2D

https://leetcode.com/problems/unique-paths/description/
(https://leetcode.com/problems/unique-paths/description/)

Brute Force

```
class Solution {
public:
    int uniquePaths(int m, int n) {
        return count(0,0,m,n);
    }

    int count(int i, int j, int m, int n) {
        if (i == m-1 && j == n-1) {
            return 1;
        } else if (i >=m || j >=n) {
            return 0;
        }

        return count(i+1, j, m,n) + count(i, j+1, m, n);
    }
};
```

Memoization/Recursive

```
TC: O(m*n)
SC: O(m*n)
```

```
class Solution {
    public int uniquePaths(int m, int n) {

        int dp[][] = new int[m][n];

        for(int[] x : dp)
```

```
class Solution {
public:
    int uniquePaths(int m, int n) {
        map< pair<int,int> , int> cache;
        return count(0,0,m,n, cache);
    }

    int count(int i, int j, int m, int n, map< pair<int,int> , int> &
cache) {
        if (i == m-1 && j == n-1) {
            return 1;
        } else if (i >=m || j >=n) {
            return 0;
        }

        if (cache.count( make_pair(i,j)) > 0){
            return cache[make_pair(i,j)];
        }
        cache[make_pair(i,j)] = count(i+1, j, m,n, cache) + count(i,
j+1, m, n, cache);
        return cache[make_pair(i,j)];
    }
};
```

Tabulation/Iterative

TC: O(m * n)
SC: O(m * n)

```java
class Solution {
    public int uniquePaths(int m, int n) {
        int[][] dp = new int[m][n];

        // Base cases
        for (int i = 0; i < m; i++) {
            dp[i][0] = 1;
        }
        for (int j = 0; j < n; j++) {
            dp[0][j] = 1;
        }
```

```java
class Solution {
    public int uniquePaths(int m, int n) {
      int count[][] = new int[m][n];
        for (int i = 0; i < m; i++)
            count[i][0] = 1;
        for (int j = 0; j < n; j++)
            count[0][j] = 1;
        for (int i = 1; i < m; i++) {
            for (int j = 1; j < n; j++)
                count[i][j] = count[i - 1][j] + count[i][j - 1];
        }
        return count[m - 1][n - 1];
    }
}
```

Nikon
TC: O(m*n)
SC: O(n)

```python
class Solution:
    def uniquePaths(self, m: int, n: int) -> int:
        row = [1]*n
        for i in range(m-1):
            newRow = [1]*n
            for j in range(n-2, -1, -1):
                newRow[j] = newRow[j+1] + row[j]
            row = newRow
        return row[0]
```

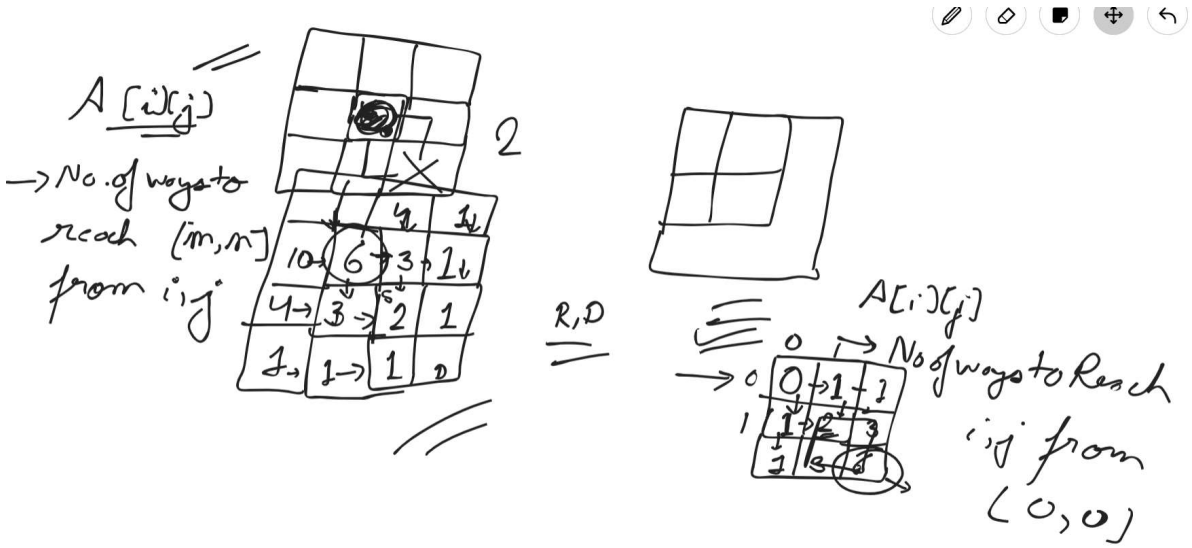
TC: O(m * n)
SC: O(n)

```java
class Solution {
    public int uniquePaths(int m, int n) {
        int[] dp = new int[n];

        for (int i = 0; i < n; i++) {
            dp[i] = 1;
        }

        for (int i = 1; i < m; i++) {
            for (int j = 1; j < n; j++) {
                dp[j] = dp[j-1] + dp[j];
            }
        }

        return dp[n-1];
    }
```



