bfs 和 dfs

目录

bfs

bfs 前置知识

- 什么是 bfs
- BFS 全称是 Breadth First Search ，中文名是宽度优先搜索，也叫广度优先搜索。
- 所谓宽度优先。就是每次都尝试访问同一层的节点。 如果同一层都访问完了，再访问下一层。

queue

- 队列（ queue ）是一种具有「先进入队列的元素一定先出队列」性质的表，

- 队列其实就是一种特殊的数组，对其输入和输出进行限制，即先进先出

常见成员函数

- 元素访问
- q.front() 返回队首元素
- q.back() 返回队尾元素
- 修改
- q.push() 在队尾插入元素
- q.pop() 弹出队首元素
- 容量
- q.empty() 队列是否为空
- q.size() 返回队列中元素的数量

pair

- pair<T1,T2> 变量名 ;
- T1,T2 都为变量类型，例如 pair<int,double> a;

- pair 就相当于一个结构体;
- pair<int,double> a;
- struct node{
  - int first;
  - double second;
- }a;

四连通和八连通

- 四连通：对于某个中心像素或单元而言，四连通指的是其上下左右四个方向相邻的像素或单元与它是连通的

```
int dx[4] = { 1,-1,0,0 };
int dy[4] = { 0,0,1,-1 };
```

- 八连通：除了上下左右四个方向外，还包括四个对角线方向相邻的像素或单元

```
int dx[8] = { 1,-1,0,0 ,1,1,-1,-1 };
int dy[8] = { 0,0,1,-1,  1,-1,1,-1 };
```

