934. Shortest Bridge

You are given an `n x n` binary matrix `grid` where `1` represents land and `0` represents water.

An **island** is a 4-directionally connected group of `1`'s not connected to any other `1`'s. There are **exactly two islands** in `grid`.

You may change `0`'s to `1`'s to connect the two islands to form **one island**.

Return *the smallest number of `0`'s you must flip to connect the two islands*.

Example 1:

```
Input: grid = [[0,1],[1,0]]
Output: 1
```

Example 2:

```
Input: grid = [[0,1,0],[0,0,0],[0,0,1]]
Output: 2
```

Example 3:

```
Input: grid = [[1,1,1,1,1],[1,0,0,0,1],[1,0,1,0,1],[1,0,0,0,1],[1,1,1,1,1]]
Output: 1
```

Constraints:

- `n == grid.length == grid[i].length`
- `2 <= n <= 100`
- `grid[i][j]` is either `0` or `1`.
- There are exactly two islands in `grid`.

```python
class Solution:
    def shortestBridge(self, grid: List[List[int]]) -> int:
        queue = []
        flag = True
        i = 0

        visited = [[False for j in range(len(grid[0]))] for i in
range(len(grid))]
        while i<len(grid) and flag!=False:
```

```
            j = 0
            while j<len(grid[0]) and flag!=False:
                if grid[i][j]==1:
                    self.dfs(grid,i,j,queue,visited)
                    flag = False
                j=j+1
            i=i+1
        limit = 0
        # print(queue)
        while len(queue):
            length = len(queue)
            while length:
                x,y = queue.pop(0)
                for dx,dy in [(-1,0),(0,1),(1,0),(0,-1)]:
                    r = x+dx
                    c = y+dy
                    if r<0 or c<0 or r>=len(grid) or c>=len(grid[0]) or
visited[r][c]==True:
                        continue
                    if grid[r][c]==1:
                        return limit
                    queue.append((r,c))
                    visited[r][c]=True
                length = length-1
            limit = limit+1

    def dfs(self,grid,r,c,queue,visited):
        if r<0 or c<0 or r>=len(grid) or c>=len(grid[0]) or grid[r][c]==0 or
visited[r][c]==True:
            return
        visited[r][c] = True
        queue.append((r,c))
        self.dfs(grid,r-1,c,queue,visited)
        self.dfs(grid,r,c+1,queue,visited)
        self.dfs(grid,r+1,c,queue,visited)
        self.dfs(grid,r,c-1,queue,visited)
```