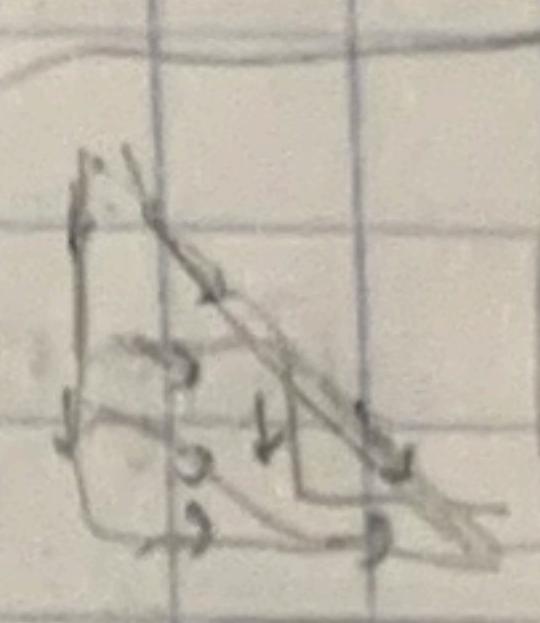


6.

- a) Use Dijkstra's, with a graph only directed right and down with end node being goal, start node being where the robot starts
- b) Use Dijkstra's again / BFS
- c) Use Dynamic Programming
- d) This is DFS,

grid = 2d array of nodes that are spaces
end = grid [R-1] [C-1]



```
def recurse (position in grid).  
    if (position == end):  
        return 1  
    if can go down and right:  
        return recurse (Position(j-1)) + recurse (Position(i+1))  
  
    if only can go down  
        return recurse (Position(j-1))  
  
    if only can go right  
        return recurse (Position(i+1))
```

Time complexity: $O(V+E)$, this is because V is number of vertexes (possible squares) E is paths from those squares. Each path has at most 2 possible paths, it is a simple DFS algorithm

Space complexity: $O(R \cdot C)$, this is for the initial grid, nothing else is taking up space besides $O(1)$ returns

One thing I could improve is by using some dynamic programming algorithms, it might improve the time complexity a bit.