1330：【例8.3】最少步数

题目描述

在各种棋中，棋子的走法总是一定的，如中国象棋中马走“日”。有一位小学生就想如果马能有两种走法将增加其趣味性，因此，他规定马既能按“日”走，也能如象一样走“田”字。他的同桌平时喜欢下围棋，知道这件事后觉得很有趣，就想试一试，在一个（100×100）的围棋盘上任选两点A、B，A点放上黑子，B点放上白子，代表两匹马。棋子可以按“日”字走，也可以按“田”字走，俩人一个走黑马，一个走白马。谁用最少的步数走到左上角坐标为(1,1)的点时，谁获胜。现在他请你帮忙，给你A、B两点的坐标，想知道两个位置到（1,1）点可能的最少步数。

输入格式

A、B两点的坐标。

输出格式

最少步数。

输入样例

```
12 16
18 10
```

输出样例

```
8
9
```

解析

广搜模板，本题给出了两种广搜的写法。

编码

```
#include<bits/stdc++.h>
using namespace std;
struct Node {
    int Row; //行数
    int Col; //列数
    int Step; //到达当前节点时，一共走了多少步
    //有参数的构造函数
    Node(int row, int col, int step) {
```

```
        this->Row = row;
        this->Col = col;
        this->Step = step;
    }
    //有参数的构造函数
    Node(int row, int col) {
        this->Row = row;
        this->Col = col;
    }

    //无参数的构造函数
    Node() {};
};

//访问记录表，所有存在这里的点，都不可以被二次使用
bool Vis[101][101];
//搜索队列，数组版
Node SearchArray[101 * 101];
//搜索队列，队列版
queue<Node> SearchQueue;

int Forward[12][2] = {
        //x描述的纵向变化，y描述的是横向变化
       {1,  -2}, //左下1日
       {2,  -1}, //左下2日
       {2,  1},  //右下2日
       {1,  2},  //右下1日

       {-1, 2},  //右上2日
       {-2, 1},  //右上1日
       {-1, -2},  //右上2日
       {-2, -1},  //右上1日

       {2,  -2}, //左下的田
       {2,  2}, //右下的田
       {-2, 2}, //左上的田
       {-2, -2}, //右上的田
};


//检测节点是否有效，可以被存储
bool CheckNode(int row, int col) {
    //1、是否越界
    //2、是否被访问过
    if (row >= 1 && col >= 1 && row <= 100 && col <= 100) {
        if (!Vis[row][col]) {
            return true;
        }
```

```
    }
    return false;
}

//检测是否为终点
bool CheckEnd(int row, int col, Node end) {
    if (row == end.Row && col == end.Col) {
        return true;
    }
    return false;
}

//广度优先搜索（通过队列）
void BfsByQueue(Node start, Node end) {
    memset(Vis, false, sizeof(Vis));
    //对列中有值,清空队列
    while (!SearchQueue.empty()) {
        SearchQueue.pop();
    }
    //将起点放到队列
    SearchQueue.push(start);
    //启动搜索循环
    while (!SearchQueue.empty()) {
        //取出第一个点
        Node curNode = SearchQueue.front();
        //从队列中删除
        SearchQueue.pop();
        for (int i = 0; i < 12; ++i) {
            int newRow = curNode.Row + Forward[i][0];
            int newCol = curNode.Col + Forward[i][1];
            if (CheckNode(newRow, newCol)) {
                //将当前节点存入队列
                SearchQueue.push(Node(newRow, newCol, curNode.Step + 1)
                Vis[newRow][newCol] = true;
            }
            if (CheckEnd(newRow, newCol, end)) {
                cout << curNode.Step + 1 << endl;
                return;
            }
        }
    }
}

//广度优先搜索（通过数组）
void BfsByArray(Node start, Node end) {
    int head = 0; //头部索引
    int tail = 0; //尾部索引
    //数组做初始化
    memset(Vis, false, sizeof(Vis));
```

```
    //将起点放到队列
    SearchArray[head] = start;
    //启动搜索循环
    while (head <= tail) {
        //取出第一个点
        Node curNode = SearchArray[head];
        //从队列中删除
        head++;
        for (int i = 0; i < 12; ++i) {
            int newRow = curNode.Row + Forward[i][0];
            int newCol = curNode.Col + Forward[i][1];
            if (CheckNode(newRow, newCol)) {
                //将当前节点存入队列
                SearchArray[++tail] = Node(newRow, newCol, curNode.Step
                //将新的节点标记成为已经访问，其他的节点不可以再使用
                Vis[newRow][newCol] = true;
            }
            if (CheckEnd(newRow, newCol, end)) {
                cout << curNode.Step + 1 << endl;
                return;
            }
        }
    }
}

int main() {
    int row1, col1;
    int row2, col2;
    Node start;
    Node end = Node(1, 1);
    cin >> row1 >> col1;
    start = Node(row1, col1, 0);
    BfsByQueue(start, end);
    cin >> row2 >> col2;
    start = Node(row2, col2, 0);
    BfsByArray(start, end);
    return 0;
}
```