Lesson:

Dynamic Programming-2

PW SKILLS

Q1. Decode Ways

A message containing letters from A-Z can be encoded into numbers using the following mapping:
'A' -> "1"
'B' -> "2"
...
'Z' -> "26"

To decode an encoded message, all the digits must be grouped then mapped back into letters using the reverse of the mapping above (there may be multiple ways). For example, "11106" can be mapped into:
"AAJF" with the grouping (1 1 10 6)
"KJF" with the grouping (11 10 6)
Note that the grouping (1 11 06) is invalid because "06" cannot be mapped into 'F' since "6" is different from "06".
Given a string s containing only digits, return the number of ways to decode it.

Input1: s = "12"
Output1: 2
Explanation: "12" could be decoded as "AB" (1 2) or "L" (12).

Input2: s = "226"
Output2: 3
Explanation: "226" could be decoded as "BZ" (2 26), "VF" (22 6), or "BBF" (2 2 6).

EXPLANATION:

- **Intuition behind the recursive solution:** We take out 1 letter and 2 letters at a time from the string. We then check if the letters taken out are valid (can be decoded), then we recurse the function without those letter(s), otherwise assign a 0 to that component if its invalid.
- In the code below, our base case is defined as idx == s.length() which is the same as no characters left in the string anymore. We then check if we have a stored value of this substring and just return that. Otherwise, we first take out one letter, check the letter's validity and then call the function again without this letter that we took out. Once that step is completed, we then move on to doing the same thing by taking out 2 letters this time.
- Before working for 2 letters, we need to check if we have anymore letters remaining and also, if the first letter we took out was 1 or 2. This is because the numbers in our range are till 26 only.

CODE: https://pastebin.com/pyNF9d8c

OUTPUT:

```
Enter the string:
12
2

Process finished with exit code 0
```

Q2. Given an integer array of coins[ ] of size N representing different types of currency and an integer sum, The task is to find the number of ways to make a sum by using different combinations from coins[]. Assume that you have an infinite supply of each type of coin.

Input1: sum = 4, coins[] = {1,2,3},
Output1: 4
Explanation: there are four solutions: {1, 1, 1, 1}, {1, 1, 2}, {2, 2}, {1, 3}.

Input: sum = 10, coins[] = {2, 5, 3, 6}
Output: 5

Approach 1: https://pastebin.com/3vVTY7Q5

Now lets see better approach

Explanation:

- The Idea to Solve this Problem is by using the Bottom Up(Tabulation). By using the linear array for space optimization.
- Initialize with a linear array table with values equal to 0.
- With sum = 0, there is a way.
- Update the level wise number of ways of coin till the ith coin.
- Solve till j <= sum.

Code link: https://pastebin.com/2kCiVQDQ

Leetcode link : https://leetcode.com/problems/coin-change-ii/description/

Q3. There are N stones, numbered 1,2,…,N. The height of ith stone is hi.
There is a frog who is initially on Stone 1. He will repeat an action some number of times to reach Stone N. The action is that if the frog is currently on Stone i, it jumps to one of the following: Stone i+1,i+2,…,i+K. Here, a cost of |hi - hj| is incurred, where j is the stone to land on.
Find the minimum possible total cost incurred before the frog reaches Stone N.

Input1:
n = 5
k = 3
10 30 40 50 20
Output1:
30

Input2:
3 1
10 20 10
Output2:
20

Explanation:

- We create a dp array of the same length as the input array.
- Let's define dp[i] as the minimum cost to reach Stone i from Stone 1. Our goal is to find dp[N], which represents the minimum possible total cost incurred before the frog reaches Stone N.
- We know that the minimum cost to reach Stone 1 from Stone 1 is 0, so we set dp[1] = 0.
- To calculate dp[i], we need to consider all possible jumps the frog can make from the previous stones (i-1, i-2, ..., i-K). We choose the jump that incurs the minimum cost. Mathematically, we can express this as: dp[i] = min(dp[i-1] + |hi - hi-1|, dp[i-2] + |hi - hi-2|, ..., dp[i-K] + |hi - hi-K|)
- We run 2 nested loops, the outer loop runs from 1 to n and the second loop runs from 1 to k.
- We initialize a pointer minn for every iteration.
- We check if the outer index variable is greater than or equal to inner index variable and if so, we update minn as minimum of its current value and height difference of the two indices.

- Update dp of current index as minn.
- Return dp of n-1 in the end.

Code: https://pastebin.com/FZN7YYWh

Output:

```
Enter the number of days :
5
Enter the number of maximum jumps allowed at a time:
3
Enter the elements of the array:
10 30 40 50 20
30

Process finished with exit code 0
```

Upcoming lectures:

- Problems on Dynamic Programming