```
/*
你要开发一座金矿，地质勘测学家已经探明了这座金矿中的资源分布，并用大小为 m * n 的网格 grid 进行了标注。每个单元格中的整数就表示这一单元格中的黄金数量；如果该单元格是空的，那么就是 0。

为了使收益最大化，矿工需要按以下规则来开采黄金：

    每当矿工进入一个单元，就会收集该单元格中的所有黄金。
    矿工每次可以从当前位置向上下左右四个方向走。
    每个单元格只能被开采（进入）一次。
    不得开采（进入）黄金数目为 0 的单元格。
    矿工可以从网格中 任意一个 有黄金的单元格出发或者是停止。



示例 1：

输入：grid = [[0,6,0],[5,8,7],[0,9,0]]
输出：24
解释：
[[0,6,0],
 [5,8,7],
 [0,9,0]]
一种收集最多黄金的路线是：9 -> 8 -> 7。

示例 2：

输入：grid = [[1,0,7],[2,0,6],[3,4,5],[0,3,0],[9,0,20]]
输出：28
解释：
[[1,0,7],
 [2,0,6],
 [3,4,5],
 [0,3,0],
 [9,0,20]]
一种收集最多黄金的路线是：1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7。



提示：

    1 <= grid.length, grid[i].length <= 15
    0 <= grid[i][j] <= 100
    最多 25 个单元格中有黄金。

来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/path-with-maximum-gold
著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。
*/
```

分析:

- 回溯算法:遍历每一种可能,保留最大值即可.

方法一:C++_回溯算法

```cpp
class Solution
{

    private:

        void helper(    int&                    max_gold    ,
                        int                     cur_gold    ,
                        vector<vector<int>>&    grid        ,
                        vector<vector<bool>>&   vvb         ,
                        int                     cur_i       ,
                        int                     cur_j       ,
                        int                     R           ,
                        int                     C
                    )
        {
            if(     (   cur_i < 0                       )
                ||  (   cur_i >= R                      )
                ||  (   cur_j < 0                       )
                ||  (   cur_j >= C                      )
                ||  (   grid[cur_i][cur_j] == 0         )
                ||  (   vvb[cur_i][cur_j] == true       )
            )
            {
                if(cur_gold > max_gold)
                {
                    max_gold = cur_gold;
                }

                return;
            }
            vvb[cur_i][cur_j] = true;
            cur_gold += grid[cur_i][cur_j];
            helper(max_gold,cur_gold,grid,vvb,cur_i+1,cur_j,R,C);
            helper(max_gold,cur_gold,grid,vvb,cur_i-1,cur_j,R,C);
            helper(max_gold,cur_gold,grid,vvb,cur_i,cur_j+1,R,C);
            helper(max_gold,cur_gold,grid,vvb,cur_i,cur_j-1,R,C);
            vvb[cur_i][cur_j] = false;
        }



    public:
        int getMaximumGold(vector<vector<int>>& grid)
        {

            if(grid.size() < 1)
            {
                return 0;
            }

            int max_gold = 0;
            int m = grid.size();
            int n = grid[0].size();
```

```
            vector<vector<bool>> vvb = vector<vector<bool>>(m,vector<bool>
(n,false));
            for(int i = 0 ; i < m ; i++)
            {
                for(int j = 0 ; j < n ; j++)
                {
                    helper(max_gold,0,grid,vvb,i,j,m,n);
                }
            }
            return max_gold;
        }
};


/*
执行结果：
通过
显示详情
执行用时 :132 ms, 在所有 cpp 提交中击败了20.06% 的用户
内存消耗 :8.9 MB, 在所有 cpp 提交中击败了100.00%的用户
*/
```

AlimyBreak
2019.12.14