Started on	Monday, 28 April 2025, 1:37 PM
State	Finished
Completed on	Wednesday, 30 April 2025, 2:37 PM
Time taken	2 days
Overdue	1 day 22 hours
Grade	**100.00** out of 100.00

Question **1**

Correct

Mark 20.00 out of 20.00

Given a 2D matrix **tsp[][]**, where each row has the array of distances from that indexed city to all the other cities and **-1** denotes that there doesn't exist a path between those two indexed cities. The task is to print minimum cost in TSP cycle.

tsp[][] = {{-1, 30, 25, 10},
{15, -1, 20, 40},
{10, 20, -1, 25},
{30, 10, 20, -1}};

Answer: (penalty regime: 0 %)

Reset answer

```
from typing import DefaultDict


INT_MAX = 2147483647


def findMinRoute(tsp):
    sum = 0
    counter = 0
    j = 0
    i = 0
    min = INT_MAX
    visitedRouteList = DefaultDict(int)


    visitedRouteList[0] = 1
    route = [0] * len(tsp)


    while i < len(tsp) and j < len(tsp[i]):
        #Start here
        if counter >= len(tsp[i]) - 1:
```

	Expected	Got	
✔	Minimum Cost is : 50	Minimum Cost is : 50	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question **2**

Correct

Mark 20.00 out of 20.00

LONGEST COMMON SUBSTRING PROBLEM

The longest common substring problem is the problem of finding the longest string (or strings) that is a substring (or are substrings) of two strings.

Answer: (penalty regime: 0 %)

```
def lcs(str1, str2):
    m, n = len(str1), len(str2)
    table = [[0] * (n+1) for _ in range(m+1)]
    for i in range(1, m+1):
        for j in range(1, n+1):
            if str1[i-1] == str2[j-1]:
                table[i][j] = 1 + table[i-1][j-1]
            else:
                table[i][j] = max(table[i-1][j], table[i][j-1])
    lcs = ""
    i, j = m, n
    while i > 0 and j > 0:
        if str1[i-1] == str2[j-1]:
            lcs = str1[i-1] + lcs
            i -= 1
            j -= 1
        elif table[i-1][j] > table[i][j-1]:
            i -= 1
        else:
            j -= 1
    return lcs
str1=input()
```

	Input	Expected	Got	
✔	ABC BABA	The longest common substring is AB	The longest common substring is AB	✔
✔	abcdxyz xyzabcd	The longest common substring is abcd	The longest common substring is abcd	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question **3**

Correct

Mark 20.00 out of 20.00

Create a python program to find the maximum value in linear search.

For example:

Test	Input	Result
find_maximum(test_scores)	10 88 93 75 100 80 67 71 92 90 83	Maximum value is 100

Answer: (penalty regime: 0 %)

Reset answer

```
def find_maximum(lst):
    ##############  Add your code here ###############
    #Start here
    max=None
    for i in lst:
        if max == None or i > max:
            max = i
    return max
    #End here
test_scores = []
n=int(input())
for i in range(n):
    test_scores.append(int(input()))
print("Maximum value is ",find_maximum(test_scores))
```

	Test	Input	Expected	Got	
✔	find_maximum(test_scores)	10 88 93 75 100 80 67 71 92 90 83	Maximum value is 100	Maximum value is 100	✔

	Test	Input	Expected	Got	
✔	find_maximum(test_scores)	5 45 86 95 76 28	Maximum value is 95	Maximum value is 95	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question **4**

