Week 9 Live coding Solution

Problem 1

There are N stones, numbered $0, 1, 2, \ldots, N-1$. For each $i (0 \leq i \leq N-1)$, the height of Stone i is h_i .

There is a frog who is initially on Stone 0. He will repeat the following action some number of times to reach last stone.

If the frog is currently on Stone i, can jump to Stone $i+1$ or Stone $i+2$. Here, a cost of $\mid h_i - h_j \mid$ is incurred, where j is the stone to land on.

Find the minimum possible total cost to reach at last stone.

Write a function **minCost(H)**, where H is a list of heights for N stones. The function returns the minimum possible total cost to reach at last stone.

Sample Input

```
[10 30 40 20]
```

Output

```
30
```

Explanation

If we follow the path $0 \rightarrow 1 \rightarrow 3$, the total cost incurred would be $\mid 10-30 \mid + \mid 30-20 \mid = 30 \mid 10-30 \mid + \mid 30-20 \mid = 30$.

Solution

Recursive

```python
def solver(n,height):
    if n == 0:
        ans = 0
    elif n == 1:
        ans = abs(height[1]-height[0])
    elif n > 1:
            ans = min(solver(n-1,height)+abs(height[n]-height[n-1]),solver(n-2,height)+abs(height[n]-height[n-2]))
    return ans
def minCost(H):
    return solver(len(H)-1,H)
```

DP Memoization (Top down approach)

```python
def solvem(n,height,memo):
    if memo[n]==-1:
        if n == 0:
            ans = 0
        elif n == 1:
            ans = abs(height[1]-height[0])
        elif n > 1:
            ans = min(solvem(n-1,height,memo)+abs(height[n]-height[n-1]),solvem(n-2,height,memo)+abs(height[n]-height[n-2]))
        memo[n]=ans
    return memo[n]

def minCost(H):
    memo={}
    for i in range(len(H)):
        memo[i]=-1
    return solvem(len(H)-1,H,memo)
```

DP Tabular (Bottom-up approach)

```
def solvet(N, h):
    dp = [0]*N
    dp[1] = abs(h[1] - h[0])

    for i in range(2, N):
        dp[i] = min(dp[i-1] + abs(h[i-1] - h[i]), dp[i-2] + abs(h[i-2] - h[i]))

    return dp[N-1]
def minCost(h):
    N = len(h)
    return solvet(N, h)
```

Prefix Code

```
H = eval(input())
print(minCost(H))
```

Public Test case

Input 1

```
[10, 30, 40, 20]
```

Output

```
30
```

Input 2

```
[10, 10]
```

Output

```
0
```

Input 3

```

[30, 10, 60, 10, 60, 50]
```

Output

```
40
```

Private Test case

Input 1

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Output

```
8
```

Input 2

```
[1,2,2,3,3,4,4,5,5]
```

Output

```
4
```

Input 3

```
[1,2,3,4,5,5,5,5,5]
```

Output

```
4
```

Problem 2

Count Subsequence

A **subsequence** is a sequence that can be derived from another sequence by deleting some elements without changing the order of the remaining elements.

Write a function **countSubseq(S)** that accepts a string **S** which contains only digit characters. The function returns the number of non-empty subsequences that can be obtained from S such that every digit in the subsequence is strictly greater than all previous digits(if exist).

Example:-

If `S = '7598'` then there are `8` subsequences which follow the above constraint. These are `'7','5','9','8','79','78','59','58'`. Notice that `'7598'` is not a valid required subsequence because `7 > 5` and `9 > 8`.

Input

```
7598
```

Output

```
8
```

Solution

Solution Code

```python
def countSubseq(S):
    L=[]
    n = len(S)
    for d in S:
        L.append(int(d))
    count = [0 for i in range(10)]
    for i in range(n):
        for j in range(L[i] - 1, -1, -1):
            count[L[i]] += count[j]
        count[L[i]] += 1
    result = 0
    for i in range(10):
        result += count[i]
    return result
```

Suffix code(Visible)

```python
#Suffix Code
S = input()
print(countSubseq(S))
```