In [ ]:

In [ ]:
```java
class Solution {
    public int uniquePaths(int m, int n) {
        long ans = 1;
        for(int i = 1; i <= m-1; i++){
            ans = ans*(n-1+i)/i;
        }
        return (int)ans;
    }
}
```

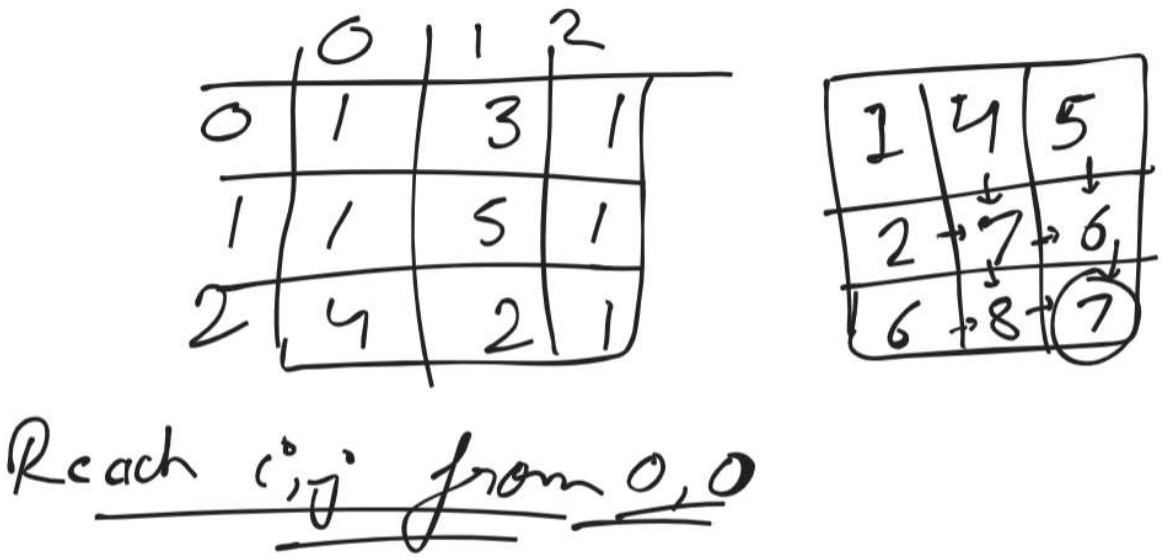
In [ ]:

https://leetcode.com/problems/minimum-path-sum/submissions/1170637068/
(https://leetcode.com/problems/minimum-path-sum/submissions/1170637068/)

Brute Force

```
class Solution {
    public int minPathSum(int[][] grid) {
        return minPathSumUtil(grid, 0,0);
    }

    public int minPathSumUtil(int[][] grid, int i , int j) {
        if (i >= grid.length || j >= grid[0].length) {
            return Integer.MAX_VALUE;
        } else if (i == grid.length - 1 && j == grid[0].length - 1) {
            return grid[i][j];
        }
        return grid[i][j] + Math.min(minPathSumUtil(grid, i+1, j), mi
nPathSumUtil(grid, i, j+1));
    }
}
```



In [ ]: