Backtracking

- Backtracking
  - Yes/No
  - Yes/No (what is the path)
- Dynamnamic Programming / DP
  - Best solutionm
  - Min/Max

Function structure:

- Boundary
- Invalid State
- Valid/Final State
- Recursion: all possible paths from here

In [ ]:

```
Neighbors
```

In [ ]:

```
```

In [ ]:

```
- is there a solution possible
  Maze: matrix [1, 0]




[R, D]
   [
     1  1  1  1
     0  1  0  1
     0  1  1  1
   ]

[U,D,L,R]
   [
     1  1  1  1
     0  0  1  1
     0  1  1  1
     1  1  0  0
     1  1  1  1
   ]

[U,D,L,R, UL, UR, DL, DR]
   [
     1  1  1  1
     0  0  1  1
     0  0  1  1
     1  1  0  0
     1  1  1  1
   ]


- Find path with max reward/min penalty
DP
   [
     1  2  1  1
     0  5  0  6
     0  1  1  1
   ]
```

In [ ]:

```
```

In [ ]:

```
```

V1: Rat in a Maze D=(R,D), Return a Boolean

```
Given a maze of size N*N represented in the form of 0s and 1s.
Where 1 denotes a cell that can be visisted and 0 denotes a blocker/s
tone/obstacle

Given a starting point (0,0) and and ending point (N-1, N-1), return
whether there exists a path or not.
At a time the Rat can move only one step in Right or Down directions
only.
      0  1  2  3
   [
0     1  1  1  1
1     0  1  0  1
2     0  1  1  1
3     0  1  1  1
   ]
```

In [22]:

```
maze1 = [
   [ 1,  1,  1,  1 ],
   [ 0,  1,  0,  1 ],
   [ 0,  1,  1,  1 ],
   [ 0,  1,  1,  1 ],
   ]

maze2 = [
     [1,  1,  1,  1,  1],
     [0,  0,  1,  1,  1],
     [0,  0,  1,  1,  0],
     [1,  1,  0,  0,  0],
     [1,  1,  1,  1,  1],
   ]

maze3 = [
   [ 1,  1,  1,  1 ],
   [ 1,  1,  1,  1 ],
   [ 1,  1,  1,  1 ],
   [ 1,  1,  1,  0 ],
]

maze4 = [
     [1,  1,  1,  1,  1],
     [0,  0,  1,  1,  1],
     [0,  1,  1,  1,  0],
     [1,  1,  0,  0,  0],
     [1,  1,  1,  1,  1],
   ]

def find_path(maze):
    return find_path_util(maze, 0, 0)


def find_path_util(maze, x, y):

    side = len(maze)
    if x <0 or y < 0 or x >= side or y == side:
        return False

    if maze[x][y] == 0:
        return False

    if x == side-1 and y == side-1:
        return True

    return find_path_util(maze,x,y+1) or find_path_util(maze, x+1, y)

print(find_path(maze1))
print(find_path(maze2))
print(find_path(maze3))
print(find_path(maze4))
#      0   1   2   3
#  0 [ 1,  1,  1,  1 ],
#  1 [ 0,  1,  0,  1 ],
#  2 [ 0,  1,  1,  1 ],
#  3 [ 0,  1,  1,  1 ],
#                                                     f(0,0)
#                                     f(0,1)
#                         F(0,2)
#                 f(0,3)
#             f(0,4)
```

```
True
False
False
False
```

In [ ]:

In [ ]:

In [ ]:

V2: Rat in a Maze D=(R,D), Return a list of coordinates which denotes the path in maze

```
Given a maze of size N*N represented in the form of 0s and 1s.
Where 1 denotes a cell that can be visisted and 0 denotes a blocker/s
tone/obstacle

Given a starting point (0,0) and and ending point (N-1, N-1), return
whether there exists a path or not.
At a time the Rat can move only one step in Right or Down directions
only.
      0  1  2  3
   [
0     1  1  1  1
1     0  1  0  1
2     0  1  1  1
3     0  1  1  1
   ]

00,01,02,03,13,23,33
...
multiple options
```

