1237. 找出给定方程的正整数解（简单）

1. 题目描述

   给出一个函数 `f(x, y)` 和一个目标结果 `z`，请你计算方程 `f(x,y) == z` 所有可能的正整数 **数对** `x` 和 `y`。

   给定函数是严格单调的，也就是说：

   - `f(x, y) < f(x + 1, y)`
   - `f(x, y) < f(x, y + 1)`

   函数接口定义如下：

   ```
   interface CustomFunction {
   public:
     // Returns positive integer f(x, y) for any given positive integer x and y.
     int f(int x, int y);
   };
   ```

   如果你想自定义测试，你可以输入整数 `function_id` 和一个目标结果 `z` 作为输入，其中 `function_id` 表示一个隐藏函数列表中的一个函数编号，题目只会告诉你列表中的 `2` 个函数。

   你可以将满足条件的 **结果数对** 按任意顺序返回。

   示例 1：

   ```
   输入：function_id = 1, z = 5
   输出：[[1,4],[2,3],[3,2],[4,1]]
   解释：function_id = 1 表示 f(x, y) = x + y
   ```

   示例 2：

   ```
   输入：function_id = 2, z = 5
   输出：[[1,5],[5,1]]
   解释：function_id = 2 表示 f(x, y) = x * y
   ```

   提示：

   - `1 <= function_id <= 9`
   - `1 <= z <= 100`
   - 题目保证 `f(x, y) == z` 的解处于 `1 <= x, y <= 1000` 的范围内。
   - 在 `1 <= x, y <= 1000` 的前提下，题目保证 `f(x, y)` 是一个 32 位有符号整数。

2. 简单实现

   对x = [1, 1000]，对y = [1, 1000]做二分法

   ```
   /*
    * // This is the custom function interface.
    * // You should not implement it, or speculate about its implementation
   ```

```
 * class CustomFunction {
 * public:
 *     // Returns f(x, y) for any given positive integers x and y.
 *     // Note that f(x, y) is increasing with respect to both x and y.
 *     // i.e. f(x, y) < f(x + 1, y), f(x, y) < f(x, y + 1)
 *     int f(int x, int y);
 * };
 */

class Solution {
public:
    vector<vector<int>> findSolution(CustomFunction& customfunction, int z) {
        vector<vector<int>> ans;
        int x = 1;
        while(x <= 1000 && customfunction.f(x, 1000) < z)//跳过f(x, 1000)小于z的那些x
            x++;
        for( ; x <= 1000; x++){
            if(customfunction.f(x, 1) > z)//f(x, 1)大于z，之后所有的都大于z，不用再遍历了
                break;
            int l = 1, r = 1000;
            while(l <= r){
                int mid = l + (r - l) / 2;
                int cur = customfunction.f(x, mid);
                if(cur == z){
                    ans.push_back({x, mid});
                    break;
                }
                else if(cur < z)
                    l = mid + 1;
                else
                    r = mid - 1;
            }
        }
        return ans;
    }
};
```

3. 最简解法

和那个方格上搜索某个值的题解法类似

```
class Solution {
public:
    vector<vector<int>> findSolution(CustomFunction& customfunction, int z) {
        vector<vector<int>> res = {};
        int x =1; int y = 1000;
        while(x<=1000 && y>=1 ){
            if(customfunction.f(x,y) == z) {
                res.push_back({x,y});
                ++x;--y;
            }
            else if(customfunction.f(x,y)<z){
                ++x;
```

```
                } else --y;
            }
            return res;
        }
};
```

1238. 循环码排列（中等）

1. 题目描述

给你两个整数 `n` 和 `start`。你的任务是返回任意 `(0,1,2,,...,2^n-1)` 的排列 `p`，并且满足：

- `p[0] = start`
- `p[i]` 和 `p[i+1]` 的二进制表示形式只有一位不同
- `p[0]` 和 `p[2^n -1]` 的二进制表示形式也只有一位不同

示例 1：

```
输入：n = 2, start = 3
输出：[3,2,0,1]
解释：这个排列的二进制表示是 (11,10,00,01)
     所有的相邻元素都有一位是不同的，另一个有效的排列是 [3,1,0,2]
```

示例 2：

```
输出：n = 3, start = 2
输出：[2,6,7,5,4,0,1,3]
解释：这个排列的二进制表示是 (010,110,111,101,100,000,001,011)
```

提示：

- `1 <= n <= 16`
- `0 <= start < 2^n`

2. 简单实现

以n=3为例，这个过程想象为在立方体的八个顶点上沿边不重复遍历八个顶点，可以发现无论怎么走都是可以的，因此，只要记录所有访问过的数，只要改变p[i]某一位后的数没访问过，就可以直接访问

```
class Solution {
public:
    vector<int> circularPermutation(int n, int start) {
        int size = pow(2, n);//数组长度
        vector<int> ans = vector<int>(size, -1);
        ans[0] = start;
        unordered_set<int> visited;
        visited.insert(start);
        for(int i = 1; i < size; i++){
            for(int j = 0; j < n; j++){//依次尝试改变从右数第0...n位
                int cur = ans[i-1] ^ (1 << j);//将第j位取反
                if(visited.count(cur) <= 0){
                    ans[i] = cur;
                    visited.insert(cur);
```

```
                    break;
                }
            }
        }
        return ans;
    }
};
```

其他解法也可以用格雷码解

1239. 串联字符串的最大长度（中等）

1. 题目描述

给定一个字符串数组 `arr`，字符串 `s` 是将 `arr` 某一子序列字符串连接所得的字符串，如果 `s` 中的每一个字符都只出现过一次，那么它就是一个可行解。

请返回所有可行解 `s` 中最长长度。

示例 1：

```
输入：arr = ["un","iq","ue"]
输出：4
解释：所有可能的串联组合是 "","un","iq","ue","uniq" 和 "ique"，最大长度为 4。
```

示例 2：

```
输入：arr = ["cha","r","act","ers"]
输出：6
解释：可能的解答有 "chaers" 和 "acters"。
```

示例 3：

```
输入：arr = ["abcdefghijklmnopqrstuvwxyz"]
输出：26
```

提示：

- `1 <= arr.length <= 16`
- `1 <= arr[i].length <= 26`
- `arr[i]` 中只含有小写英文字母

2. 简单实现

回溯剪枝，每个字符串可以选择加入或不加入当前可行字符串

```
class Solution {
public:
    int ans;
    //cnt记录目前字符串种包含26个字母中的哪些，s为新加入的字符串，判断能否加入
    bool check(string& s, vector<bool>& cnt){
        for(int i = 0; i < s.size(); i++)
```

```
            if(cnt[s[i] - 'a'] == true)//存在重复字母，不能加入
                return false;
            else
                cnt[s[i] - 'a'] = true;
        return true;//可以加入，且cnt已经对应修改
    }
    //当前遍历到arr[idx]，temp为当前可行字符串长度，cnt为当前可行字符串包含的字母情况
    void search(vector<string>& arr, int idx, int temp, vector<bool>& cnt){
        if(temp == 26 || ans == 26)
            ans = 26;
        else if(idx == arr.size()){
            if(temp > ans)
                ans = temp;
        }
        else{
            search(arr, idx+1, temp, cnt);//不加入
            vector<bool> cnt_back = cnt;
            if(temp + arr[idx].size() <= 26 && check(arr[idx], cnt))//可以加入
                search(arr, idx+1, temp+arr[idx].size(), cnt);//加入
            cnt = cnt_back;
        }
    }
    int maxLength(vector<string>& arr) {
        int len = arr.size();
        if(len == 0) return 0;
        if(len == 1) return arr[0].size();
        ans = 0;
        vector<bool> cnt = vector<bool>(26, false);
        search(arr, 0, 0, cnt);
        return ans;
    }
};
```

