2684. Maximum Number of Moves in a Grid

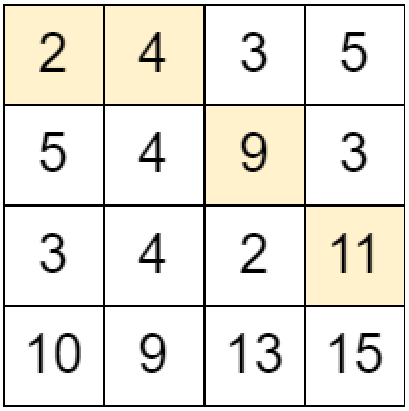
You are given a **0-indexed** `m x n` matrix `grid` consisting of **positive** integers.

You can start at **any** cell in the first column of the matrix, and traverse the grid in the following way:

- From a cell `(row, col)`, you can move to any of the cells: `(row - 1, col + 1)`, `(row, col + 1)` and `(row + 1, col + 1)` such that the value of the cell you move to, should be **strictly** bigger than the value of the current cell.

Return *the* ***maximum*** *number of* ***moves*** *that you can perform.*

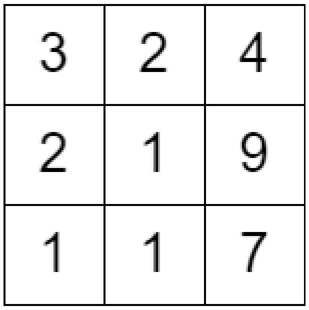
Example 1:



2	4	3	5
5	4	9	3
3	4	2	11
10	9	13	15

```
Input: grid = [[2,4,3,5],[5,4,9,3],[3,4,2,11],
[10,9,13,15]]
Output: 3
Explanation: We can start at the cell (0, 0) and make
the following moves:
- (0, 0) -> (0, 1).
- (0, 1) -> (1, 2).
- (1, 2) -> (2, 3).
It can be shown that it is the maximum number of moves
that can be made.
```

Example 2:



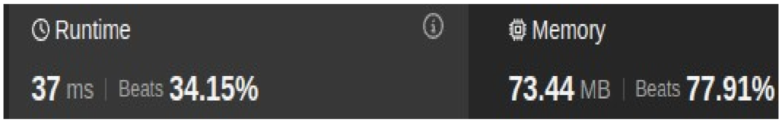
3	2	4
2	1	9
1	1	7

```
Input: grid = [[3,2,4],[2,1,9],[1,1,7]]
Output: 0
Explanation: Starting from any cell in the first column we
cannot perform any moves.
```

Constraints:

- `m == grid.length`
- `n == grid[i].length`
- `2 <= m, n <= 1000`
- `4 <= m * n <= 105`
- `1 <= grid[i][j] <= 106`

2684. Maximum Number of Moves in a Grid



Runtime

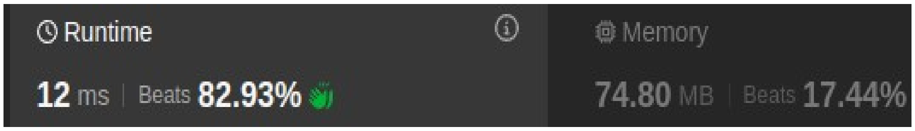
37 ms | Beats 34.15%

Memory

73.44 MB | Beats 77.91%

```
/*
  BFS
  Time complexity: O(mn)
  Space compexity:O(mn)
*/
class Solution {
  public:
    int maxMoves(std::vector<std::vector<int>>& grid){
      int m=grid.size();
      int n=grid[0].size();
      std::vector<std::vector<bool>> visited(m,std::vector<bool>(n,false));
      std::queue<std::tuple<int,int,int>> q;
      for(int row=0;row<m;++row) {
        q.push({row,0,0});
        visited[row][0]=true;
      }
      int ans=0;
      while(!q.empty()){
        auto [row,col,cnt]=q.front();
        q.pop();
        ans=std::max(ans,cnt);
        if(row-1>=0 && col+1<n && !visited[row-1][col+1] && grid[row][col]<grid[row-1][col+1]){
          visited[row-1][col+1]=true;
          q.push({row-1,col+1,cnt+1});
        }
        if(col+1<n && !visited[row][col+1] && grid[row][col]<grid[row][col+1]){
          visited[row][col+1]=true;
          q.push({row,col+1,cnt+1});
        }
        if(row+1<m && col+1<n && !visited[row+1][col+1] && grid[row][col]<grid[row+1][col+1]){
          visited[row+1][col+1]=true;
          q.push({row+1,col+1,cnt+1});
        }
      }
      return ans;
    }
};
```