In [21]:

```
maze1 = [
   [ 1,  1,  1,  1 ],
   [ 0,  1,  0,  1 ],
   [ 0,  1,  1,  1 ],
   [ 0,  1,  1,  1 ],
   ]

maze2 = [
     [1,  1,  1,  1,  1],
     [0,  0,  1,  1,  1],
     [0,  0,  1,  1,  0],
     [1,  1,  0,  0,  0],
     [1,  1,  1,  1,  1],
   ]

maze3 = [
   [ 1,  1,  1,  1 ],
   [ 1,  1,  1,  1 ],
   [ 1,  1,  1,  1 ],
   [ 1,  1,  1,  0 ],
]

maze4 = [
   [ 1,  1,  1,  1 ],
   [ 0,  1,  0,  1 ],
   [ 0,  1,  1,  0 ],
   [ 0,  0,  1,  1 ],
]

ans = []
def find_path(maze):
    stack = []
    global ans
    ans = []
    find_path_util(maze, 0, 0, stack)
    return ans


def find_path_util(maze, x, y, stack):

    side = len(maze)
    if x <0 or y < 0 or x >= side or y == side:
        return False

    if maze[x][y] == 0:
        return False

    stack.append((x,y))
    if x == side-1 and y == side-1:
        global ans
        ans = stack.copy()
        return True

    res = find_path_util(maze,x,y+1, stack) or find_path_util(maze, x+1, y, st
    stack.pop()
    return res
```

```python
print(find_path(maze1))
print(find_path(maze2))
print(find_path(maze3))
print(find_path(maze4))
```

```
[(0, 0), (0, 1), (0, 2), (0, 3), (1, 3), (2, 3), (3, 3)]
[]
[]
[(0, 0), (0, 1), (1, 1), (2, 1), (2, 2), (3, 2), (3, 3)]
```

In [ ]:

V3: Rat in a Maze D=(L,R,D,U), Return a list of coordinates which denotes the path in maze

```
Given a maze of size N*N represented in the form of 0s and 1s.
Where 1 denotes a cell that can be visisted and 0 denotes a blocker/stone/obstacle

Given a starting point (0,0) and and ending point (N-1, N-1), return whether there exists a path or not.
At a time the Rat can move only one step in Right or Down or Up or Left directions only.
      0  1  2  3
   [
0     1  1  1  1
1     0  1  0  1
2     0  1  1  1
3     0  1  1  1
   ]
```

In [2]:

```
maze1 = [
   [ 1,  1,  1,  1 ],
   [ 0,  1,  0,  1 ],
   [ 0,  1,  1,  1 ],
   [ 0,  1,  1,  0 ],
   ]

maze2 = [
     [1,  1,  1,  1,  1],
     [0,  0,  1,  1,  1],
     [0,  0,  1,  1,  0],
     [1,  1,  0,  0,  0],
     [1,  1,  1,  1,  1],
   ]

maze3 = [
     [1,  1,  1,  1,  1],
     [0,  0,  1,  1,  1],
     [0,  1,  1,  1,  0],
     [1,  1,  0,  0,  0],
     [1,  1,  1,  1,  1],
   ]


def find_path(maze):
    visited = set()
    return find_path_util(maze, 0, 0, visited)


def find_path_util(maze, x, y, visited):

    side = len(maze)
    if x <0 or y < 0 or x >= side or y == side:
        return False

    if maze[x][y] == 0:
        return False

    if x == side-1 and y == side-1:
        return True

    if (x,y) in visited:
        return False

    visited.add((x,y))
    print(x,y)
    return find_path_util(maze,x,y+1,visited) or \
            find_path_util(maze, x+1, y,visited) or \
            find_path_util(maze, x-1, y,visited) or \
            find_path_util(maze, x, y-1,visited)

print(find_path(maze1))
print(find_path(maze2))
print(find_path(maze3))
```

```
0 0
0 1
0 2
0 3
1 3
2 3
2 2
3 2
3 1
2 1
1 1
False
0 0
0 1
0 2
0 3
0 4
1 4
1 3
2 3
2 2
1 2
False
0 0
0 1
0 2
0 3
0 4
1 4
1 3
2 3
2 2
1 2
2 1
3 1
4 1
4 2
4 3
True
```

In [ ]:

https://www.hackerrank.com/contests/noi-ph-practice-page/challenges/path-in-a-maze (https://www.hackerrank.com/contests/noi-ph-practice-page/challenges/path-in-a-maze)

https://www.hackerrank.com/challenges/maze-escape (https://www.hackerrank.com/challenges/maze-escape)

In [ ]: