PSO #5 Solutions Sketch (Week 6)

Week of 2021-10-04

1 Warm-Up

1. Given n positive integers stored in array A and a value T, determine if there is a subset S whose elements sum to T.

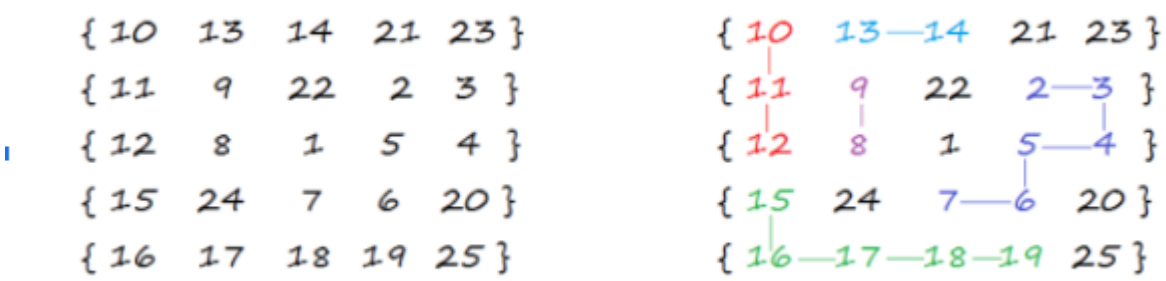
   Solution:
   Base Cases:
   $OPT(n, T) = 0$, if $T > 0$ and $n = 0$
   $OPT(n, T) = 1$, if $T = 0$
   Recurrence:
   $OPT(i, t) = \max(OPT(i-1, t), OPT(i-1, t - A[i]))$
   Time is $O(nT)$ and space is $O(nT)$ - pseudo-polynomial

2. Let L be an array of size n containing integers in arbitrary order. In the Longest Increasing Subsequence problem, the goal is to find the length of the longest subsequence in L such that all elements of the subsequence are sorted in increasing order. Once the longest length is known, generate the subsequence.

   Solution:
   Very similar to first warm-up. Simply consider at each step whether they should include this element in the LIS or not. The only base case is when you reach the last element in the array, which will always have a length 1 as its longest increasing subsequence length starting from that element.

2 Involved Problems

1. Given a n x n matrix A, find the length of the longest path of adjacent numbers through the matrix. The path can be formed of a sequence of numbers going up, down, left, or right. Adjacent numbers means each number in the sequence of numbers must be either 1 smaller or 1 greater than the two numbers on either side of it that are still part of the path (Harder Problem, give 'em a bit). Example:



Solution:
DP Runtime is $O(n^2)$ using the recurrence of taking the max when adding the optimal solutions plus 1 from each of the four directions. For each (i, j) pair, we compare the values at $A[i, j]$ with $A[i-1, j], A[i, j-1], A[i+1, j], A[i, j+1]$ to determine which ones are either one smaller or one greater than $A[i, j]$. Then, we take the max of the OPT expressions corresponding to the (i, j) pairs, where $OPT(i, j)$ is the length of the longest path of adjacent numbers starting at (i, j) and add 1 to it to signify that $OPT(i, j)$ is now included in whichever adjacent path is now formed.

2. Given a string S of length n, the longest palindrome is the longest subset of characters such that if one were to extract them from the string in order, then the resulting substring would form a palindrome. Design an algorithm to determine the length of the longest palindrome (Classic Problem, shouldn't be too hard).

   Solution:
   Base Cases:
   $OPT(i, i) = 1$
   $OPT(i, i+1) = 2, \text{ if } S[i] = S[i+1]$
   Recurrence:
   $OPT(i, j) = \max(OPT(i+1, j), OPT(i, j-1)), \text{ if } S[i] \neq S[j]$
   $OPT(i, j) = OPT(i+1, j-1) + 2, \text{ if } S[i] = S[j]$

3. Given a rod cut into n pieces, the cost of joining two pieces is twice the length of the left piece plus the length of the right piece. Only adjacent pieces may be joined. Design an algorithm to join the rod with minimum cost.

   Solution:
   $OPT(i, \ldots, j) = \min_{1 \leq m \leq j} (2 * \sum_{k=i}^{m} R[k] + \sum_{k=m}^{j} R[k] + OPT(i, \ldots, m) + OPT(m+1, \ldots, j))$,
   where R is array containing the lengths of each of the n pieces and $OPT(i, \ldots, j)$ is the optimal costs of merging the i^{th} up to the j^{th} pieces.

   Essentially, the overall idea is that we calculate the cost of merging at every possible point within the indices i to j, these points indicated by the number m, where we are treating all the rod pieces from indices i to m as already joined and the optimal costs provided by the $OPT(i, \ldots, m)$, and the same for the rod pieces from indices m to j. We multiply the sum of all the rod lengths on the left group by 2, as the cost of merging is twice the length of the

left side, and we have an expression for evaluating the cost of merging two pieces composed of all the pieces from i to j, and thus, all we need to do is take the minimum of all the ways to form such pieces.