Correct

Mark 20.00 out of 20.00

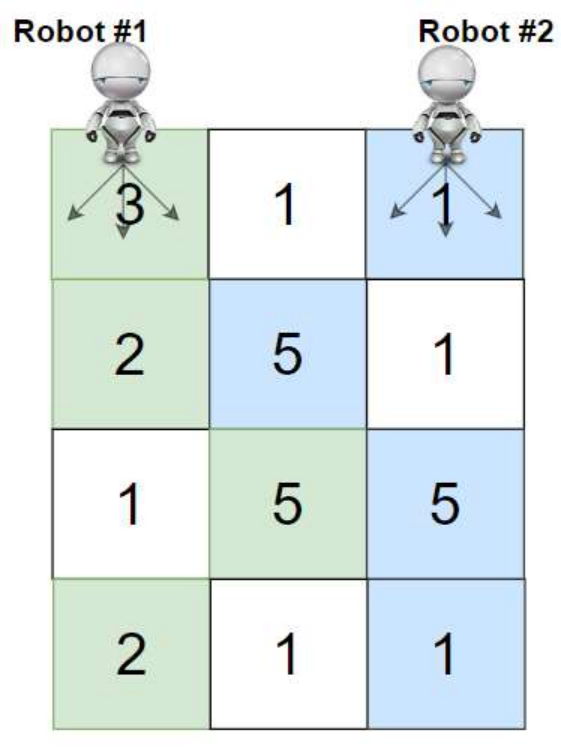
You are given a `rows x cols` matrix `grid` representing a field of cherries where `grid[i][j]` represents the number of cherries that you can collect from the `(i, j)` cell.

You have two robots that can collect cherries for you:

- **Robot #1** is located at the **top-left corner** `(0, 0)`, and
- **Robot #2** is located at the **top-right corner** `(0, cols - 1)`.

Return *the maximum number of cherries collection using both robots by following the rules below*:

- From a cell `(i, j)`, robots can move to cell `(i + 1, j - 1)`, `(i + 1, j)`, or `(i + 1, j + 1)`.
- When any robot passes through a cell, It picks up all cherries, and the cell becomes an empty cell.
- When both robots stay in the same cell, only one takes the cherries.
- Both robots cannot move outside of the grid at any moment.
- Both robots should reach the bottom row in `grid`.



For example:

Test	Result
ob.cherryPickup(grid)	24

Answer: (penalty regime: 0 %)

Reset answer

```
class Solution(object):
    def cherryPickup(self, grid):
        def dp(k):
            #########  Add your code here ##########
            #Start here
            if k == ROW_NUM - 1:
                return [[grid[-1][i] if i == j else grid[-1][i] + grid[-1][j] for j in range(COL_NUM)
                        for i in range(COL_NUM)]
            row = grid[k]
```

```
        ans = [[0] * COL_NUM for i in range(COL_NUM)]
        next_dp = dp(k + 1)
        for i in range(COL_NUM):
            for j in range(i, COL_NUM):
                for di in [-1, 0, 1]:
                    for dj in [-1, 0, 1]:
                        if 0 <= i + di < COL_NUM and 0 <= j + dj < COL_NUM:
                            if i == j:
                                ans[i][j] = max(ans[i][j], next_dp[i + di][j + dj] + row[i])
                            else:
                                ans[i][j] = max(ans[i][j], next_dp[i + di][j + dj] + row[i] + row
        return ans
```

	Test	Expected	Got	
✔	ob.cherryPickup(grid)	24	24	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question **5**

Correct

Mark 20.00 out of 20.00

Create a python program for 0/1 knapsack problem using naive recursion method

For example:

Test	Input	Result
knapSack(W, wt, val, n)	3 3 50 60 100 120 10 20 30	The maximum value that can be put in a knapsack of capacity W is: 220

Answer: (penalty regime: 0 %)

Reset answer

```
def knapSack(W, wt, val, n):
    ############### Add your code here ##############
    #Start here
    if n == 0 or W == 0 :
        return 0
    if (wt[n-1] > W):
        return knapSack(W, wt, val, n-1)
    else:
        return max(val[n-1] + knapSack(W-wt[n-1], wt, val, n-1), knapSack(W, wt, val, n-1))
    #End here
x=int(input())
y=int(input())
W=int(input())
val=[]
wt=[]
for i in range(x):
    val.append(int(input()))
for y in range(y):
    wt.append(int(input()))
n = len(val)
print('The maximum value that can be put in a knapsack of capacity W is: ',knapSack(W, wt, val, n))
```

	Test	Input	Expected	Got	
✔	knapSack(W, wt, val, n)	3 3 50 60 100 120 10 20 30	The maximum value that can be put in a knapsack of capacity W is: 220	The maximum value that can be put in a knapsack of capacity W is: 220	✔

	Test	Input	Expected	Got	
✔	knapSack(W, wt, val, n)	3 3 55 65 115 125 15 25 35	The maximum value that can be put in a knapsack of capacity W is: 190	The maximum value that can be put in a knapsack of capacity W is: 190	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.