1240. 铺瓷砖（困难）

1. 题目描述

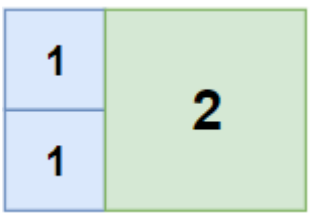
   你是一位施工队的工长，根据设计师的要求准备为一套设计风格独特的房子进行室内装修。

   房子的客厅大小为 `n` x `m`，为保持极简的风格，需要使用尽可能少的 **正方形** 瓷砖来铺盖地面。

   假设正方形瓷砖的规格不限，边长都是整数。

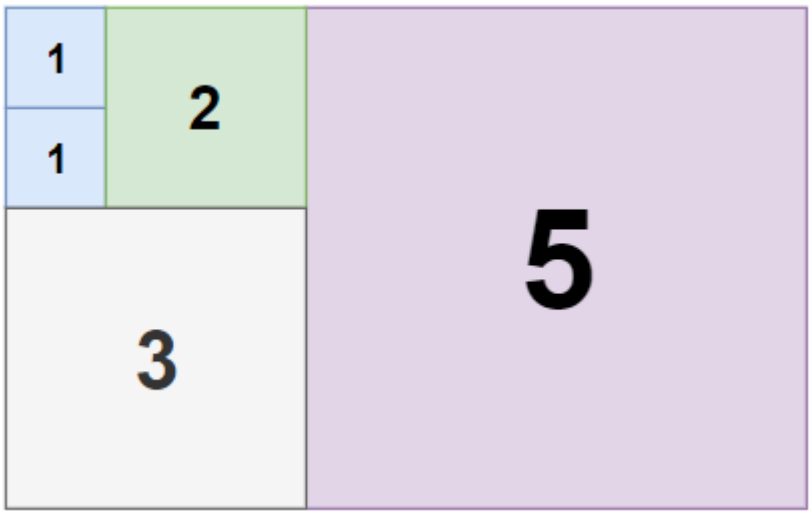
   请你帮设计师计算一下，最少需要用到多少块方形瓷砖？

   示例 1：



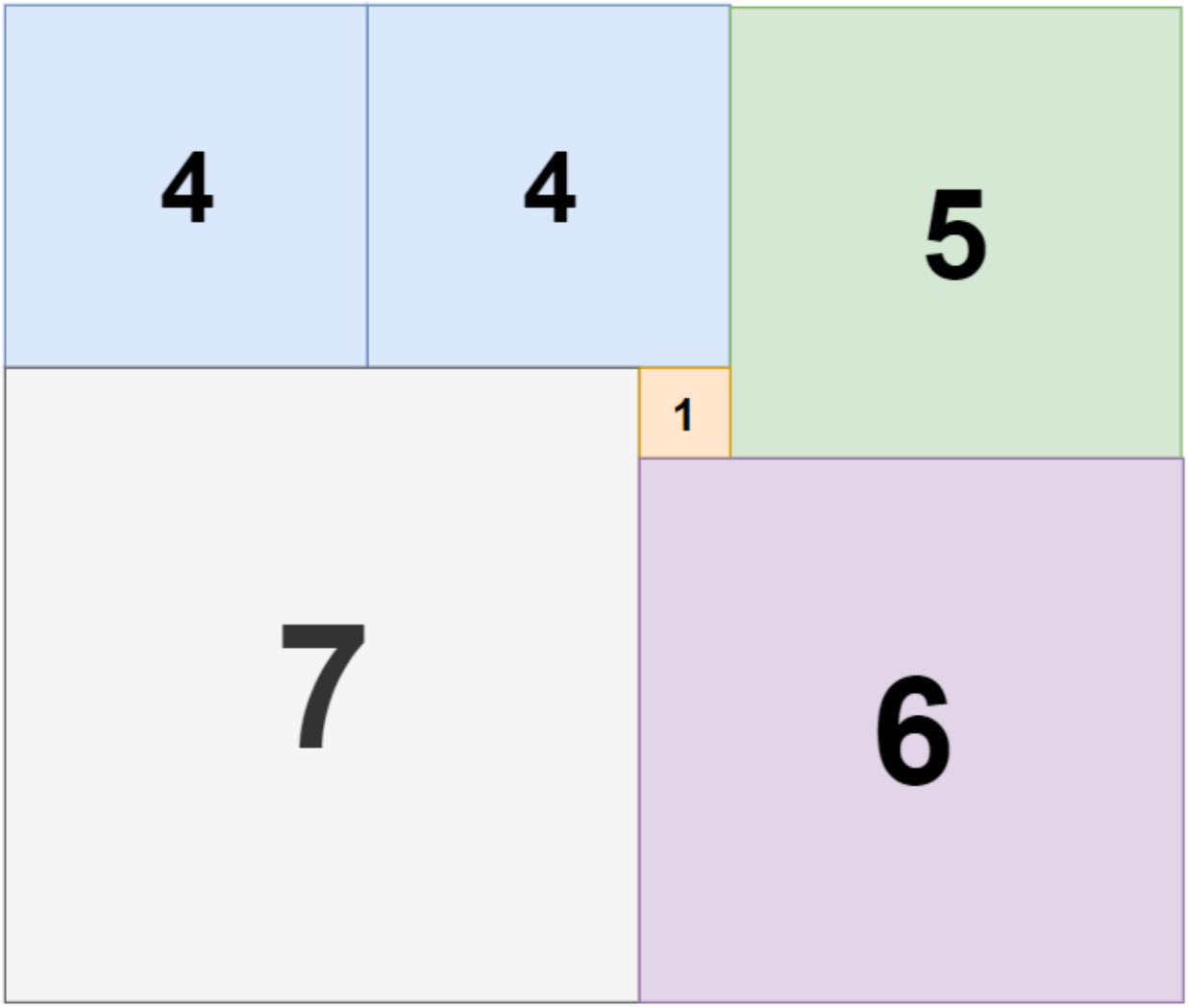
```
输入：n = 2, m = 3
输出：3
解释：3 块地砖就可以铺满卧室。
     2 块 1x1 地砖
     1 块 2x2 地砖
```