B3625 迷宫寻路 https://www.luogu.com.cn/problem/B3625

题目描述

复制 Markdown 展开 进入 IDE 模式

机器猫被困在一个矩形迷宫里。

迷宫可以视为一个 $n \times m$ 矩阵，每个位置要么是空地，要么是墙。机器猫只能从一个空地走到其上、下、左、右的空地。

机器猫初始时位于 $(1, 1)$ 的位置，问能否走到 (n, m) 位置。

输入格式

第一行，两个正整数 n, m。

接下来 n 行，输入这个迷宫。每行输入一个长为 m 的字符串，`#` 表示墙，`.` 表示空地。

输出格式

仅一行，一个字符串。如果机器猫能走到 (n, m)，则输出 `Yes`；否则输出 `No`。

输入输出样例

输入 #1 复制

```
3 5
.##.#
.#...
...#.
```

输出 #1 复制

```
Yes
```

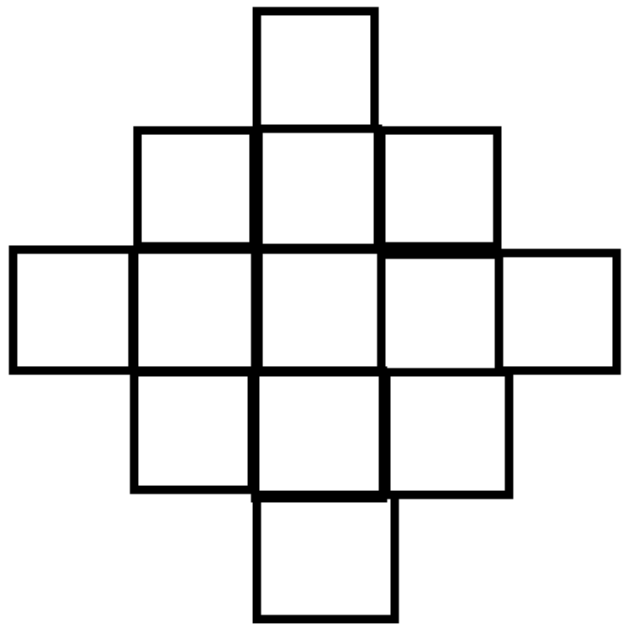
说明/提示

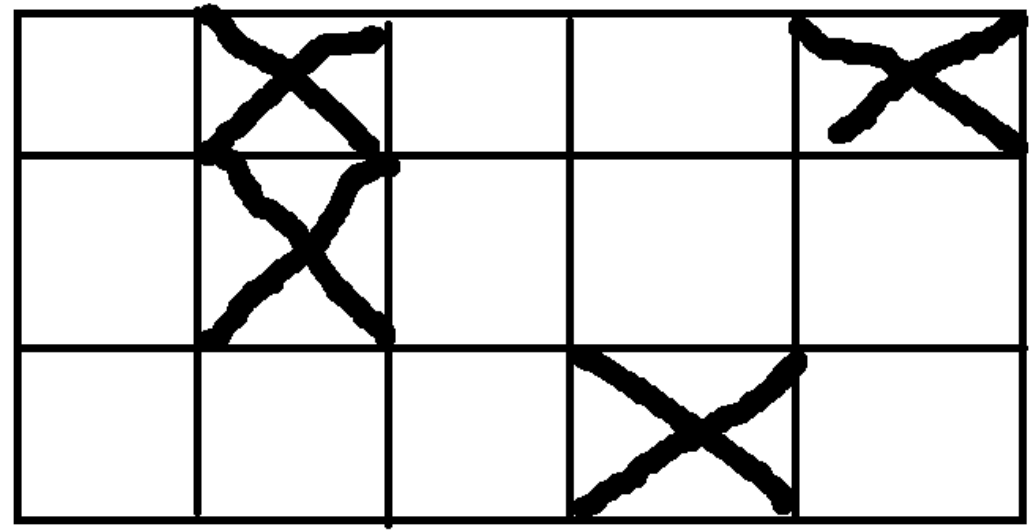
样例解释

路线如下：$(1, 1) \to (2, 1) \to (3, 1) \to (3, 2) \to (3, 3) \to (2, 3) \to (2, 4) \to (2, 5) \to (3, 5)$

数据规模与约定

对于 100% 的数据，保证 $1 \le n, m \le 100$，且 $(1, 1)$ 和 (n, m) 均为空地。

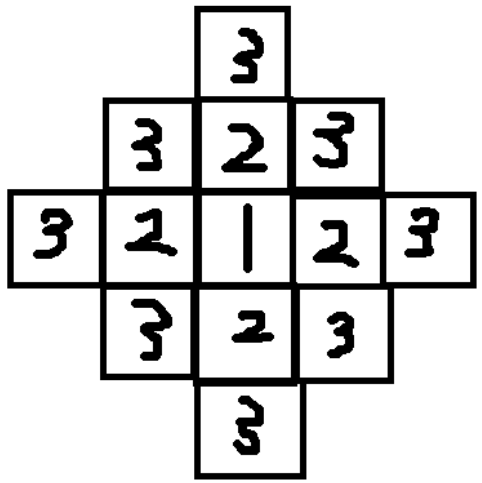


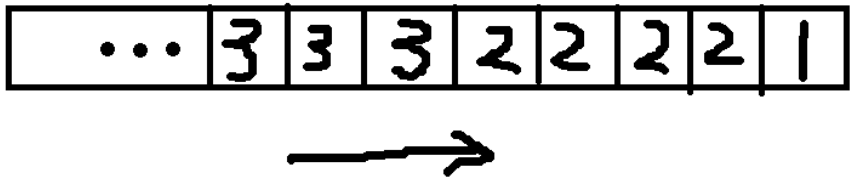


思考

- 众所周知，代码都是一行一行执行的，如何让它们实行这种并行的操作呢

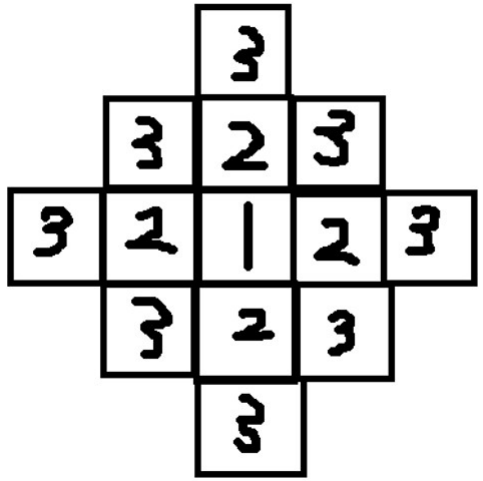
- 显然我们没办法短时间内自己手搓出一种可以并行的计算机语言，所以只能在改并行为串行





