a feasible list of coins of size n exists. If no feasible solution exists, output $n = 0$.

Example: If we only have pennies, nickels, dimes and quarters (i.e. $d = [1, 5, 10, 25]$) to make change for 30¢(i.e. $A = 30$), then the output should be $n = 2$ (with, e.g., $c = [5, 25]$), and not $n = 3$ (with, e.g., $c = [10, 10, 10]$). If the input was $d = [5, 10, 25]$ and $A = 52$, then the output would be $n = 0$ as no solution exists.

We want to use dynamic programming to solve this problem.

(a) Define an array which stores the necessary information needed from various subproblems. Clearly identify what each entry of your array means, and how to compute the solution to the problem given your array entries.

(b) Write a Bellman equation describing the recurrence relation for the array identified above, and briefly justify its correctness. Pay close attention to the base cases.

(c) Write a bottom-up algorithm (pseudocode) to implement the Bellman equation identified above.

(d) Analyze the worst-case running time and space complexity of your algorithm. Does it run in polynomial time? Explain.

(e) Modify your Bellman equation and bottom-up implementation to compute a smallest feasible list of coins $c[1 \ldots n]$, in addition to the minimum number of coins n needed. When $n = 0$, you can return an empty list $c = []$. When necessary, define a second array to store partial information regarding the optimal solution. Analyze the worst-case running time and space complexity of your algorithm.

Q2 Longest Increasing Subsequence

Consider the following Longest Increasing Subsequence (LIS) problem:

Input: Array $A[1 \ldots n]$ of integers.

Output: Length of the longest subsequence S such that each element of S is strictly larger than all the elements before it.

Example: If $A = [4, 1, 7, 3, 10, 2, 5, 9]$, then the longest subsequence is $S = [1, 3, 5, 9]$ or $S = [1, 2, 5, 9]$, so the answer is 4. Note that $[6, 7, 8]$ and $[1, 2, 3]$ are not subsequences (they either include integers not in A or include integers out-of-order); $[4, 1, 7, 10]$ is not increasing; $[1, 3, 9]$ is not the longest possible.

Our goal is to design an $O(n^2)$ time DP for this problem. The bonus question asks you to reduce the running time to $O(n \log n)$.

(a) Define an array which stores the necessary information needed from various subproblems. Clearly identify what each entry of your array means, and how to compute the solution to the problem given your array entries.

(b) Write a Bellman equation describing the recurrence relation for the array identified above, and briefly justify its correctness. Pay close attention to the base cases.

(c) Write a bottom-up algorithm (pseudocode) to implement the Bellman equation identified above.

(d) Analyze the worst-case running time and space complexity of your algorithm.

(e) (Bonus Question, Warning: Difficult) Reduce the worst-case running time of your algorithm to $O(n \log n)$.