题目：

题目 2：一起消消毒

时间限制:10000ms

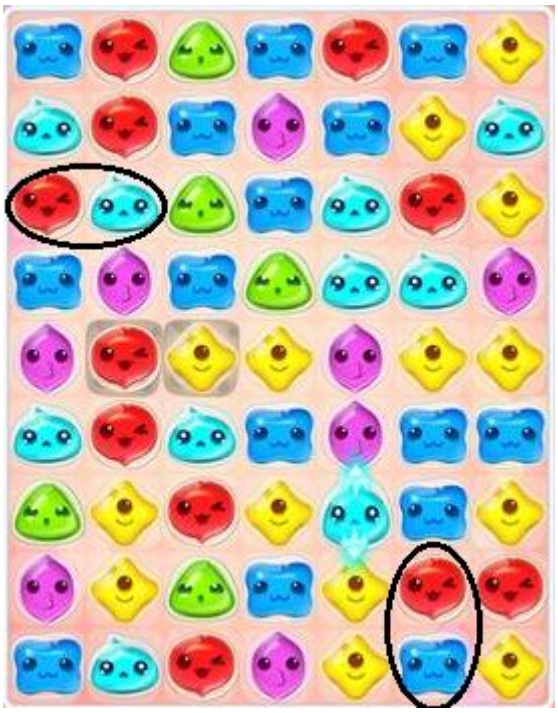
单点时限:1000ms

内存限制:256MB

描述

《一起消消毒》是网易推出的一款创新的消除类游戏。游戏的基本规则很简单，玩家每次操作可以选择把游戏中相邻的两个图形（垂直相邻或者水平相邻）进行交换，如果交换后，出现三个或三个以上的连续相同图形在一条线上，则视这次操作为一次有效的操作，并且这些相同图形会消失。图形消失后，在同一条垂直线上的图形会往下掉。然后继续把同一直线上大于等于三个的相同图形消除，直到已无可消除的图形，则这次操作结束。

如下图所示，假设左上角坐标为(0,0)，右下角坐标为(8,6)，则交换坐标为（2,0）和(3,0)的操作是一个有效的操作，该操作将使得(0,1),(1,1)和(2,1)的红色图形在同一条线上。同理，交换坐标为(7,5)和(8,5)的操作也是一个有效的操作，在同一线上的三个蓝色图形将消失。



示意图

一次操作的完整流程是：

（1）玩家进行交换操作；

（2）得到交换图形的后的局面；

（3）计算在当前局面下所有可以进行消除的图形，注意一次操作有可能使得不止一组图形在同一条线上；

（4）在（3）中计算得到的所有图形同时消失，然后有图形消失了的列，其余剩下的图形往下掉，得到新的局面；

（5）重复执行（3），直到已无图形可以消除。

给出一个局面，同时给出玩家的一次有效的交换操作，请计算出这次操作总共能消除多少个图形。

输入

每个输入数据包含多个测试点。

第一行为测试点的个数 S <= 50。之后是 S 个测试点的数据。

每个测试点的第一行为 N, M (1 <= N, M <= 20)，之后 N 行，每行包含 M 个字符，每个字符代表一个图形，'.'表示该位置为空，'A'...'Z'分别表示一种图案。除此以外局面中无其他字符。之后一行是 4 个整数 x0, y0, x1, y1，表示玩家希望交换的两个相邻图形的坐标位置(x0, y0)以及(x1, y1)。

保证初始的局面为合法局面，无三个或以上连续相同图形在同一线上。并且无悬空的空格，即'.'只会出现在每一列的顶部。

输出

对于每个测试点，对应的结果输出一行，表示进行交换操作后能消除多少个图形。

样例输入

```
2

4 5

.D.M.

.DCAE

.AABA

MDDAM
```

```
2 3 2 4

2 4

F.Z.

ZZAZ

1 2 0 2
```

样例输出

```
8

4
```

1.时间：O（N^2）;空间：O（N^2）

```
class Solution{
    typedef std::pair<int, int> Direction;
public:
    int calcVanishNum(std::vector<std::vector<char>>& board, int x1, int y1, int x2,
int y2){
        if (board.empty() || board.front().empty()) return 0;
```

```
        const int rows = board.size();
        const int cols = board.front().size();
        visited = std::vector<std::vector<int>>(rows, std::vector<int>(cols, 0));
        int result = 0;

        std::swap(board[x1][y1], board[x2][y2]);
        result += check(board, x1, y1);

        for (int i = rows - 1; i >= 0; --i){
            for (int k = 0; k < cols; ++k){
                result += check(board, i, k);
            }
        }

        return result;
    }
private:
    int check(std::vector<std::vector<char>>& board, int x, int y){
        if (board[x][y] == '.') return 0;
        int row_num = check(board, x, y, true);
        int col_num = check(board, x, y, false);
        int result = (row_num >= 3 ? row_num : 0) + (col_num >= 3 ? col_num : 0);
```

```
        if (row_num >= 3 && col_num >= 3) result--;
        if (result >= 3){
            board[x][y] = '.';
            update(board);
        }
        return result;
    }

    int check(std::vector<std::vector<char>>& board, int x, int y, bool isRow){
        if (board[x][y] == '.') return 0;
        int sameNum = 1;
        dfs(board, x, y, sameNum, isRow);
        if (sameNum >= 3){
            vanish(board, x, y, isRow);
        }
        return sameNum;
    }

    void dfs(std::vector<std::vector<char>>& board, int x, int y, int& totalNum,
bool isRow){
        const int rows = board.size();
        const int cols = board.front().size();
```

```
        std::vector<std::pair<int, int>> directions;
        if (isRow == false){
            directions = std::vector<std::pair<int, int>>{{ x + 1, y }, { x - 1, y }};
        }else{
            directions = std::vector<std::pair<int, int>>{{ x, y - 1 }, { x, y + 1 }};
        }
        visited[x][y] = 1;
        for (int i = 0; i < directions.size(); ++i){
            int nextX = directions[i].first, nextY = directions[i].second;
            if (nextX < 0 || nextY < 0 || nextX >= rows || nextY >= cols
                || board[x][y] != board[nextX][nextY] || visited[nextX][nextY] == 1)
                continue;
            totalNum++;
            dfs(board, nextX, nextY, totalNum, isRow);
        }
        visited[x][y] = 0;
    }

    void update(std::vector<std::vector<char>>& board){
        const int rows = board.size();
        const int cols = board.front().size();
        for (int i = 0; i < cols; ++i){
```

```
            int not_dot_index = rows - 1;
            for (int k = rows - 1; k >= 0; --k){
                if (board[k][i] != '.'){
                    if (k != not_dot_index){
                        board[not_dot_index][i] = board[k][i];
                        board[k][i] = '.';
                    }
                    not_dot_index--;
                }
            }
        }
    }

    void vanish(std::vector<std::vector<char>>& board, int x, int y, bool isRow){
        const char cur_ch = board[x][y];
        if (isRow){
            for (int i = y - 1; i >= 0 && board[x][i] == cur_ch; --i){
                board[x][i] = '.';
            }
            for (int i = y + 1; i < board.front().size() && board[x][i] == cur_ch; ++i){
                board[x][i] = '.';
            }
```

```
            } else{
                for (int i = x - 1; i >= 0 && board[i][y] == cur_ch; --i){
                    board[i][y] = '.';
                }
                for (int i = x + 1; i < board.size() && board[i][y] == cur_ch; ++i){
                    board[i][y] = '.';
                }
            }
        }
    private:
        std::vector<std::vector<int>> visited;
};
```