vis 数组

记录那些点已经被访问过，避免
左右反复横跳导致死循环



```
queue<pair<int,int>>q;
q.push({ 1,1 });
vis[1][1] = 1;
while (!q.empty()) {
    int x = q.front().first;
    int y = q.front().second;
    q.pop();
    for (int i = 0; i < 4; i++) {
        if (x+dx[i]>=1&&x+dx[i]<=n&&y+dy[i]>=1&&y+dy[i]<=m&&
            vis[x + dx[i]][y + dy[i]] == 0&&a[x+dx[i]][y+dy[i]]=='.') {
            q.push({ x + dx[i],y + dy[i] });
            vis[x + dx[i]][y + dy[i]] = 1;
        }
    }
}
```

模板

- 创建队列
- 更新数据　　// 注意点一
- while (队列不为空) {
- 　　int u = 队首 ;
- 　　弹出队首 ;　　// 注意点二
- 　　for (枚举 u 的邻居) {
- 　　if (满足条件) {　　// 注意点三
- 　　添加到队首 ;
- 　　更新数据 ;　　// 注意点四
- 　　}
- 　　}
- }

bfs 能解决什么样的问题

- 最短路问题
- 连通性问题

P1443 马的遍历 https://www.luogu.com.cn/problem/P1443

题目描述

复制 Markdown 展开 进入 IDE 模式

有一个 $n \times m$ 的棋盘，在某个点 (x, y) 上有一个马，要求你计算出马到达棋盘上任意一个点最少要走几步。

输入格式

输入只有一行四个整数，分别为 n, m, x, y。

输出格式

一个 $n \times m$ 的矩阵，代表马到达某个点最少要走几步（不能到达则输出 -1）。

输入输出样例

输入 #1	复制	输出 #1	复制
3 3 1 1		0 3 2 3 -1 1 2 1 4	

说明/提示

数据规模与约定

对于全部的测试点，保证 $1 \leq x \leq n \leq 400$，$1 \leq y \leq m \leq 400$。

- 创建队列
- 更新数据 //注意点一
- while (队列不为空) {
- int u = 队首;
- 弹出队首; //注意点二
- for (枚举 u 的邻居) {
- if (满足条件) { //注意点三
- 添加到队首;
- 更新数据; //注意点四
- }
- }
- }

```
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++) {
        vis[i][j] = -1;
    }
}
queue<pair<int,int>>q;
q.push({ x,y });
vis[x][y] = 0;
while (!q.empty()) {
    int x = q.front().first;
    int y = q.front().second;
    q.pop();
    for (int i = 0; i < 8; i++) {
        if (x+dx[i]>=1&&x+dx[i]<=n&&y+dy[i]>=1&&y+dy[i]<=m&&vis[x + dx[i]][y + dy[i]] == -1) {
            q.push({ x + dx[i],y + dy[i] });
            vis[x + dx[i]][y + dy[i]] = vis[x][y] + 1;
        }
    }
}
```

拯救 oibh 总部 https://www.luogu.com.cn/problem/P1506

```
• 创建队列
• 更新数据                    //注意点一
• while (队列不为空) {
•     int u = 队首;
•     弹出队首;               //注意点二
•     for (枚举 u 的邻居) {
•         if (满足条件) {     //注意点三
•             添加到队首;
•             更新数据;       //注意点四
•         }
•     }
• }
```

