Unique Paths Problem - Solutions Notes

1. Recursive Solution

- Idea: Start from (0,0) and move only right or down until reaching (m-1,n-1). - Base Case: If out of bounds → 0, if at destination → 1. - Time Complexity: O(2^(m+n)), very high (exponential). - Space Complexity: O(m+n) recursion stack.

```
class Solution {
public:
    int countPaths(int i, int j, int m, int n) {
        if(i >= m || j >= n) return 0;
        if(i == m-1 && j == n-1) return 1;
        return countPaths(i+1, j, m, n) + countPaths(i, j+1, m, n);
    }

    int uniquePaths(int m, int n) {
        return countPaths(0, 0, m, n);
    }
};
```

2. Recursive + Memoization

- Optimization: Store results of subproblems in DP table. - Avoids recomputation of overlapping subproblems. - Time Complexity: O(m*n). - Space Complexity: O(m*n) + recursion stack.

```
class Solution {
public:
    int solve(int i, int j, int m, int n, vector<vector<int>>& dp) {
        if(i >= m || j >= n) return 0;
        if(i == m-1 && j == n-1) return 1;
        if(dp[i][j] != -1) return dp[i][j];
        return dp[i][j] = solve(i+1, j, m, n, dp) + solve(i, j+1, m, n, dp);
    }

    int uniquePaths(int m, int n) {
        vector<vector<int>> dp(m, vector<int>(n, -1));
        return solve(0, 0, m, n, dp);
    }
};
```

3. Tabulation (Bottom-Up DP)

- Build DP table iteratively. - dp[i][j] = dp[i-1][j] + dp[i][j-1]. - Base Case: dp[0][0] = 1. - 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, 0));
        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 = (i > 0) ? dp[i-1][j] : 0;
                    int left = (j > 0) ? dp[i][j-1] : 0;
                    dp[i][j] = up + left;
                }
            }
        }
        return dp[m-1][n-1];
    }
```

```
};
```

4. Space Optimization

- Use only two rows (previous + current). - Reduces space from O(m*n) → O(n). - Time Complexity: O(m*n). - Space Complexity: O(n).

```
class Solution {
public:
    int uniquePaths(int m, int n) {
        vector<int> prev(n, 0);
        for(int i = 0; i < m; i++) {
            vector<int> curr(n, 0);
            for(int j = 0; j < n; j++) {
                if(i == 0 && j == 0) curr[j] = 1;
                else {
                    int up = (i > 0) ? prev[j] : 0;
                    int left = (j > 0) ? curr[j-1] : 0;
                    curr[j] = up + left;
                }
            }
            prev = curr;
        }
        return prev[n-1];
    }
};
```