CS 124 Section 4 Solutions

Haneul Shin

February 2021

1. Given an array a with n integers, design a divide and conquer algorithm to find the sum of the maximum sum subarray. This is a subarray with the maximum possible sum, which may be empty. For instance, for a = [2, 3, -7, 1, 5, -3], the maximum subarray sum is 6.

   Solution. We can construct a divide and conquer algorithm that does the following:

   - Divide the array into two equally sized subarrays $\ell = a[0, n/2-1]$ and $r = a[n/2, n-1]$.
   - Recursively compute the maximum subarray sum of ℓ.
   - Recursively compute the maximum subarray sum of r.
   - Find the maximum subarray sum that crosses the divide between ℓ and r.
     - Iteratively find i that maximizes the subarray sum of $a[i, n/2-1]$.
     - Iterately find j that maximizes the subarray sum of $a[n/2, j]$.
     - Add the two sums.
   - Return the maximum of the above three sums.

   The time complexity of this solution is given by the recurrence $T(n) = 2T(n/2) + O(n)$, so $T(n) = O(n \log n)$.

2. Given an array a with n integers, find the length of a longest increasing subsequence of the array. (This is a maximum-length subsequence of the array such that each element is strictly larger than the previous element.)

   Solution. We use dynamic programming. Let length$[k]$ be the length of the longest increasing subsequence that ends at index k. Then, our DP relation is

   $$\text{length}[k] = \max\left(\max_{i \mid i<k, a[i]<a[k]} \text{length}[i] + 1, 1\right)$$

   since any increasing subsequence that ends at index k can be formed by appending $a[k]$ to the end of an increasing subsequence ending at some index $i < k$ such that $a[i] < a[k]$ (and if there is no such increasing subsequence, then the best we can do is the single element subarray $a[k]$). The pseudocode for this algorithm is shown below:

```
for k from 0 to n - 1:
    length[k] = 1
  for i from 0 to k - 1:
    if a[i] < a[k]:
      length[k]= max(length[k], length[i] + 1)
```

This solution takes $O(n^2)$ time.

3. Given a set of coin values $c = \{c_1, c_2, \ldots, c_k\}$ and a target sum of money n, determine the number of ways to produce the target sum where order matters. For instance, if $c = \{1, 2\}$ and $n = 3$, there are 3 distinct ways: (1+2, 2+1, 1+1+1).

   Solution. We use dynamic programming. Let count$[m]$ denote the number of ways to create the sum m with the coin values c. Then, our DP relation is

$$\text{count}[m] = \sum_{i | m \geq c_i} \text{count}[m - c_i]$$

   since for any coin such that $m \geq c_i$, the number of ways for c_i to be the last coin in an ordered set of coins that sum to m is count$[m - c_i]$. The pseudocode for this algorithm is shown below:


```
count[0] = 1
for m from 1 to n:
  for i from 1 to k:
    if m ≥ c[i]:
      count[m] += count[m - c[i]]
```


This solution takes $O(nk)$ time.