

C++方向编程题答案

第四周

day24

题目ID: 26020-年终奖

链接: <https://www.nowcoder.com/practice/72a99e28381a407991f2c96d8cb238ab?tpId=49&ttId=29305&rp=1&ru=/activity/oj&ru=/ta/2016test/question-ranking>

【题目解析】:

本题题意明确

【解题思路】:

本题需要使用动态规划求解。

定义 $f(i,j)$ 表示从左上角走到坐标 (i, j) 处能获得的最大奖励。

搜索所有从左上角走到右下角的路径, 找到最优路径。

$f(i,j)$ 分三种情况:

第一列: $f(i, 0) = f(i-1, 0) + \text{board}(i, 0)$

第一行: $f(0, j) = f(0, j-1) + \text{board}(0, j)$

其它位置: $f(i, j) = \max\{f(i-1, j), f(i, j-1)\} + \text{board}(i, j)$

最后返回右下角的值。

【示例代码】

```
class Bonus {
public:
    int getMost(vector<vector<int>> board) {
        int length = board.size();
        int width = board[0].size();
        vector<vector<int>> allPrice;
        for (int i = 0; i < length; i++) {
            vector<int> tmp(width, 0);
            allPrice.push_back(tmp);
        }
        allPrice[0][0] = board[0][0];
        for (int i = 0; i < length; i++) {
            for (int j = 0; j < width; j++) {
                //如果是起点坐标, 不做任何处理。
                if (i == 0 && j == 0)
                    continue;
                //如果走在行的临界边, 也就是第一行, 那么他只能向右走
                //向右走的时候该点就要将后面的值加起来。

                else if (i == 0) {
```

```

        allPrice[i][j] = allPrice[i][j - 1] + board[i][j];
    }
    //如果走在列的临界边，也就是第一列，那么他只能向下走
    //向下走的时候该点就要将上面的值加起来。
    else if (j == 0) {
        allPrice[i][j] = allPrice[i - 1][j] + board[i][j];
    }
    else {
        //除去两个临界边，剩下的就是既能向右走，也能向下走，
        //这时候就要考虑走到当前点的所有可能得情况，也就是走到当前点
        //各自路径的和是不是这些所有到达该点路径当中最大的了。
        allPrice[i][j] = max(allPrice[i][j - 1], allPrice[i - 1][j]) + board[i]
[j];
    }
}
}
// 返回最后一个坐标点的值，它就表示从左上角走到右下角的最大奖励
return allPrice[length - 1][width - 1];
}
};

```

题目ID:36867-迷宫问题

链接: <https://www.nowcoder.com/practice/cf24906056f4488c9ddb132f317e03bc?tpId=37& tqId=21266&rp=1&ru=/activity/oj&qu=/ta/huawei/question-ranking>

【题目解析】：

本题题意明确

【解题思路】：

本题可用回溯法求解 具体步骤为：

1. 首先将当前点加入路径，并设置为已走
2. 判断当前点是否为出口，是则输出路径，保存结果；跳转到4
3. 依次判断当前点的上、下、左、右四个点是否可走，如果可走则递归走该点
4. 当前点推出路径，设置为可走

【示例代码】

```

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

int N, M; //分别代表行和列
vector<vector<int>> maze; //迷宫矩阵
vector<vector<int>> path_temp; //存储当前路径，第一维表示位置
vector<vector<int>> path_best; //存储最佳路径

void MazeTrack(int i, int j)
{
    maze[i][j] = 1; //表示当前节点已走，不可再走
    path_temp.push_back({ i, j }); //将当前节点加入到路径中
}

```

```

if (i == N - 1 && j == M - 1) //判断是否到达终点
if (path_best.empty() || path_temp.size() < path_best.size())
    path_best = path_temp;

if (i - 1 >= 0 && maze[i - 1][j] == 0) //探索向上走是否可行
    MazeTrack(i - 1, j);
if (i + 1 < N && maze[i + 1][j] == 0) //探索向下走是否可行
    MazeTrack(i + 1, j);
if (j - 1 >= 0 && maze[i][j - 1] == 0) //探索向左走是否可行
    MazeTrack(i, j - 1);
if (j + 1 < M && maze[i][j + 1] == 0) //探索向右走是否可行
    MazeTrack(i, j + 1);
maze[i][j] = 0; //恢复现场, 设为未走
path_temp.pop_back();
}

int main()
{
    while (cin >> N >> M)
    {
        maze = vector<vector<int>>(N, vector<int>(M, 0));
        path_temp.clear();
        path_best.clear();
        for (auto &i : maze)
            for (auto &j : i)
                cin >> j;
        MazeTrack(0, 0); //回溯寻找迷宫最短通路
        for (auto i : path_best)
            cout << '(' << i[0] << ',' << i[1] << ')' << endl; //输出通路
    }
    return 0;
}

```