DP-4

Assignment Solutions

PW SKILLS

Q1. given two strings. Find the length of the longest common substring.

Input 1: s = "abncjdnc" , t = "vvnfjnvabn"
Output 1: 3

Input 2: s = "nshjnb" , t = "vmk"
Output 2: 0

Solution :
Code link : https://pastebin.com/8snLmfh8

Output :

```
abcde
bcdae
3
```

Approach :

- The idea is to find the longest common suffix for all pairs of prefixes of the strings using dynamic programming using the relation:
- LCSuffix[i][j] = | LCSuffix[i-1][j-1] + 1 (if X[i-1] = Y[j-1])
  | 0 (otherwise)

where,

- 0 <= i – 1 < m, where m is the length of string X
- 0 <= j – 1 < n, where n is the length of string Y
- For example, consider strings ABAB and BABA.
-

		A	B	A	B
	0	0	0	0	0
B	0	0	1	0	1
A	0	1	0	2	0
B	0	0	2	0	3
A	0	1	0	3	0

- Finally, the longest common substring length would be the maximal of these longest common suffixes of all possible prefixes.

Q2. There are n stairs, a person standing at the bottom wants to reach the top. The person can climb either 1,2,3...m stairs at a time where m is a user given integer. Count the number of ways the person can reach the top.

Input 1 : n = 5 , m = 3
Output 1: 7

Solution :
Code Link : https://pastebin.com/AGaMhF87

Output :

```
5 3
7
```

Approach :

- The following recurrence relation has been used :
  - ways(n, m) = ways(n-1, m) + ways(n-2, m) + ... ways(n-m, m)
- We create a table res[] in bottom up manner using the following relation:
- res[i] = res[i] + res[i-j] for every (i-j) >= 0
- Such that the ith index of the array will contain the number of ways required to reach the ith step considering all the possibilities of climbing (i.e. from 1 to i).

Q3. The Tribonacci sequence Tn is defined as follows:
T0 = 0, T1 = 1, T2 = 1, and Tn+3 = Tn + Tn+1 + Tn+2 for n >= 0.
Given n, return the value of nth tribonacci number.

Example 1:
Input: n = 4
Output: 4
Explanation:
T_3 = 0 + 1 + 1 = 2
T_4 = 1 + 1 + 2 = 4

Example 2:
Input: n = 25
Output: 1389537

Solution :
Code link : https://pastebin.com/maqdKJTZ

Output :

```
25
1389537
```

Approach :

- We use an array to store the calculated values so that repeating values can be fetched without spending all that computing time on calculating the same thing again.

Q4. Given a set of positive integers S, partition set S into two subsets, S1 and S2, such that the difference between the sum of elements in S1 and S2 is minimized. The solution should return the minimum absolute difference between the sum of elements of two partitions.

For example, consider S = {10, 20, 15, 5, 25}.

We can partition S into two partitions where the minimum absolute difference between the sum of elements is 5.

S1 = {10, 20, 5}
S2 = {15, 25}

Note that this solution is not unique. The following is another solution:

S1 = {10, 25}
S2 = {20, 15, 5}

Solution :
Code link : https://pastebin.com/UpvC9fr6

Output :

```
5
10 20 15 5 25
5
```

Approach :
The idea is to consider each item in the given set S one by one, and for each item, there are two possibilities:
1. Include the current item in subset S1 and recur for the remaining items.
2. Include the current item from the subset S2 and recur for the remaining items.

Finally, return the minimum difference we get by including the current item in S1 and S2. When there are no items left in the set, return the absolute difference between elements of S1 and S2.
Q5. Given a positive integer n, count all n–digit binary numbers without any consecutive 1's.

Input 1: n = 5,
Output 1: 13
Explanation : [00000, 00001, 00010, 00100, 00101, 01000, 01001, 01010, 10000, 10001, 10010, 10100, 10101].

Solution :
Code link : https://pastebin.com/YyD3Czxq

Output :

```
5
13
```

Approach :
- The idea is to use recursion. We append 0 and 1 to the partially formed number and recur with one less digit at each point in the recursion.
- Here, the trick is to append 1 and recur only if the last digit of the partially formed number is 0.
- That way, we will never have any consecutive 1's in the output string.

Q6. Given a rod of length n and a list of rod prices of length i, where 1 <= i <= n, find the optimal way to cut the rod into smaller rods to maximize profit.

