Name:-Rushikesh Thorat

```
from typing import List, Tuple
from collections import defaultdict, deque

def drone_pathfinding(drones: List[Tuple[int, int, int, int, int]]) -> List[List[Tuple[int, int]]]:
    # Create a graph representation of the grid, where each cell is a node
    graph = defaultdict(list)
    for i in range(20):
        for j in range(20):
            if i > 0:
                graph[(i, j)].append((i-1, j))
            if i < 19:
                graph[(i, j)].append((i+1, j))
            if j > 0:
                graph[(i, j)].append((i, j-1))
            if j < 19:
                graph[(i, j)].append((i, j+1))

    # Create a dictionary of drone schedules, where each time step is a key
    schedules = defaultdict(list)
    for drone in drones:
        for t in range(drone[4], drone[4]+abs(drone[0]-drone[2])+abs(drone[1]-drone[3])+1):
            schedules[t].append(drone)

    # Use BFS to find the shortest path for each drone at each time step
    paths = []
    for t in sorted(schedules.keys()):
        positions = {drone[:2]: drone for drone in schedules[t]}
        visited = set(positions.keys())
```

```
    queue = deque(positions.keys())
    while queue:
        pos = queue.popleft()
        for neighbor in graph[pos]:
            if neighbor in visited:
                continue
            visited.add(neighbor)
            queue.append(neighbor)
 if neighbor in positions:
 drone = positions[neighbor]
 paths.append([(drone[0]+i*(drone[2]-drone[0])//abs(drone[0]-drone[2]), drone[1]+i*(drone[3]-drone[1])//abs(drone[1]-drone[3])) for i in range(abs(drone[0]-drone[2])+abs(drone[1]-drone[3])+1)])
 positions.pop(neighbor)
 if not positions:
        break
  if not positions:
        break
        return paths
```