C++方向编程题答案

第一周

day6

> 题目ID：45840-不要二
>
> 链接：https://www.nowcoder.com/practice/1183548cd48446b38da501e58d5944eb?tpId=85&&tqId=29840&rp=1&ru=/activity/oj&qru=/ta/2017test/question-ranking

```
// 直接暴力计算，默认所有蛋糕的位置标记成1,不能放的地方标记成0
// 举个例子，就可以找出规律，是以4为周期重复出现的
// 1 1 0 0 1 1
// 1 1 0 0 1 1
// 0 0 1 1 0 0
// 0 0 1 1 0 0

#include<iostream>
#include<vector>
using namespace std;
int main()
{
    int w,h,res = 0;
    cin >> w >> h;
    vector<vector<int>> a;
    a.resize(w);
    for(auto& e : a)
        e.resize(h, 1);

    for(int i=0;i<w;i++)
    {
        for(int j=0;j<h;j++)
        {
            if(a[i][j]==1)
            {
                res++;
                if((i+2)<w)
                    a[i+2][j] = 0;

                if((j+2)<h)
                    a[i][j+2] = 0;
            }
        }
    }
    cout << res;
    return 0;
}
```

> 58539-地下迷宫

https://www.nowcoder.com/practice/571cfbe764824f03b5c0bfd2eb0a8ddf?tpId=85&&tqId=29860&rp=1&ru=/activity/oj&qru=/ta/2017test/question-ranking

```
#include <iostream>
#include <vector>
using namespace std;

/*
 * 声明：这份代码是照搬@null_ptr的 但是代码没什么注释，乍一看不好理解，
 * 所以我作了比较详细的注释，方便参考
 * 基本思想：典型的迷宫问题，DFS穷举所有路径，找出剩余体力最多的路径
 */

#define VISITED 2


int m, n, P;        // 输入m, n, P
int maze[10][10];   // 迷宫地图
int dir[4][2] = {{0, -1}, {0, 1}, {-1, 0}, {1, 0}};     // 左，右，上，下移动，例如：{0, -1}代表向左移动一步
int cost[4] = {-1, -1, -3, 0};    // 左，右，上，下移动的体力消耗，例如：{0,  -1}对应-1
int final_P = -200;    // 剩余的体力值，初始为较小的数，保证final_P被正确更新

// 存储各点的数据结构
struct mazePoint {
    mazePoint(int _x, int _y): x(_x), y(_y) {}
    int x, y;
};
// 存储每次遍历到的路径
vector<mazePoint> pathStack;
// 存储最终的最优路径
vector<mazePoint> minCostPath;

// 函数：打印路径
void printPath(const vector<mazePoint>& path) {
    for (int i = 0; i < path.size(); ++i) {
        cout << "[" << path[i].x << "," << path[i].y << "]";
        if (i < path.size() - 1) {
            cout << ",";
        }
    }
}

// 函数：寻找最优路径
void search(int x, int y, int cur_P) {
    // 将当前点加入路径并标记为VISITED
    pathStack.push_back(mazePoint(x, y));
    maze[x][y] = VISITED;

    // 如果当前点为出口且当前体力值>=0，则更新final_P与minCostPath，并返回
    if (x == 0 && y == m-1 && cur_P >= 0) {
        if (cur_P > final_P) {
            final_P = cur_P;
```

```
            minCostPath = pathStack;
        }
        pathStack.pop_back();   // 为了回退至之前的节点，将当前结点弹出
        maze[x][y] = 1;    // 注意：之前maze[x][y]被标记为VISITED（值为2），回退后应该将其还原为1
        return;
    }

    // 如果当前点并非出口且当前体力值>=0，则分别向左右上下四个方向探索，并计算相应的消耗
    // 如果新的点再边界内且为可达点，递归调用search函数
    if (cur_P > 0) {
        for (int i = 0; i < 4; ++i) {
            int nx = x + dir[i][0];
            int ny = y + dir[i][1];
            int nP = cur_P + cost[i];
            if (nx >= 0 && nx < n && ny >= 0 && ny < m && maze[nx][ny] == 1)
                search(nx, ny, nP);
        }
    }

    pathStack.pop_back();    // 为了回退至之前的节点，将当前结点弹出
    maze[x][y] = 1;    // 注意：之前maze[x][y]被标记为VISITED（值为2），回退后应该将其还原为1
}

// 主函数
int main()
{
    cin >> n >> m >> P;
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            cin >> maze[i][j];

    search(0, 0, P);

    if (final_P != -200)    // 如果final_P更新过，输出最优路径
        printPath(minCostPath);
    else    // 如果final_P为初始值-200，代表其没有被更新过，也就意味着没有可行路径
        cout << "Can not escape!";

    return 0;
}
```