Input:
length[ ] = [1, 2, 3, 4, 5, 6, 7, 8]
price[ ] = [1, 5, 8, 9, 10, 17, 17, 20]

Rod length: 4

Best: Cut the rod into two pieces of length 2 each to gain revenue of 5 + 5 = 10

Cut	Profit
4	9
1, 3	(1 + 8) = 9
2, 2	(5 + 5) = 10
3, 1	(8 + 1) = 9
1, 1, 2	(1 + 1 + 5) = 7
1, 2, 1	(1 + 5 + 1) = 7
2, 1, 1	(5 + 1 + 1) = 7
1, 1, 1, 1	(1 + 1 + 1 + 1) = 4

Solution :
Code link : https://pastebin.com/wt9L8Huy

Output :

```
Profit is 10
```

Approach :

- We are given an array price[], where the rod of length i has a value price[i-1]. The idea is simple – one by one, partition the given rod of length n into two parts: i and n-i.
- Recur for the rod of length n-i but don't divide the rod of length i any further.
- Finally, take the maximum of all values. This yields the following recursive relation:
- rodcut(n) = max { price[i – 1] + rodCut(n – i) } where 1 <= i <= n
- We will solve this problem in a bottom-up manner.
- In the bottom-up approach, we solve smaller subproblems first, then solve larger subproblems from them.
- The following bottom-up approach computes T[i], which stores maximum profit achieved from the rod of length i for each 1 <= i <= n. It uses the value of smaller values 'i' that are already computed.

Q7. Given two strings str1 and str2 and below operations that can be performed on str1. Find the minimum number of edits (operations) required to convert 'str1' and 'str2'.
Insert , Remove , Replace
All of the above operations are of equal cost.

Input 1: str1 = "geek", str2 = "gesek"
Output: 1
Explanation: We can convert str1 into str2 by inserting a 's'.

Input 1: str1 = "cat", str2 = "cut"
Output: 1
Explanation: We can convert str1 into str2 by replacing 'a' with 'u'.

Input 1: str1 = "sunday", str2 = "saturday"
Output: 3
Explanation: Last three and first characters are the same. We basically need to convert "un" to "atur". This can be done using below three operations. Replace 'n' with 'r', insert t, insert a.

Solution :
Code link : https://pastebin.com/J97YanyZ

Output :

```
9
```

Approach :

- The idea is to process all characters one by one starting from either the left or right sides of both strings.
- Let us traverse from the right corner, there are two possibilities for every pair of characters being traversed.
- m: Length of str1 (first string)
- n: Length of str2 (second string)
- If the last characters of two strings are the same, there isn't much to do. Ignore last characters and count for remaining strings. So we recur for lengths m-1 and n-1.
- Else (If the last characters are not the same), we consider all operations on 'str1', consider all three operations on the last character of the first string, recursively compute the minimum cost for all three operations and take a minimum of three values.
- Insert: Recur for m and n-1
- Remove: Recur for m-1 and n
- Replace: Recur for m-1 and n-1

Q8. Given a distance 'dist', count the total number of ways to cover the distance with 1, 2 and 3 steps.

Input: n = 3

Output: 4
Explanation:
Below are the four ways
1 step + 1 step + 1 step
1 step + 2 step
2 step + 1 step
3 step

Input: n = 4
Output: 7

Solution :
Code link : https://pastebin.com/5fQHsSBk

Output :

```
7
```

Approach :

- Create an array of size n + 1 and initialize the first 3 variables with 1, 1, 2. The base cases.
- Run a loop from 3 to n.
- For each index i, compute the value of ith position as dp[i] = dp[i-1] + dp[i-2] + dp[i-3].
- Print the value of dp[n], as the count of the number of ways to cover a distance.

Q9. Given a positive integer K, the task is to find the minimum number of operations of the following two types, required to change 0 to K:

- **Add one to the operand**
- **Multiply the operand by 2.**

Input: K = 1
Output: 1
Explanation:
Step 1: 0 + 1 = 1 = K

Input: K = 4
Output: 3
Explanation:
Step 1: 0 + 1 = 1,
Step 2: 1 * 2 = 2,
Step 3: 2 * 2 = 4 = K

Solution :
Code link : https://pastebin.com/ScSq407z

Output :

```
5
```

Approach :

- If K is an odd number, the last step must be adding 1 to it.
- If K is an even number, the last step is to multiply by 2 to minimize the number of steps.
- Create a dp[] table that stores in every dp[i], the minimum steps required to reach i.

$$dp[i] = \begin{cases} dp[i-1] + 1 & ; \text{if i is odd} \\ dp[\frac{i}{2}] + 1 & ; \text{if i is even} \end{cases}$$

- **Observation :**
  - Let us consider an even number X
  - Now we have two options:
  - i) X-1
  - ii)X/2
  - Now if we choose X-1:
  - -> X-1 is an odd number , since X is even
  - -> Hence we choose subtract 1 operation and we have X-2
  - ->Now we have X-2, which is also an even number , Now again we have two options either to subtract 1 or to divide by 2
  - ->Now let us choose divide by 2 operation , hence we have (X – 2)/2 => (X/2) -1
  - Now if we choose X/2:
  - -> We can do the subtract 1 operation to reach (X/2)-1
  - Now let us consider sequence of both the cases:
  - When we choose X-1 : X -> X-1 -> X-2 -> (X/2)-1 [totally three operations]
  - When we choose X/2 : X -> X/2 -> (X/2)-1 [totally two operations]
  - This diagram shows how choosing divide operation for even numbers leads to optimal solution recursively
  - Hence we can say that for a given even number , choosing the divide by 2 operation will always give us the minimum number of steps.