题目描述

oibh 被突来的洪水淹没了，还好 oibh 总部有在某些重要的地方起一些围墙。用 `*` 号表示，而一个四面被围墙围住的区域洪水是进不去的。

oibh 总部内部也有许多重要区域，每个重要区域在图中用一个 `0` 表示。

现在给出 oibh 的围墙建设图，问有多少个没被洪水淹到的重要区域。

输入格式

第一行为两个正整数 x, y。

接下来 x 行，每行 y 个整数，由 `*` 和 `0` 组成，表示 oibh 总部的建设图。

输出格式

输出没被水淹没的 oibh 总部的 `0` 的数量。

输入输出样例

输入 #1 复制

```
4 5
00000
00*00
0*0*0
00*00
```

输出 #1 复制

```
1
```

输入 #2 复制

```
5 5
*****
*0*0*
**0**
*0*0*
*****
```

输出 #2 复制

```
5
```

说明/提示

对于 100% 的数据，$1 \leq x, y \leq 500$。

```cpp
int ans = 0;
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++) {
        if (a[i][j] == '0' && vis[i][j] == 0) {
            int flag = 1;
            int num = 0;
            queue<pair<int,int>>q;
            q.push({ i,j });
            vis[i][j] = 1;
            num++;
            while (!q.empty()) {
                int x = q.front().first;
                int y = q.front().second;
                q.pop();
                for (int k = 0; k < 4; k++) {
                    if (x + dx[k] == 0 || x + dx[k] == n + 1 || y + dy[k] == 0 || y + dy[k] == m + 1) {
                        flag = 0;
                    }
                    else {
                        if (a[x + dx[k]][y + dy[k]] == '0' && vis[x + dx[k]][y + dy[k]] == 0) {
                            q.push({ x + dx[k],y + dy[k] });
                            vis[x + dx[k]][y + dy[k]] = 1;
                            num++;
                        }
                    }
                }
            }
            if (flag == 1)ans += num;
        }
    }
}
cout << ans << endl;
```

P1141 01 迷宫 https://www.luogu.com.cn/problem/P1141

- 创建队列
- 更新数据 //注意点一
- while (队列不为空) {
- int u = 队首;
- 弹出队首; //注意点二
- for (枚举 u 的邻居) {
- if (满足条件) { //注意点三
- 添加到队首;
- 更新数据; //注意点四
- }
- }
- }

题目描述

复制 Markdown 展开 进入 IDE 模式

有一个仅由数字 0 与 1 组成的 $n \times n$ 格迷宫。若你位于一格 0 上，那么你可以移动到相邻 4 格中的某一格 1 上，同样若你位于一格 1 上，那么你可以移动到相邻 4 格中的某一格 0 上。

你的任务是：对于给定的迷宫，询问从某一格开始能移动到多少个格子（包含自身）。

输入格式

第一行为两个正整数 n, m。

下面 n 行，每行 n 个字符，字符只可能是 0 或者 1，字符之间没有空格。

接下来 m 行，每行两个用空格分隔的正整数 i, j，对应了迷宫中第 i 行第 j 列的一个格子，询问从这一格开始能移动到多少格。

输出格式

m 行，对于每个询问输出相应答案。

输入输出样例

输入 #1	复制	输出 #1	复制
2 2 01 10 1 1 2 2		4 4	

说明/提示

对于样例，所有格子互相可达。

- 对于 20% 的数据，$n \leq 10$；
- 对于 40% 的数据，$n \leq 50$；
- 对于 50% 的数据，$m \leq 5$；
- 对于 60% 的数据，$n, m \leq 100$；
- 对于 100% 的数据，$1 \leq n \leq 1000$，$1 \leq m \leq 100000$。

```
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
        if (vis[i][j] == 0) {
            int cnt = 0;
            queue<pair<int, int>>q;
            q.push({ i,j });
            cnt++;
            vis[i][j] = -1;
            while (!q.empty()) {
                int x = q.front().first;
                int y = q.front().second;
                q.pop();
                for (int k = 0; k < 4; k++) {
                    if (x + dx[k] >= 1 && x + dx[k] <= n && y + dy[k] >= 1 && y + dy[k] <= n
                        && vis[x + dx[k]][y + dy[k]] == 0 && (a[x + dx[k]][y + dy[k]] != a[x][y])) {
                        q.push({ x + dx[k],y + dy[k] });
                        cnt++;
                        vis[x + dx[k]][y + dy[k]] = -1;
                    }
                }
            }
            q.push({ i,j });
            vis[i][j] = cnt;
            while (!q.empty()) {
                int x = q.front().first;
                int y = q.front().second;
                q.pop();
                for (int k = 0; k < 4; k++) {
                    if (x + dx[k] >= 1 && x + dx[k] <= n && y + dy[k] >= 1 && y + dy[k] <= n
                        && vis[x + dx[k]][y + dy[k]] != cnt && (a[x + dx[k]][y + dy[k]] != a[x][y])) {
                        q.push({ x + dx[k],y + dy[k] });
                        vis[x + dx[k]][y + dy[k]] = cnt;
                    }
                }
            }
        }
    }
}
```

