Design and Analysis of Algorithms

Homework # 7

Problem 1

You are going to Europe by road! Wow. And its one single road (wow!). You start on the road at distance d_1 =0. There are many stopovers on this roads and these are located at distances $d_1 < d_2 < \cdots < d_n$. You are driving the car yourself and will need to stop at some of these stopovers. In the end, you will definitely stop at the final stopover at distance d_n.

Your aim is to travel 500 kms every day (you make stopover at night), as this seems to be the ideal distance that balances the fatigue with the pleasure of travelling the most. However, it's not always possible to do so because the stopovers are not always 500 kms apart from each other. (If they were, you could travel 500 Kms and make a stop, and so on). Being of a quantitative bent of mind, you have formulated that if you travel α miles in a day, the amount of 'loss' (in terms of more fatigue and less joy) for that day is $(500 - \alpha)^2$.

You want to make stops such that this total loss is minimized — that is, the sum, over all travel days, of the daily losses is minimized. Give an efficient DP algorithm that determines the optimal sequence of stopovers at which to stop so as to minimize loss.

(a) Define a subproblem
(b) Write a recurrence
(c) Write a bottom up algorithm to compute the subproblem solutions
(d) Write a method to retrieve the optimal stopover sequence that produces the optimal (minimum) value of loss.

Solution

a. Sub-Problem: dp[i] is the minimum fatigue the person can have if he stops at the ith stop. b.

b. Recurrence: dp[i] = min(sq(500-(X[i]-X[j]) + dp[j]) where 0<=j<i

Problem 2

Now at this point you have become an algorithm expert and it's the right time to increase the difficulty level. You have given an array M = {1 ,2, 3, 8, 6, 7, 9, 10, 2, 1}. You have to calculate the longest increasing with decreasing subsequence in the array. Let's understand what a subsequence is through an example.

1, 3, 6, 10, 1 à Subsequence

1, 2, 3, 5 à Sequence

There is a subtle difference between subsequence and sequence. In Subsequence we can skip some elements but it should be done in increasing order, like 1 3 2 5 6 is not subsequence. Order of elements in parent array should remain same in subsequence. Like here 3 comes after 1 so there exist no subsequence in which 3 comes before 1.

In the problem you have to find such subsequence which has 2 parts, whose first part is **increasing** in order and second part is **decreasing** in order and it is the ***longest*** subsequence.

Here, (red is increasing subsequence and green is decreasing subsequence)

i) 1 2 3 8 9 10 2 1
ii) 1 2 3 6 7 9 10 2 1

are one of such subsequences but we can see that second subsequence is longest so we will choose this one

1) Define a subproblem for it.
2) Provide its recurrence.
3) Pseudo code for the algorithm devised.
4) Run time complexity

Solution

We will be having LIS and LDS , Longest increasing Sequence at *ending* index i and Longest decreasing Subsequence *Starting* at index i

= MAX(LIS[i] , DIS[i]) where i will range from 0 to size of array.

Problem 3

You are fascinated with stars and your friend challenges you with the algorithms question. In Problem you are given a m * m matrix containing only 1 and 0. Your challenge is to find biggest star in matrix in min amount of time and computations. For example in the following matrix the biggest start is highlighted, it is M[5][4] with length = 3. The star at M[5][4] entry with length 3 means all of the following entries have 1.

M[5][3] , M[5][2], M[5][1], //same row, previous columns

M[5][5], M[5][6], M[5][7], //same row, next columns

M[4][4], M[3][4], M[2][4], //same column, previous rows

M[6][4], M[7][4], M[8][4], //same column, next rows

M[4][3], M[3][2], M[2][1], //diagonal

M[6][5], M[7][6], M[8][7], // diagonal

M[6][3], M[7][2], M[8][1] // diagonal

M[4][5], M[3][6], M[2][7], // diagonal

1	1	0	1	1	0	1	1	0	1
1	1	1	0	1	1	1	0	1	0
0	1	0	1	1	1	0	0	0	1
0	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	0	0
1	0	1	0	1	1	1	0	0	1
0	1	0	0	1	0	1	1	1	1
0	1	1	1	1	0	0	1	0	1

A) DP Recurrence

B) Time Complexity for DP Solution

C) Time Complexity for Naive Solution

Solution

Given a 2D matrix **M** of binary digits, we have to find the size of the largest Star of 1s in it. Its DP solution is quite simple
As we simply populate 8 , 2D arrays up [][] down[][] right [][] left [][] and 4 diagonals in a single pass.

Recurrence: Up[i][j] = Up[i-1][j] + 1 if Matrix[i][j] == 1 , same for other matrices
Result = MAX(MIN(up[i][j] , down[i][j] ,right[i][j] ,left[i][j], Dig1[i][j], Dig2[i][j], Dig3[i][j], Dig4[i][j]))

Problem 4

As a CS student, your foremost duty is to find prime numbers, but not just some normal prime numbers. You are given n (Positive integer) which represents the no of digits in the number. Out of all possible numbers consisting on n digits, you have to find the numbers that satisfy following three rules:

1) Sum of every **three** consecutive digits is a prime number

For example, 3 Consecutive, **283**002 , 2**830**02, 28**300**2, 283**002**

2) Sum of every **four** consecutive digits is a prime number

For example, 4 Consecutive : **2830**02 , 2**8300**2, 28**3002**

3) Sum of every **five** consecutive digits is a prime number

For example, 5 Consecutive : **28300**2 , 2**83002**

Given n the no of digits in the number, you have to find all those numbers which follow these 3 rules.

A) Provide DP recurrence

B) Provide Run Time complexity

C) Provide Pseudo code for its implementation with **EXPLANATION**

Solution

```
isValid(num, range, dictionary) {
array = convert_num_to_array(num) // O(n)
sum_array[size(array) + 1]
sum_array[1] = 0

for (i -> 2 to size(array)) {
sum_array[i] = array[i - 1] + sum_array[i - 1]
}

i = range;

for (j -> i + 1 to size(array)) {
diff = sum_array[j] - sum_array[j - i]
if (dictionary.contains(diff) == true) {
continue
}
else {
if (isPrime(diff) == true) { // O(sqrt(n))
continue
}
else {
return false
}
}
i++;
}
return true

printNums(digits) {
dictionary = {}
upper_limit = pow(10, digits)
range = 3
for (i -> 0 to upper_limit) {

if (i / 99999 > 0)
range = 5
else if (i / 9999 > 0)
range = 4

if (isValid(i, range, dictionary) == true) {
dictionary.add(i);
cout << i;
}
}
}
```

Problem 5

As a successful bus conductor you job is to determine the number of ways you can make a change for a given amount and available coins.

i.e Your amount is 3 and no of available coins are 1 2 3.

So you can pay change in following ways

{1, 1, 1, 1}

{1, 1, 2}

{2, 2}

{1, 3}

You just have to return the Count.

Note: *Number of coins that you can use are infinite, no limit on coins quantity as you can see in the example coin 1 is used 4 times to make a change of 4,Secondly {1, 1, 2} and {2, 1, 1} are the same change, they will be counted as a single possible way*

D) Provide DP recurrence

E) Provide Run Time complexity

F) Provide Pseudo code for its implementation with **EXPLANATION**

G) Solution

https://www.geeksforgeeks.org/coin-change-dp-7/