Started on	Thursday, 29 May 2025, 2:20 PM
State	Finished
Completed on	Thursday, 29 May 2025, 2:31 PM
Time taken	10 mins 37 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

```
def match(string,sub):
    l = len(string)
    ls = len(sub)
    start = sub[0]
    for i in range(l-ls+1):
        if string[i:i+ls]==sub:
            print(f"Found at index {i}")

    ############ Add your code here #######

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**

Not answered

Mark 0.00 out of 20.00

Write a Program for Implementing merge sort on float values using python recursion.

For example:

Test	Input	Result
`merge_sort(inp_arr)`	5 3.2 1.6 9.5 4.3 4.55	Input Array: [3.2, 1.6, 9.5, 4.3, 4.55] Sorted Array: [1.6, 3.2, 4.3, 4.55, 9.5]
`merge_sort(inp_arr)`	6 3.2 1.2 5.3 9.6 8.5 7.4	Input Array: [3.2, 1.2, 5.3, 9.6, 8.5, 7.4] Sorted Array: [1.2, 3.2, 5.3, 7.4, 8.5, 9.6]

Answer: (penalty regime: 0 %)

```
1
```

Question **3**

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 ##############
    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 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

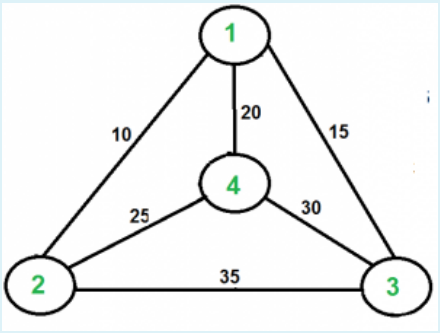
Marks for this submission: 20.00/20.00.

Question **4**

Correct

Mark 20.00 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
    vertex = []
    for i in range(V):
        if i != s:
            vertex.append(i)
    min_path = maxsize
    next_permutation=permutations(vertex)
    for i in next_permutation:

        current_pathweight = 0
        k = s
        for j in i:
            current_pathweight += graph[k][j]
            k = j
        current_pathweight += graph[k][s]
        min_path = min(min_path, current_pathweight)
```

	Expected	Got	
✔	80	80	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question **5**

Correct

Mark 20.00 out of 20.00

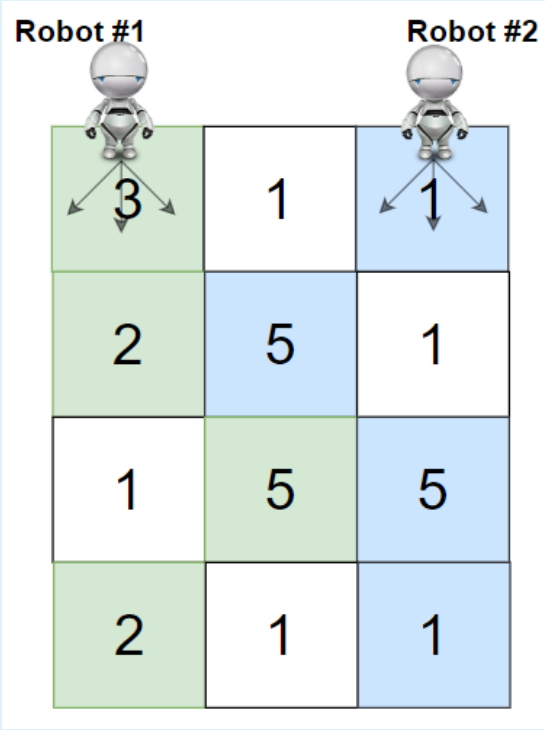
You are given a `rows x cols` matrix `grid` representing a field of cherries where `grid[i][j]` represents the number of cherries that you can collect from the `(i, j)` cell.

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):
        def dp(i, j, k):
            if (i, j, k) in memo:
                return memo[(i, j, k)]

            if i == ROW_NUM - 1:
                return grid[i][j] + (grid[i][k] if j != k else 0)

            cherries = grid[i][j] + (grid[i][k] if j != k else 0)

            max_cherries = 0
            for dj in [-1, 0, 1]:
                for dk in [-1, 0, 1]:
                    next_j, next_k = j + dj, k + dk
                    if 0 <= next_j < COL_NUM and 0 <= next_k < COL_NUM:
```

```
                    max_cherries = max(max_cherries, dp(i + 1, next_j, next_k))

                memo[(i, j, k)] = cherries + max_cherries
                return memo[(i, j, k)]

        ROW_NUM = len(grid)
```

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

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.