Public Test case

Input 1

```
7598
```

Output

```
8
```

Input 2

```
111324355
```

Output

```
95
```

Input 3

```
1123
```

Output

```
11
```

Input 4

```
54321
```

Output

```
5
```

Private Test case

Input 1

```
543216
```

Output

```
11
```

Input 2

```
6458132
```

Output

```
14
```

Input 3

```
1653587269
```

Output

```
99
```

Input 4

```
546766112378
```

Output

```
103
```

Input 5

```
987654321
```

Output

```
9
```

Problem 3

You are given weights and values of `N` items, put these items in a knapsack of capacity `W` to get the maximum total value in the knapsack. Note that we have only one quantity of each item. In other words, given two integer list `value[0..N-1]` and `weight[0..N-1]` which represent values and weights associated with `N` items respectively. Also given an integer `W` which represents knapsack capacity, find out the maximum `value` subset of value such that sum of the weights of this subset is smaller than or equal to capacity `W`. You cannot break an item, either pick the complete item or don’t pick it (0-1 property).

Write the function `knapSack(W, weight, value, N)` that returns the maximum possible value you can get.

Sample Input

```
3 #N
4 #W
[4,5,1] #weight
[1,20,3] #value
```

Output:

```
3
```

Solution

Solution Code

```
def knapSack(W, weight, value, N):
    st = [[0 for i in range(W+1)]for j in range(N+1)]
    for i in range(1,N+1):
        for j in range(1,W+1):
            if (weight[i-1]<=j):
                st[i][j]=max(value[i-1]+st[i-1][j-weight[i-1]],st[i-1][j])
            else:
                st[i][j]=st[i-1][j]
    return st[N][W]
```

Suffix Code

```
N=int(input())
W=int(input())
weight=eval(input())
values=eval(input())
print(knapSack(W,weight,values,N))
```

Public Test case

Input 1

```
3
4
[4,5,1]
[1,20,3]
```

Output

```
3
```

Input 2

```
3
3
[4,5,6]
[1,2,3]
```

Output

```
0
```

Input 3

```
6
10
[4,4,5,6,7,2]
[50,40,60,6,91,2]
```

Output

```
110
```

Private Test case

Input 1

```
6
8
[4,4,5,6,7,2]
[60,40,60,90,108,30]
```

Output

```
120
```

Input 2

```
6
10
[1, 2, 3, 8, 7, 4]
[20, 5, 10, 40, 15, 25]
```

Output

```
60
```

Input 3

```
8
100
[25, 35, 30, 46, 12, 65, 19, 32]
[22, 34, 56, 77, 86, 12, 33, 60]
```

Output

```
235
```

Input 4

```
8
50
[20, 30, 22, 10, 33, 19, 20, 40]
[34, 56, 78, 23, 45, 70, 67, 45]
```

Output

```
160
```

Problem 4

Given a rod of length `n` inches and an list of prices `price` that contains prices of all pieces of size smaller or equal `n`. Determine the maximum value obtainable by cutting up the rod and selling the pieces.

Write a function `cutRod(n,price)` that return the he maximum value obtainable by cutting up the rod and selling the pieces.

Sample Input

```
8 #n
[1, 5, 8, 9, 10, 17, 17, 20] #price
```

Output

```
22 #maximum value
```

Explanation:

The maximum obtainable value is 22 by cutting in two pieces of lengths 2 and 6, i.e., 5+17=22.

Solution

Solution code

```
def cutRod(n,price):
    length=[]
    st=[[0 for i in range(n+1)] for j in range(n+1)]
    for i in range(1,n+1):
        length.append(i)
    for i in range(1,n+1):
        for j in range(1,n+1):
            if(length[i-1]<=j):
                st[i][j]=max(price[i-1]+st[i][j-length[i-1]],st[i-1][j])
            else:
                st[i][j]=st[i-1][j]
    return st[n][n]
```

Suffix code (visible)

```
N = int(input())
price= eval(input())
print(cutRod(N,price))
```

Public Test case

Input 1

```
8
[1, 5, 8, 9, 10, 17, 17, 20]
```

Output

```
22
```

Input 2

```
8
[10, 5, 8, 9, 10, 17, 17, 20]
```

Output

```
80
```

Input 3

```
8
[1, 5, 8, 9, 10, 17, 17, 25]
```

Output

```
25
```

Private Test case

Input 1

```
10
[1,2,3,4,5,6,7,8,9,10]
```

Output

```
10
```

Input 2

```
10
[2,1,10,20,5,16,7,8,9,10]
```

Output

```
44
```

Input 3

```
10
[2,1,10,2,50,16,7,8,9,10]
```

Output

```
100
```