Unique Paths - Detailed Notes with Code

1. Recursive Solution

Idea: Explore all paths by moving either right or down recursively. Time Complexity: Exponential O(2^(m+n)). Space Complexity: O(m+n) (recursion stack).

```
class Solution {
public:
    int f(int i, int j){
        if(i == 0 && j == 0) return 1;
        if(i < 0 || j < 0) return 0;

        int up = f(i - 1, j);
        int left = f(i, j - 1);

        return up + left;
    }

    int uniquePaths(int m, int n) {
        return f(m - 1, n - 1);
    }
};
```

2. Recursive + Memoization

Idea: Use recursion but store (memoize) intermediate results to avoid recomputation. Time Complexity: O(m*n). Space Complexity: O(m*n) + recursion stack.

```
class Solution {
public:
    int f(int i, int j, vector<vector<int>>& dp){
        if(i == 0 && j == 0) return 1;
        if(i < 0 || j < 0) return 0;

        if(dp[i][j] != -1) return dp[i][j];

        int up = f(i - 1, j, dp);
        int left = f(i, j - 1, dp);

        return dp[i][j] = up + left;
    }

    int uniquePaths(int m, int n) {
        vector<vector<int>> dp(m, vector<int>(n, -1));
        return f(m - 1, n - 1, dp);
    }
};
```

3. Tabulation (Bottom-Up DP)

Idea: Iteratively fill a dp table where each cell is the sum of top and left cells. Time Complexity: O(m*n). Space Complexity: O(m*n).

```
class Solution {
public:
    int uniquePaths(int m, int n) {
        vector<vector<int>> dp(m, vector<int>(n, -1));

        for(int i = 0; i < m; i++){
            for(int j = 0; j < n; j++){
                if(i == 0 && j == 0){
                    dp[i][j] = 1;
                }
                else{
```

```
                int up = 0, left = 0;
                if(i > 0) up = dp[i - 1][j];
                if(j > 0) left = dp[i][j - 1];
                dp[i][j] = up + left;
            }
        }
    }
    return dp[m - 1][n - 1];
}
};
```

4. Space Optimized DP

Idea: Instead of keeping a full dp table, use only one array to keep track of the current row. Time Complexity: O(m*n). Space Complexity: O(n).

```
class Solution {
public:
    int uniquePaths(int m, int n) {
        vector<int> prevRow(n, 0);

        for(int i = 0; i < m; i++){
            vector<int> cur(n, 0);
            for(int j = 0; j < n; j++){
                if(i == 0 && j == 0){
                    cur[j] = 1;
                }
                else{
                    int up = 0, left = 0;
                    if(i > 0) up = prevRow[j];
                    if(j > 0) left = cur[j - 1];
                    cur[j] = up + left;
                }
            }
            prevRow = cur;
        }
        return prevRow[n - 1];
    }
};
```