Lab 11

Exercise 0: You are given two integers. Compute their sum.

Input: The first line of the input is T, the number of test cases. Then there are T lines, each giving two integers N M.

Output: For each test case, print the sum of the two integers given.

Example:

Sample Input	Sample Output
10	4
5 -1	19
12 7	201
178 23	0
0 0	1
0 1	1
1 0	-1
-1 0	-1
0 -1	11722
9247 2475	3763
2439 1324	

Example solution:

```
def main():
    infile = open("in.txt", "r")
    numCases = int(infile.readline().strip())
    for case in xrange(numCases):
        s = infile.readline().strip().split(' ')
        x = int(s[0])
        y = int(s[1])
        print x + y

if __name__ == '__main__':
    main()
```

Exercise 1: You are in a 2D-maze. Find the shortest way to get from the start location to the end location. The starting location is specified by 'S', and the ending location by 'E'. A cell with '.' represents an empty square, and you can walk through it. A cell with '#' represents a wall, and you cannot walk through it. From any given cell, you can walk either up, down, left, or right,

provided that you don't walk outside the maze or into a wall. Find the shortest path from the starting location to the end location.

Example solution: Create a graph where each cell is a vertex, and two vertices have an edge between them if the corresponding cells are adjacent. Then, we do a BFS starting from the cell containing 'S' to find the distance to the cell containing 'E'.

```
from collections import deque

def visit(x, y, queue, distance, D):
    distance[x][y] = D
    queue += [[x,y]] # push x

def main():
    infile = open("in.txt", "r")
    numCases = int(infile.readline().strip())
    for case in xrange(numCases):
        maze = []
        s = infile.readline().strip().split(' ')
        n = int(s[0])
        m = int(s[1])
        for i in xrange(n):
            maze += [infile.readline().strip()]
        for i in xrange(n):
            for j in xrange(m):
                if maze[i][j] == 'S':
                    sx = i
                    sy = j
                elif maze[i][j] == 'E':
                    ex = i
                    ey = j
        # BFS
        distance = []
        for i in xrange(n):
            distance += [[-1]*m]
        queue = deque()
        visit(sx, sy, queue, distance, 0)
        dx = [1, -1, 0, 0]
        dy = [0, 0, 1, -1]
        while len(queue) > 0:
            b = queue.popleft()
            x = b[0]
            y = b[1]
            if x==ex and y==ey:
                print distance[x][y]
```

```
                break
            # there are at most 4 vertices adjacent to (x,y)
            for i in xrange(4):
                nx = x + dx[i]
                ny = y + dy[i]
                if nx>=0 and nx<n and ny>=0 and ny<m and maze[nx][ny]!='#':
                    if distance[nx][ny] == -1:
                        visit(nx, ny, queue, distance, distance[x][y] + 1)
        if distance[ex][ey] == -1:
            print -1

if __name__ == '__main__':
    main()
```

Input: The first line of the input is T, the number of test cases. Then, for each of the T test cases, the first line gives N M, the dimensions of the maze (N is the number of rows, and M is the number of columns). The next N lines are then each M characters long and give the layout of the maze.

Output: For each test case, print the length of the shortest path from the start to end location, followed by a newline. If there is no way to get from the start to the end, print -1.

Example:

Sample Input	**Sample Output**
3 2 4 S... ###E 4 4 .##. #S.. ###. E... 3 4 E#.. #S..	4 7 -1

Exercise 2: You are given a 2D-image drawn using 10 colors. Each color is represented by an integer from 0 to 9. The image is described by giving the color of each pixel in the image. Two pixels are adjacent if one is immediately to the left of the other or immediately above the other. The image consists of many objects, and two pixels are part of the same object if they are adjacent

and have the same color. The area of an object is the number of pixels it has. Find the area of the largest object in the image.

Input: The first line of the input is T, the number of test cases. Then, for each of the T test cases, the first line gives N M, the dimensions of the image (N is the number of rows, and M is the number of columns). The next N lines are then each M characters long and describe the colors in the image.

Output: For each test case, print the area of the largest object in the image, followed by a newline.

Example:

Sample Input	Sample Output
3 2 4 0011 0000 4 4 9876 9987 9998 9999 3 4 1212 2121 1212	6 10 1

Example solution: Create a graph where each pixel is a vertex, and two vertices have an edge between them if the corresponding pixels are adjacent. Then, we are looking for the size of the largest connected component. The `dfs` function below is recursive and returns the number of pixels in the connected component of (x, y) which have not already been visited in the variable `visited`. In the `main` function, we loop over all pixels (the two nested `for` loops), i.e. all vertices, and `dfs` on vertices which are in connected components that we haven't already visited. We return the maximum size over all such `dfs` calls.

Here is a recursive solution:

```
def dfs(x, y, visited, image):
    dx = [1, -1, 0, 0]
    dy = [0, 0, 1, -1]
    n = len(image)
    m = len(image[0])
```

```
    visited[x][y] = True
    ans = 1
    for i in xrange(4):
        nx = x + dx[i]
        ny = y + dy[i]
        if nx>=0 and nx<n and ny>=0 and ny<m and image[x][y]==image[nx][ny]:
            if not visited[nx][ny]:
                ans += dfs(nx, ny, visited, image)
    return ans

def main():
    infile = open("in.txt", "r")
    numCases = int(infile.readline().strip())
    for case in xrange(numCases):
        image = []
        s = infile.readline().strip().split(' ')
        n = int(s[0])
        m = int(s[1])
        for i in xrange(n):
            image += [infile.readline().strip()]
        visited = []
        for i in xrange(n):
            visited += [[False]*m]
        ans = 0
        for i in xrange(n):
            for j in xrange(m):
                if not visited[i][j]:
                    ans = max(ans, dfs(i, j, visited, image))
        print ans

if __name__ == '__main__':
    main()
```

Here is a non-recursive solution using a `stack` variable and `while` loop:

```
def visit(x, y, stack, visited):
    visited[x][y] = True
    stack += [[x,y]]

def main():
    infile = open("in.txt", "r")
    numCases = int(infile.readline().strip())
    for case in xrange(numCases):
        image = []
        s = infile.readline().strip().split(' ')
```

```
        n = int(s[0])
        m = int(s[1])
        for i in xrange(n):
            image += [infile.readline().strip()]
        # DFS
        visited = []
        for i in xrange(n):
            visited += [[False]*m]
        ans = 0
        dx = [1, -1, 0, 0]
        dy = [0, 0, -1, 1]
        for i in xrange(n):
            for j in xrange(m):
                if not visited[i][j]:
                    stack = []
                    componentSize = 0
                    visit(i, j, stack, visited)
                    while len(stack) > 0:
                        t = stack.pop()
                        x = t[0]
                        y = t[1]
                        componentSize += 1
                        for k in xrange(4):
                            nx = x + dx[k]
                            ny = y + dy[k]
                            if nx>=0 and nx<n and ny>=0 and ny<m:
                                if image[x][y]==image[nx][ny]:
                                    if not visited[nx][ny]:
                                        visit(nx, ny, stack, visited)
                    ans = max(ans, componentSize)
        print ans

if __name__ == '__main__':
    main()
```