637. Average of Levels in Binary Tree

```
class Solution:
    def averageOfLevels(self, root: TreeNode) -> List[float]:
        if not root:
            return []
        cur = [root]
        result = []
        while cur:
            next_level = []
            cur_sum = 0
            for node in cur:
                cur_sum += node.val
                if node.left:
                    next_level.append(node.left)
                if node.right:
                    next_level.append(node.right)
            result.append(cur_sum / len(cur))
            cur = next_level
        return result
```

63. Unique Paths II

O(n**2)

```
class Solution:
    def uniquePathsWithObstacles(self, obstacleGrid: List[List[int]]) -> int:
        if not obstacleGrid or not obstacleGrid[0] or obstacleGrid[0][0] == 1:
            return 0
        rows = len(obstacleGrid)
        cols = len(obstacleGrid[0])
        obstacleGrid[0][0] = 1
        for i in range(rows):
            for j in range(cols):
                if i == 0 and j == 0:
                    obstacleGrid[i][j] = 1
                elif obstacleGrid[i][j] == 1:
                    obstacleGrid[i][j] = 0
                else:
                    top = obstacleGrid[i - 1][j] if i > 0 else 0
                    left = obstacleGrid[i][j - 1] if j > 0 else 0
                    obstacleGrid[i][j] = top + left
        return obstacleGrid[-1][-1]
```

Shortest Path:

O(n**2)

```python
def shortest_path(path):
   if not path or not path[0]:
       return -1
   directions = [[0, 1], [0, -1], [1, 0], [-1, 0]]
   cur_steps = set()
   visited = set()
   cur_steps.add((0, 0))
   rows, cols = len(path), len(path[0])
   steps = 0
   if rows == 1 and cols == 1:
       return 0
   while cur_steps:
       next_steps = set()
       steps += 1
       for x, y in cur_steps:
           for d_x, d_y in directions:
               new_x, new_y = x + d_x, y + d_y
               if (new_x, new_y) not in visited and 0 <= new_x < rows and 0 <= new_y <
cols and path[new_x][new_y] == 0:
                   if new_x == rows - 1 and new_y == cols - 1:
                       return steps
                   next_steps.add((new_x, new_y))
       cur_steps = next_steps
   return -1


path = [[0, 0, 0], [0, 1, 0], [0, 0, 0]]
print(shortest_path(path) == 4)
path = [[0]]
print(shortest_path(path) == 0)
path = [[0, 1, 0]]
print(shortest_path(path) == -1)
```

981. Time Based Key-Value Store
from collections import defaultdict
from bisect import *
class TimeMap:

```
    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.memo = defaultdict(list)
        self.timestamp = defaultdict(list)

    def set(self, key: str, value: str, timestamp: int) -> None:
        self.memo[key].append(value)
        self.timestamp[key].append(timestamp)

    def get(self, key: str, timestamp: int) -> str:
        idx = bisect(self.timestamp[key], timestamp)
        return self.memo[key][idx - 1] if idx else ""
```

251. Flatten 2D Vector

```
from collections import deque
class Vector2D:

    def __init__(self, v: List[List[int]]):
        self.q = deque()
        for arr in v:
            self.q.extend(arr)

    def next(self) -> int:
        return self.q.popleft()

    def hasNext(self) -> bool:
        return len(self.q) > 0
```