6 Dynamic Programming Problems

1 - Stone Game

Recursive Definition

The recursive definition of this problem is to decide whether or not Alex would net the biggest piles of stones more than lee if they where to select optimally, would be: picking the biggest integer in the array until the array is empty.

The recursive solutions would be storing the initial values of the array into a matrix, and accumulate the selected value (The maximum).

Storing solutions:

The solutions we are storing are the accumulative value depended on the max of the previous stored values, which in the beginning case would be the initial values of the piles array. The solutions that are being stored will always be the maximum solutions of piles[i] –dp[I+1][i+d], and piles[I+d] – dp[i][i+d-1]

if we are assuming that Alex and lee always chose optimally. Understanding this also makes the problem unusual because if there are an even number of piles with an odd number of stones, then the solutions stored will always favor Alex.

However, if the length of the piles where odd then then it can change the solution, which dynamic programming will help.

2-Minimum Falling Path Sum

Recursive Definition

The recursive definition of this problem is finding the smallest summation on indices going down a grid. Starting from the top of a grid the path only allows you to go down left, down, or down right. Finding the correct path is very simple, you simply find the min of the previously stored solutions, which in the beginning case is going to be the first row of the matrix given. For instance, in the example the first row is [1,2,3] so the first row of my solutions matrix is [1,2,3]

Storing solutions

The solutions to the index that you are currently at will be the minimum value of the top left, top middle, and top right plus the value you are currently at. Example: [[1,2,3], [x1, x2, x3]]

x1 = min of (1,2) + x1, x2 = min of (1,2,3,) + x2, x3 = min of (2,3) + x3. This is how you can store solutions to the problem. You can also have a variable that keeps track of the smallest element as you are storing solutions to solve this problem in O(n) time

2-Minimum Falling Path Sum

Recursive Definition

The recursive definition of this problem is finding the smallest summation on indices going down a grid. Starting from the top of a grid the path only allows you to go down left, down, or down right. Finding the correct path is very simple, you simply find the min of the previously stored solutions, which in the beginning case is going to be the first row of the matrix given. For instance, in the example the first row is [1,2,3] so the first row of my solutions matrix is [1,2,3]

Storing solutions

The solutions to the index that you are currently at will be the minimum value of the top left, top middle, and top right plus the value you are currently at. Example: [[1,2,3], [x1, x2, x3]]

x1 = min of (1,2) + x1, x2 = min of (1,2,3,) + x2, etc. This is how you can store solutions to the problem. You can also have a variable that keeps track of the smallest element as you are storing solutions to solve this problem in O(n) time

3-Arithmetic Slices

Recursive Definition

The recursive definition of this problem is finding the amount substring that each element in those substrings differ consecutively from their adjacent elements. Its best to start from the smallest substring you can which must contain 3 elements so starting the solution at i=2 is the very logical.

Storing solutions

The way I’m storing solutions to this problem is counting the length of my array and incrementing based on how many consecutive differences I had found and by the size of the array.

4. Minimum ASCII Delete Sum for Two Strings

Recursive Definition

The recursive definition of this problem is very similar to the edit distance problem except when you are calculating the solution to your current index this is what you do: If the letters are the same you take from the upper left diagonal and add to the current ascii values letter. Otherwise you are taking the max from the top index and the left index. So the way you are breaking down problem is by is by letter comparison in a matrix, if you are deleting the letter, don’t calculate it, instead take the max from different indices. Otherwise if you are keeping the letter accumulate the ascii value

Storing Solutions

The solutions you are storing is by first calculating whether or not this letter needs to be deleted, just like the edit distance problem, if the letter needs to be deleted it is unnecessary to calculate its value, because the ideal solution is getting the total ascii value of the letters you didn’t delete and subtract it by the accumulative ascii values of the two strings. This will net you your answer. In a more specific answer to this is: The solutions you are storing are depended on an if else condition. If the letters are the same in the matrix the you accumulate the top left diagonal with your letters ascii value. Else: You take the max from the top index and the left index and store that solution. You do this because you want to calculate the total letters they have in common and subtract it by the two initial strings, the result is the minimum ascii delete

5. Integer Break

Recursive definition

The recursive definition for this problem is kind of like calculating fib, except when calculating the integer break you are constantly overwriting previous solutions with more maximized solutions and then proceeding the next index at the end of the loop. Just like fib you are solving this problem using DP with a single array.

Storing solutions

The way I solved this problem was with calculating the most maximum solution dependent on the max of the previous solution and the difference between the two indices, and store this solution if and only if the current index doesn’t have a larger value.

6. Palindromic Substrings

Recursive definition-

The recursive definition for this problem is very similar to multiple sub string problems. The first idea behind the DP definition is initializing all the diagonals to 1, this counts the substrings of just the letter itself. Now to count the continues substring you must check if the next character is equal to the index you’re currently at. dp[i][j] = 0 if dp[i+1] [j-1] = 0 else dp[i][ j] = 1 if s[i] == s[j] else 0. After you are done you can calculate the matrix and return the result.

Storing Solutions

For this problem I kept of getting confused when calculating my solutions especially when the substring was becoming very extended so my strategy to simplify the problem in my head was to simple just store solutions of the current problem, and then at the end calculate them with another pass, this extends the running time, but it helps me understand the problem in a clear way.