示例 2：




```
输入：n = 5, m = 8
输出：5
```

示例 3：



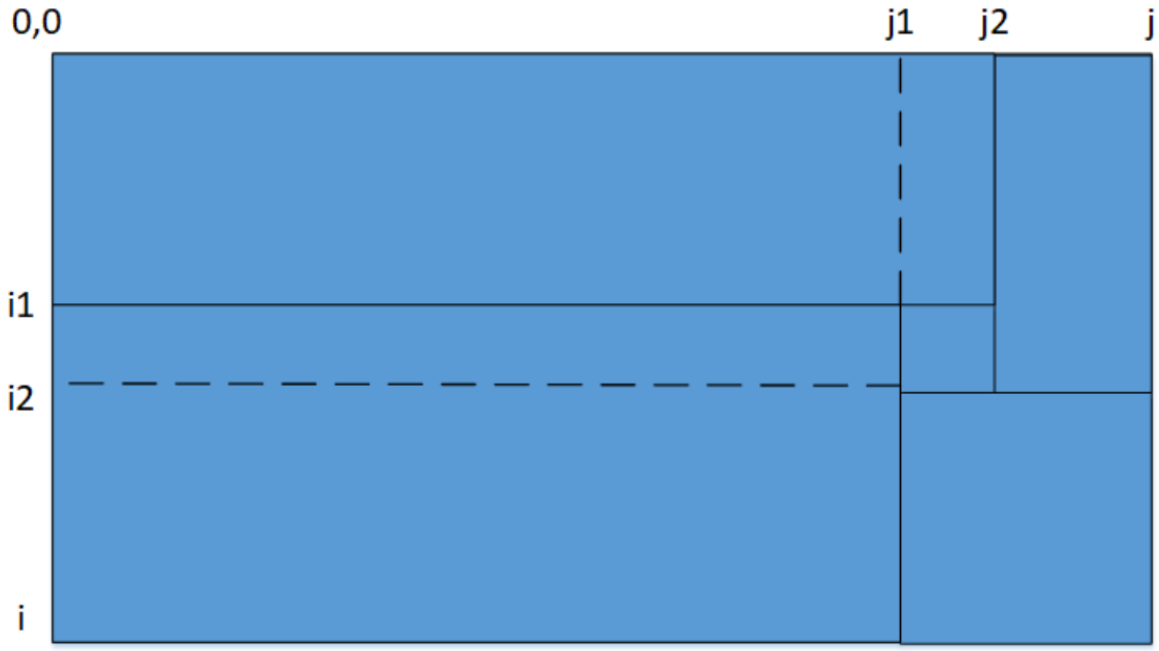

```
输入：n = 11, m = 13
输出：6
```

提示：

- `1 <= n <= 13`
- `1 <= m <= 13`

2. 解法——动态规划

任何一个矩形都可以最多分成五个整矩形（其他情况都可以认为是下面图的一般化），如下图：



现在用f(i,j)表示长i宽j的矩形最少需要多少个正方形瓷砖，考虑在矩形右下角插入边长为k的瓷砖，可以得到以下状态转移：

$$f[i][j] = min\{f[i1][j2] + f[i2][j-j2] + f[i-i1][j1] + f[i2-i1][j2-j1] + 1\}$$

$$1 \le k \le min(i,j)$$

$$i2 = i-k, j1 = j-k$$

$$0 \le i1 \le i2, j1 \le j2 \le j$$

```
class Solution {
public:
    int tilingRectangle(int n, int m) {

        int f[14][14];
        memset(f,1,sizeof(f));
        for(int i=0;i<=n;i++)f[i][0] = 0;
        for(int i=0;i<=m;i++)f[0][i] = 0;
        for(int i=1;i<=n;i++)
            for(int j=1;j<=m;j++)
                for(int k=1;k<=min(i,j);k++){
                    int i2 = i-k;
                    int j1 = j-k;
                    for(int i1=0;i1<=i2;i1++)
                        for(int j2=j1;j2<=j;j2++){
                            f[i][j] = min(f[i][j],
                                          f[i1][j2] + f[i2][j-j2] +
                                          f[i-i1][j1] + f[i2-i1][j2-j1] + 1);
                        }
                }
        return f[n][m];
    }
};
```

PS：该算法貌似仅适用于本题范围，当输入17，16时会出错

3. 正确解法——dfs

```
class Solution {
public:
    void dfs(vector<int>& state, int m, int used, int& res) {
        if (used >= res) return;
        int n = state.size();
        int i;
        for (i = 0; i < n; i++)
            if (state[i] != (1 << m) - 1) break;
        if (i == n) {
            res = min(res, used);
            return;
        }
        int j = 0;
        while ((state[i] & (1 << j))) j++;
        for (int k = m; k >= 1; k--) {
            if (i + k > n) continue;
            if (j + k > m) continue;
            bool suc = true;
            for (int r = i; r < i+k; r++) {
                if ((state[r] >> j) & ((1 << k) - 1) != 0) {
                    suc = false;
                    break;
                }
            }
            if (suc) {
                for (int r = i; r < i+k; r++)
                    state[r] ^= ((1 << k) - 1) << j;
                dfs(state, m, used+1, res);
                for (int r = i; r < i+k; r++)
                    state[r] ^= ((1 << k) - 1) << j;
            }
        }
    }
    int tilingRectangle(int n, int m) {
        int res = INT_MAX;
        vector<int> state(n, 0);
        dfs(state, m, 0, res);
        return res;
    }
};
```