2684. Maximum Number of Moves in a Grid



Runtime

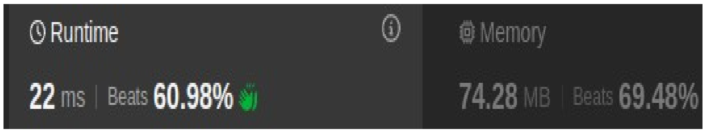
12 ms | Beats 82.93%

Memory

74.80 MB | Beats 17.44%

```
/*
    Recursion+memoization
    Time complexity: O(mn)
    Space compexity:O(mn)
*/
class Solution {
    public:
        int maxMoves(std::vector<std::vector<int>>& grid){
            int m=grid.size();
            int n=grid[0].size();

            std::vector<std::vector<int>> memo(m,std::vector<int>(n,-1));

            auto solve=[&](int row,int col,auto& self)->int{
                if(row>=m || col>=n) return 0;

                if(memo[row][col]!=-1) return memo[row][col];

                int up=0,down=0,right=0;

                if(row-1>=0 && col+1<n && grid[row][col]<grid[row-1][col+1]){
                    up=self(row-1,col+1,self);
                }

                if(col+1<n && grid[row][col]<grid[row][col+1]){
                    right=self(row,col+1,self);
                }

                if(row+1<m && col+1<n && grid[row][col]<grid[row+1][col+1]){
                    down=self(row+1,col+1,self);
                }

                return memo[row][col]=1+std::max({up,right,down});
            };

            int ans=0;
            for(int row=0;row<m;++row){
                ans=std::max(ans,solve(row,0,solve));
            }

            return ans-1;
        }
};
```

2684. Maximum Number of Moves in a Grid



Runtime

22 ms | Beats 60.98%

Memory

74.28 MB | Beats 69.48%

```
/*
   Bottom up with 2D array
   Time complexity: O(mn)
   Space compexity:O(mn)
*/
class Solution {
   public:
      int maxMoves(std::vector<std::vector<int>>& grid){
         int m=grid.size();
         int n=grid[0].size();

         std::vector<std::vector<int>> dp(m,std::vector<int>(n,0));

         for(int row=0;row<m;++row) dp[row][0]=1;

         int ans=0;

         for(int col=1;col<n;++col){
            for(int row=0;row<m;++row){
               if(row-1>=0 && grid[row][col]>grid[row-1][col-1] && dp[row-1][col-1]>0)
                  dp[row][col]=std::max(dp[row][col],dp[row-1][col-1]+1);
               if(grid[row][col]>grid[row][col-1] && dp[row][col-1]>0)
                  dp[row][col]=std::max(dp[row][col],dp[row][col-1]+1);
               if(row+1<m && grid[row][col]>grid[row+1][col-1] && dp[row+1][col-1]>0)
                  dp[row][col]=std::max(dp[row][col],dp[row+1][col-1]+1);

               ans=std::max(ans,dp[row][col]-1);
            }
         }

         return ans;
      }
};
```

2684. Maximum Number of Moves in a Grid

```
/*
   Bottom up with 1D array
   Time complexity: O(mn)
   Space compexity:O(m)
*/
class Solution {
   public:
      int maxMoves(std::vector<std::vector<int>>& grid){
         int m=grid.size();
         int n=grid[0].size();

         std::vector<int> prev_col(m,1);

         int ans=0;
         for(int col=1;col<n;++col){
            std::vector<int> cur_col(m,0);
            for(int row=0;row<m;++row){
               if(row-1>=0 && grid[row][col]>grid[row-1][col-1] && prev_col[row-1]>0)
                  cur_col[row]=std::max(cur_col[row],prev_col[row-1]+1);

               if(grid[row][col]>grid[row][col-1] && prev_col[row]>0)
                  cur_col[row]=std::max(cur_col[row],prev_col[row]+1);

               if(row+1<m && grid[row][col]>grid[row+1][col-1] && prev_col[row+1]>0)
                  cur_col[row]=std::max(cur_col[row],prev_col[row+1]+1);

               ans=std::max(ans,cur_col[row]-1);
            }
            prev_col=cur_col;
         }

         return ans;
      }
};
```