Started on	Monday, 19 May 2025, 11:29 AM
State	Finished
Completed on	Monday, 19 May 2025, 11:59 AM
Time taken	29 mins 51 secs
Grade	**80.00** out of 100.00

Question **1**

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 **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

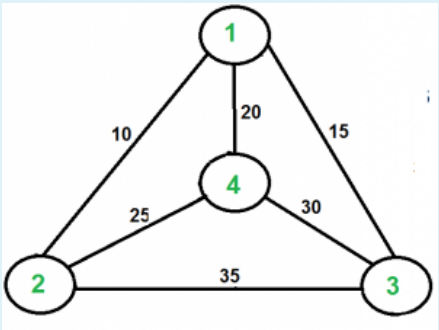
Marks for this submission: 20.00/20.00.

Question **3**

Not answered

Mark 0.00 out of 20.00

Solve Travelling Sales man Problem for the following graph



Answer: (penalty regime: 0 %)

Reset answer

```
from sys import maxsize
from itertools import permutations
V = 4


def travellingSalesmanProblem(graph, s):

    //Write your code




if __name__ == "__main__":

        graph = [[0, 10, 15, 20], [10, 0, 35, 25],
            [15, 35, 0, 30], [20, 25, 30, 0]]
    s = 0
    print(travellingSalesmanProblem(graph, s))
```

Question **4**

Correct

Mark 20.00 out of 20.00

Create a python program to compute the edit distance between two given strings using iterative method.

For example:

Input	Result
kitten sitting	3

Answer: (penalty regime: 0 %)

```
def mind(x,y):
    m=len(x)
    n=len(y)
    # dp=[[0]*(n+1) for_ in range(m+1)]
    dp = [[0] * (n + 1) for _ in range(m + 1)]
    # print(dp)
    for i in range(m+1):
        for j in range(n+1):
            if i==0:
                dp[i][j]=j
            elif j==0:
                dp[i][j]=i
            elif x[i-1]==y[j-1]:
                dp[i][j]=dp[i-1][j-1]
            else:
                dp[i][j]=min(dp[i-1][j-1],dp[i][j-1],dp[i-1][j])+1
    return dp[m][n]
x=input()
y=input()
print(mind(x,y))
```

	Input	Expected	Got	
✔	kitten sitting	3	3	✔
✔	medium median	2	2	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question **5**

Correct

Mark 20.00 out of 20.00

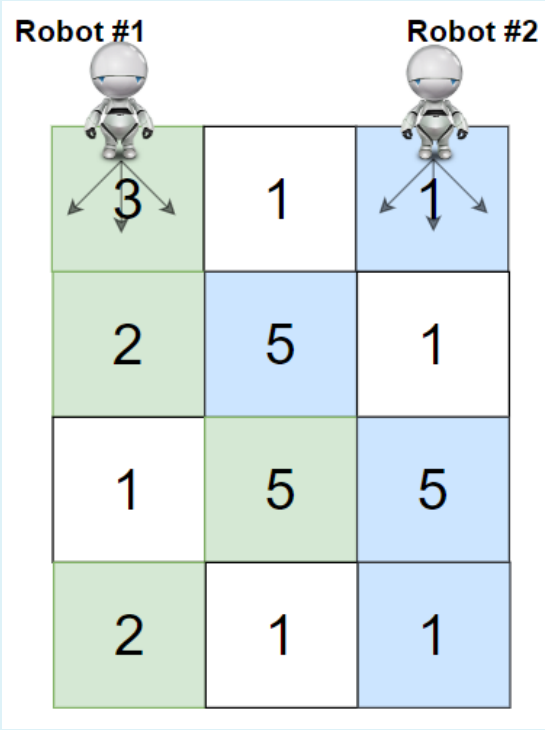
You are given a `rows x cols` matrix `grid` representing a field of cherries where `grid[i][j]` represents the number of cherries that you can collect from the `(i, j)` cell.

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.