11.

Ball Moving Out of Grid in N Steps

Aim

Find the number of ways a ball can move out of a grid in exactly **N steps**.

Algorithm

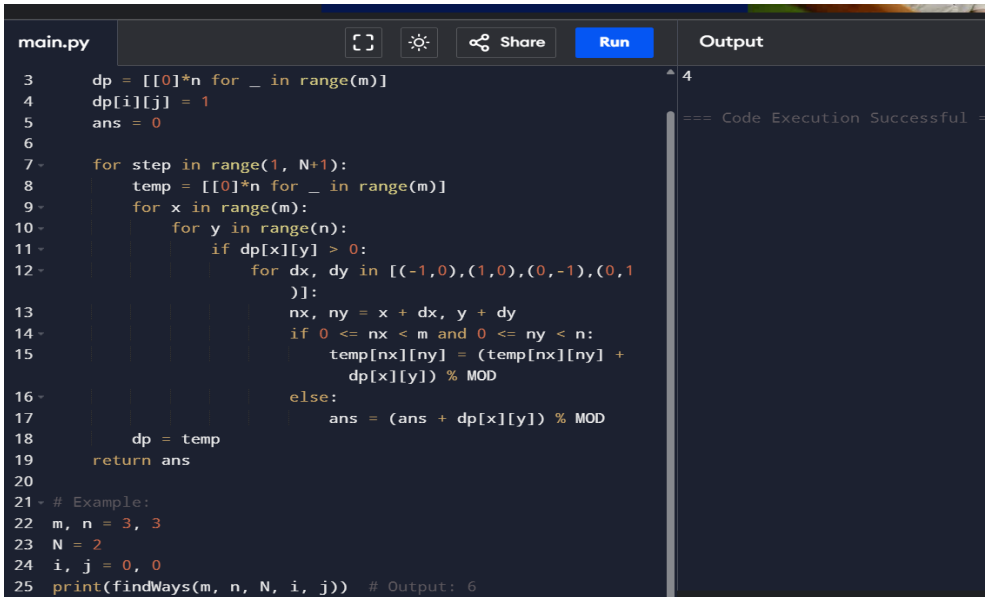
1. Use **DFS with memoization (Dynamic Programming)**.
2. At each step, move in four directions (up, down, left, right).
3. If the ball goes outside the grid, count it as 1.
4. If steps run out inside the grid, return 0.
5. Use memoization (i, j, steps) to avoid recomputation.

Input

m=2, n=2, N=2, i=0, j=0

Output

6



12.

House Robber II (Circular Street)

Aim

Find the maximum money that can be robbed without robbing two adjacent houses in a circle.

Algorithm

1. Normal House Robber (linear) uses DP.
2. For circular houses → two cases:
    - Exclude first house and rob from 1 to n-1.
    - Exclude last house and rob from 0 to n-2.
3. Take the maximum of both cases.

Input

nums = [2, 3, 2]

Output

3

13.

Climbing Stairs

Aim

Find distinct ways to climb n steps, moving either **1** or **2** steps at a time.

Algorithm

1. This is **Fibonacci sequence**:
   ways(n) = ways(n-1) + ways(n-2)
2. Use DP to avoid recomputation.

Input

n = 4

Output

5

14.

Robot Unique Paths

Aim

Find the number of unique paths from **top-left** to **bottom-right** moving only **down** or **right**.

Algorithm

1. Use **DP table**.

2. Each cell = paths from top + paths from left.

Input

m=3, n=2

Output

3

15.

Large Groups in String

Aim

Find intervals of large groups (≥3 same consecutive characters).

Algorithm

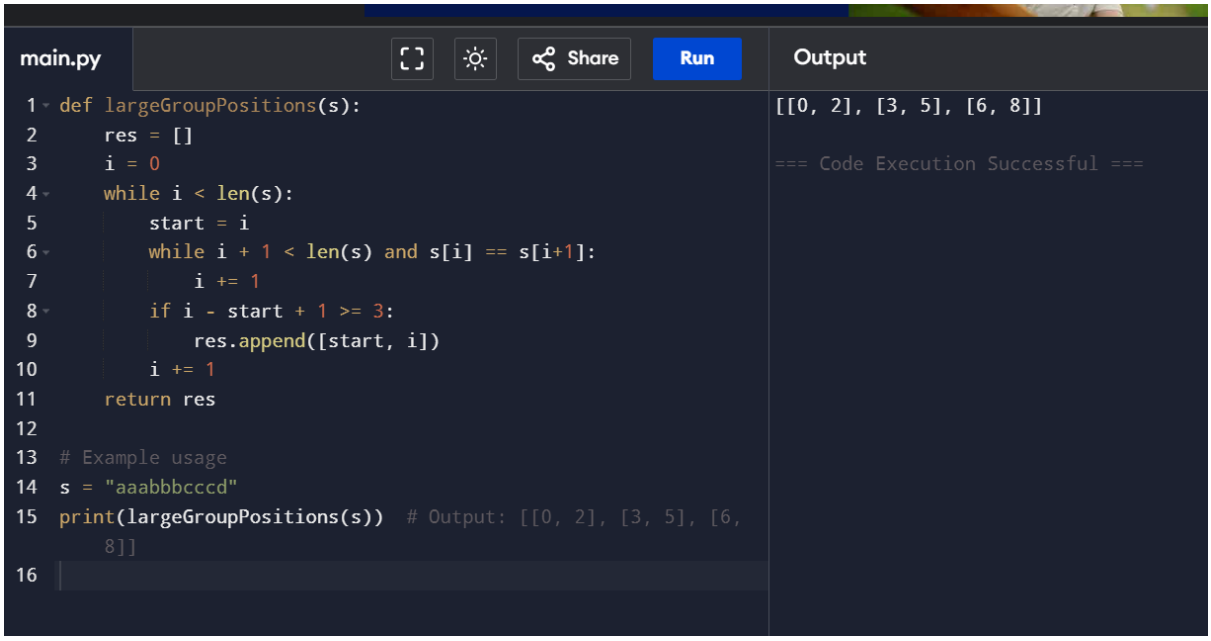
1. Iterate through string with two pointers.
2. If group length ≥ 3, store [start, end].

