Math 182 Homework #4

Nathan Solomon

March 2, 2025

Problem 0.1.

Given a price p and a list of coin values $v = [v_1, v_2, \ldots, v_n]$, we want to design a function $min\_coins(p, v)$ which finds the minimum number of coins needed to sum up to exactly p. One efficient way to do this is to calculate $min\_coins(q, v)$ for $q = 0, q = 1, q = 2$, and so on, all the way to $q = p$, where $min\_coins(q, v) = \infty$ whenever there is no way to get the coins to add up to exactly q. The runtime of this algorithm is $O(nk)$, where $k = |v|$.

```
>>> from functools import lru_cache
>>> @lru_cache
... # v must be a tuple (not list), because the @lru_cache decorator only
... # works when parameters are all hashable types
... def min_coins(p, v):
...     if p == 0:
...         return 0
...     return 1 + min((min_coins(p - val, v) for val in v if p >= val))
...
>>> min_coins(31, (1, 2, 7, 20))
4
```

Homework 5

Math 182, Winter 2025

When designing an algorithm, you must include the following:

1. A short (1-2 sentence) sketch of the main idea of the algorithm.
2. The algorithm itself, written in pseudo-code.
3. The runtime of the algorithm.
4. A proof that the algorithm is correct.
5. A proof that the runtime is correct.

If you wish to sort a list, assume that you can do so in $O(n \log n)$ time; you do not need to write out the details of the sorting algorithm. If you wish to use an algorithm we have discussed in class (eg BFS/DFS, Interval Scheduling, Dijkstra's Algorithm, etc) you may do so without elaboration; if you need to modify the algorithm, you should provide sufficient details for the modification in your pseudocode.

Question 1:

Design an efficient algorithm to solve the problem from question 1 on homework 3. Recall that in that problem you are given a list of coin values $v_1, \ldots, v_n$ and a price p and you want to find the smallest number of coins whose values add up to p. Assume that you have an unlimited supply of each type of coin and that there is always some choice of coins whose values add up to p.

Example. Suppose the coins have values $1, 2, 7$, and 20. If the price is 31 then minimum number of coins required is 4—one coin of value 20, one coin of value 7 and two coins of value 2.

Question 2:

You operate a chain of restaurants along a highway. There is a rest stop located every mile along the highway and you can put a restaurant at any rest stop. However, some rest stops are more popular than others so restaurants placed at those rest stops will make more money. Also, to prevent your restaurants from competing with each other, you are not allowed to place any restaurants within k miles of each other (exactly k miles apart is fine though). Design an efficient algorithm to find the amount of money the restaurants will earn when they are placed in the best possible way.

Assume that you are given a length n array `A` containing the expected income from placing a restaurant at each mile along the highway and a positive integer k, indicating the closest that two restaurants can be to each other. You need to find the maximum possible value of `A[`i_1`] + A[`i_2`] + ... + A[`i_m`]` for all choices of indices $i_1 < i_2 < \ldots < i_m$, subject to the constraint that $i_i \leq i_2 - k$, $i_2 \leq i_3 - k$ and so on. For full credit, your algorithm should run in $O(nk)$ time. For partial credit, find any algorithm whose running time is a polynomial in n and k.

Example. Suppose `A = [2, 1, 5, 6, 5, 1, 1, 3]` and $k = 2$. Then the maximum amount of money possible is 15, which is given by choosing the restaurants at positions $1, 3, 5$, and 8 in the array.

Question 3:

Suppose you have a large rectangular piece of cloth of dimensions $n \times m$, where n and m are positive integers. Also, for each pair of positive integers, (w, l), there is a fixed price at which you can sell rectangular pieces of cloth with dimensions $w \times l$. You also have a machine that can cut any rectangular piece of cloth into two pieces along any horizontal or vertical line. You want to find the maximum amount of money you can get for your cloth if you cut it up in the optimal way. Design an efficient algorithm to solve this problem.

You should assume that you are given the integers n and m and an array `P` such that for each pair of positive integers w and l, `P[w, l]` is the price for which you can sell a rectangular piece of cloth of dimensions $w \times l$. Also assume that `P[w, l] = P[l, w]` for all w, l (i.e. you cannot get a higher price for a piece of cloth by rotating it 90°).

Question 4:

A *path* through an $n \times n$ grid is a sequence of squares in the grid such that each square is either horizontally or vertically adjacent to the previous square. A path is called *increasing* if each square is either to the right of or above the previous square in the path.

Suppose you are given an $n \times n$ grid where some squares have been marked as blocked. You want to know many increasing paths there are in the grid which start in the bottom left corner, end in the upper right corner and do not contain any blocked squares.

Example. Two grids are shown below, with blocked squares filled in with black. In the first grid, there are exactly 5 increasing paths from the bottom left corner to the upper right corner. In the second grid, there are none.

Assume that the input comes in the following format: `A` is an $n \times n$ array such that `A[i, j]` records whether the square in the i^{th} row and j^{th} column of the grid is blocked. In particular, assume `A[i, j]` is 0 if the square is blocked and 1 otherwise. Also assume that $(1, 1)$ corresponds to the square in the bottom left of the grid, so increasing i and j corresponds to going up and to the right in the grid, respectively.

Question 5:

Optional: Design an efficient algorithm to solve the following problem: given two sequences of integers, $a_1, \ldots, a_n$ and $b_1, \ldots, b_m$, find the length of the longest sequence which is a subsequence of both of them.

Example. If the sequences are $1, 2, 3, 4, 5, 6, 7$ and $5, 7, 2, 4, 8, 1, 6, 7, 7$ then the longest common subsequence is $2, 4, 6, 7$, which has length 4.