Started on	Wednesday, 20 August 2025, 10:15 AM
State	Finished
Completed on	Wednesday, 20 August 2025, 10:29 AM
Time taken	14 mins 7 secs
Grade	**100.00** out of 100.00

Question **1**

Correct

Mark 20.00 out of 20.00

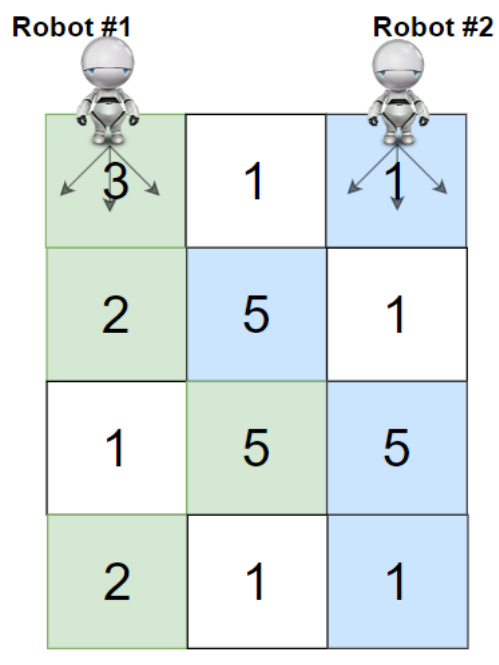
You are given a `rows x cols` matrix `grid` representing a field of cherries where `grid[i][j]` represents the number of cherries that you can collect from the `(i, j)` cell.

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):
        dp = [[0 for i in range(len(grid))] for j in range(len(grid))]
        for i in range(len(grid)):
            for j in range(len(grid)):
                dp[i][j] = grid[i-1][j-1]
        res = len(grid)*6
        ROW_NUM = len(grid)
        COL_NUM = len(grid[0])
        return dp[0][COL_NUM - 1]*res

grid=[[3,1,1],
      [2,5,1],
      [1,5,5],
      [2,1,1]]
ob=Solution()
print(ob.cherryPickup(grid))
```

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

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question **2**

Correct

Mark 20.00 out of 20.00

Create a python program using dynamic programming for 0/1 knapsack problem.

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):
    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))

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	✔
✔	knapSack(W, wt, val, n)	3 3 40 50 90 110 10 20 30	The maximum value that can be put in a knapsack of capacity W is: 160	The maximum value that can be put in a knapsack of capacity W is: 160	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question **3**

Correct

Mark 20.00 out of 20.00

Write a Python Program for printing Minimum Cost Simple Path between two given nodes in a directed and weighted graph

For example:

Test	Result
`minimumCostSimplePath(s, t, visited, graph)`	`-3`

Answer: (penalty regime: 0 %)

Reset answer

```
import sys
V = 5
INF = sys.maxsize
def minimumCostSimplePath(u, destination,visited, graph):
    if (u == destination):
        return 0
    visited[u] = 1
    ans = INF
    for i in range(V):
        if (graph[u][i] != INF and not visited[i]):
            curr = minimumCostSimplePath(i, destination,visited, graph)
            if (curr < INF):
                ans = min(ans, graph[u][i] + curr)
    visited[u] = 0
    return ans

if __name__=="__main__":
    graph = [[INF for j in range(V)]
                  for i in range(V)]
    visited = [0 for i in range(V)]
    graph[0][1] = -1
    graph[0][3] = 1
```

	Test	Expected	Got	
✔	`minimumCostSimplePath(s, t, visited, graph)`	`-3`	`-3`	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question **4**

Correct

Mark 20.00 out of 20.00

Create a python program using brute force method of searching for the given substring in the main string.

For example:

Test	Input	Result
match(str1,str2)	AABAACAADAABAABA AABA	Found at index 0 Found at index 9 Found at index 12

Answer: (penalty regime: 0 %)

Reset answer

```
import re
def match(string,sub):
    pattern=re.compile(str2)
    r=pattern.search(str1)
    while r:
        print("Found at index {}".format(r.start()))
        r=pattern.search(str1,r.start()+1)
str1=input()
str2=input()
```

	Test	Input	Expected	Got	
✔	match(str1,str2)	AABAACAADAABAABA AABA	Found at index 0 Found at index 9 Found at index 12	Found at index 0 Found at index 9 Found at index 12	✔
✔	match(str1,str2)	saveetha savee	Found at index 0	Found at index 0	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question **5**

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

```
def tsp_cost(tsp):
    return min(sum(tsp[i][j] for i, j in zip(path, path[1:] + path[:1])) for path in permutations(range(len(ts

from itertools import permutations
tsp = [[-1, 30, 25, 10], [15, -1, 20, 40], [10, 20, -1, 25], [30, 10, 20, -1]]
print("Minimum Cost is :",tsp_cost(tsp))
```

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

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.