Input

s = "abbxxxxzzy"

Output

[[3, 6]]



16.

Game of Life

Aim

Compute the next state of Conway's Game of Life grid.

Algorithm

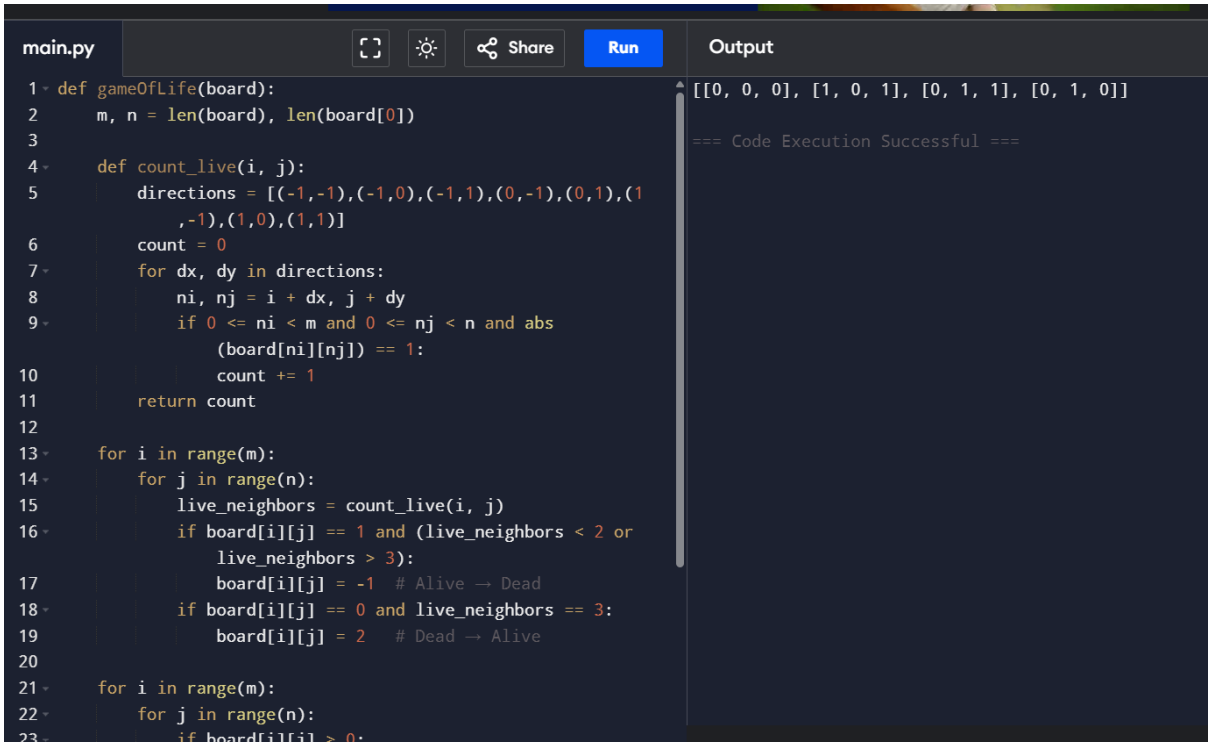
1. For each cell, count live neighbors.
2. Apply rules:
    - <2 → dies
    - 2 or 3 → lives
    - 3 → dies
    - dead + 3 neighbors → live

Input

[[0,1,0],[0,0,1],[1,1,1],[0,0,0]]

Output

[[0,0,0],[1,0,1],[0,1,1],[0,1,0]]



17.

Champagne Tower

Aim

Find how full the query_glass is after pouring poured cups.

Algorithm

1. Use DP.
2. Each glass spills extra equally to next row (left and right).
3. Stop when reaching query_row.

Input

poured=2, query_row=1, query_glass=1

Output

0.5

main.py

Share Run

```python
def champagneTower(poured, query_row, query_glass):
    tower = [[0]*101 for _ in range(101)]
    tower[0][0] = poured

    for r in range(query_row + 1):
        for c in range(r + 1):
            if tower[r][c] > 1:
                excess = (tower[r][c] - 1) / 2
                tower[r+1][c] += excess
                tower[r+1][c+1] += excess
                tower[r][c] = 1

    return tower[query_row][query_glass]

# Example usage
poured = 10
query_row = 3
query_glass = 1
print(champagneTower(poured, query_row, query_glass))  # Output: 1
```

Output

```
1

=== Code Execution Successful ===
```