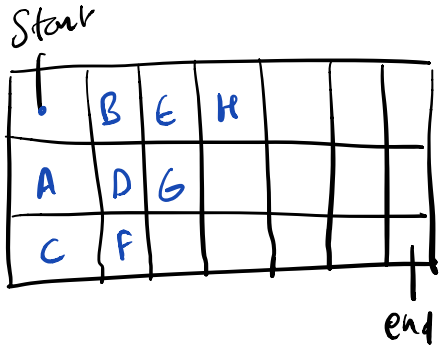
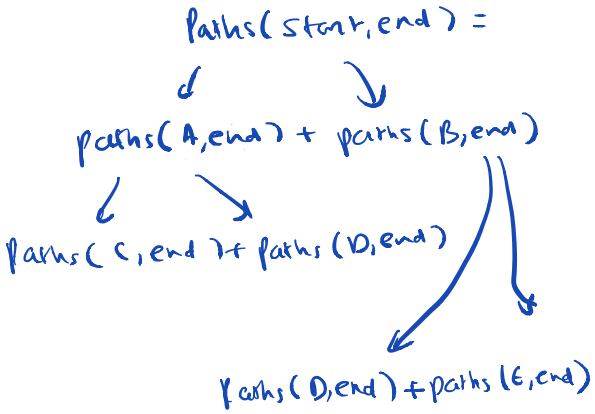
#62: Unique Paths



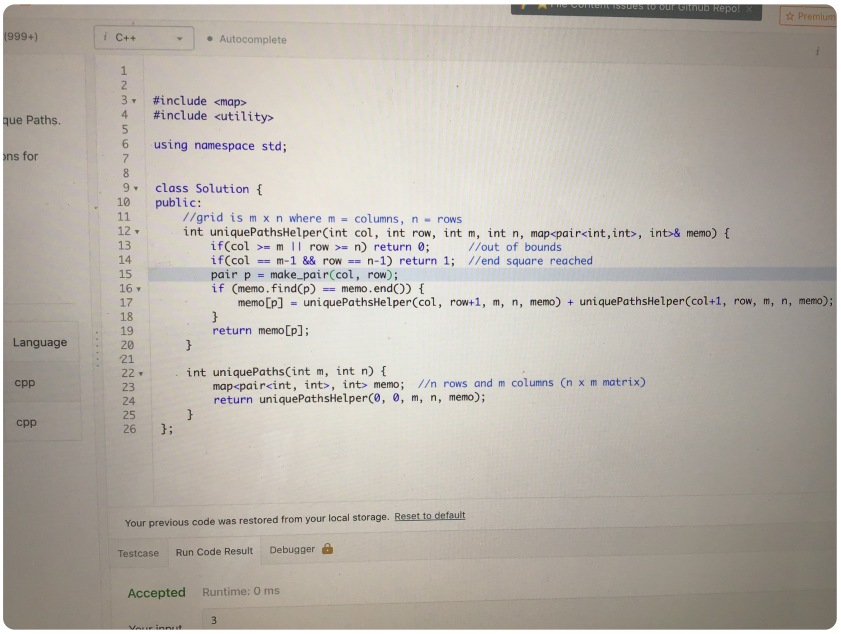



```
//Naive recursion
paths (bool grid[][], int row, int col) {
    if(!withinBounds(row, col)) return 0;
    if( reachEnd(row, col)) return 1;
    return (paths(grid, row+1, col) + paths(grid, row, col+1));
}
```

- Naive Recursive Solution (Time-limit exceeded)

```
3
4   class Solution {
5   public:
6       //grid is m x n where m = columns, n = rows
7       int uniquePathsHelper(int col, int row, int m, int n) {
8           if(col >= m || row >= n) return 0;       //out of bounds
9           if(col == m-1 && row == n-1) return 1;  //end square reached
10          return uniquePathsHelper(col, row+1, m, n) + uniquePathsHelper(col+1, row, m, n);
11      }
12
13      int uniquePaths(int m, int n) {
14          return uniquePathsHelper(0, 0, m, n);
15      }
16  };
```

- Approach #2 Top down / memoization



```cpp
#include <map>
#include <utility>

using namespace std;

class Solution {
public:
    //grid is m x n where m = columns, n = rows
    int uniquePathsHelper(int col, int row, int m, int n, map<pair<int,int>, int>& memo) {
        if(col >= m || row >= n) return 0;      //out of bounds
        if(col == m-1 && row == n-1) return 1;  //end square reached
        pair p = make_pair(col, row);
        if (memo.find(p) == memo.end()) {
            memo[p] = uniquePathsHelper(col, row+1, m, n, memo) + uniquePathsHelper(col+1, row, m, n, memo);
        }
        return memo[p];
    }

    int uniquePaths(int m, int n) {
        map<pair<int, int>, int> memo;  //n rows and m columns (n x m matrix)
        return uniquePathsHelper(0, 0, m, n, memo);
    }
};
```

Your previous code was restored from your local storage. Reset to default

Testcase | Run Code Result | Debugger

Accepted Runtime: 0 ms

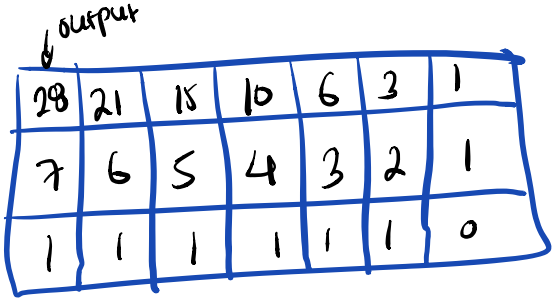
$O(n^2)$

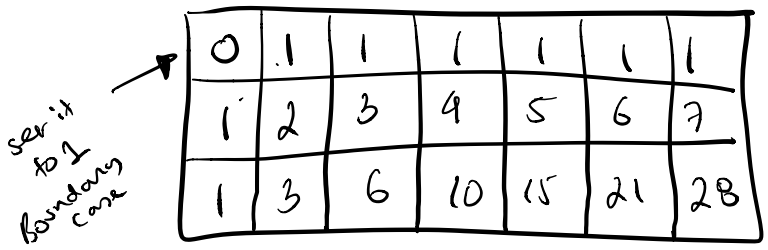
std::map takes std::pair as key

std::unordered_map does NOT

Approach #3 - Bottom-up dynamic Programming

7x3






```cpp
#include <map>
#include <utility>

using namespace std;


class Solution {
public:
    //grid is m x n where m = columns, n = rows

    int uniquePaths(int m, int n) {
        vector<int> columns(m, 0);
        vector<vector<int>> grid(n, columns); //n rows of m columns [nxm] matrix

        //boundary case of one square gird
        grid[0][0] = 1;
        for(int col=1; col < m; col++) {
            grid[0][col] = 1;
        }
        for(int row = 1; row < n; row++) {
            grid[row][0] = 1;
        }
        for (int row = 1; row < n; row++) {
            for (int col = 1; col < m; col++) {
                grid[row][col] = grid[row-1][col] + grid[row][col-1];
            }
        }

        return grid[n-1][m-1];
    }
};
```