P2895Meteor Shower S https://www.luogu.com.cn/problem/P2895

题目描述

复制 Markdown　中文　展开　进入 IDE 模式

贝茜听说一场特别的流星雨即将到来：这些流星会撞向地球，并摧毁它们所撞击的任何东西。她为自己的安全感到焦虑，发誓要找到一个安全的地方（一个永远不会被流星摧毁的地方）。

如果将牧场放入一个直角坐标系中，贝茜现在的位置是原点，并且，贝茜不能踏上一块被流星砸过的土地。

根据预报，一共有 M 颗流星 $(1 \leq M \leq 50,000)$ 会坠落在农场上，其中第 i 颗流星会在时刻 T_i（$0 \leq T_i \leq 1000$）砸在坐标为 $(X_i, Y_i)(0 \leq X_i \leq 300, 0 \leq Y_i \leq 300)$ 的格子里。流星的力量会将它所在的格子，以及周围 4 个相邻的格子都化为焦土，当然贝茜也无法再在这些格子上行走。

贝茜在时刻 0 开始行动，她只能在会在横纵坐标 $X, Y \geq 0$ 的区域中，平行于坐标轴行动，每 1 个时刻中，她能移动到相邻的（一般是 4 个）格子中的任意一个，当然目标格子要没有被烧焦才行。如果一个格子在时刻 t 被流星撞击或烧焦，那么贝茜只能在 t 之前的时刻在这个格子里出现。 贝茜一开始在 $(0, 0)$。

请你计算一下，贝茜最少需要多少时间才能到达一个安全的格子。如果不可能到达输出 -1。

输入格式

共 $M + 1$ 行，第 1 行输入一个整数 M，接下来的 M 行每行输入三个整数分别为 X_i, Y_i, T_i。

输出格式

贝茜到达安全地点所需的最短时间，如果不可能，则为 -1。

输入输出样例

输入 #1　复制

```
4
0 0 2
2 1 2
1 1 2
0 3 5
```

输出 #1　复制

```
5
```

```
• 创建队列
• 更新数据                    //注意点一
• while (队列不为空) {
•     int u = 队首;
•     弹出队首;               //注意点二
•     for (枚举 u 的邻居) {
•         if (满足条件) {     //注意点三
•             添加到队首;
•             更新数据;       //注意点四
•         }
•     }
• }
```

```
for (int i = 0; i <= 305; i++) {
    for (int j = 0; j <= 305; j++) {
        a[i][j] = 1e9;
        vis[i][j] = -1;
    }
}
for (int i = 1; i <= m; i++) {
    int x, y, t;
    cin >> x >> y >> t;
    a[x][y] = min(a[x][y], t);
    for (int j = 0; j < 4; j++) {
        if (x + dx[j] >= 0 && y + dy[j] >= 0) {
            a[x + dx[j]][y + dy[j]] = min(a[x + dx[j]][y + dy[j]], t);
        }
    }
}
```

```
queue<pair<int, int>>q;
q.push({ 0,0 });
vis[0][0] = 0;
while (!q.empty()) {
    int x = q.front().first;
    int y = q.front().second;
    q.pop();
    for (int i = 0; i < 4; i++) {
        if (x + dx[i] >= 0 && y + dy[i] >= 0 && x + dx[i] <= 305 && y + dy[i] <= 305
            && vis[x + dx[i]][y + dy[i]] == -1 && vis[x][y] + 1 < a[x + dx[i]][y + dy[i]]) {
            q.push({ x + dx[i],y + dy[i] });
            vis[x + dx[i]][y + dy[i]] = vis[x][y] + 1;
        }
    }
}
```

dfs

dfs 前置知识

- 什么是 dfs
- DFS 全称是 Depth First Search ，中文名是深度优先搜索，是一种用于遍历或搜索树或图的算法。所谓深度优先，就是说每次都尝试向更深的节点走。

递归 + 回溯

- 回溯：递归前保存，递归后复原

全排列问题 https://www.luogu.com.cn/problem/P1706

题目描述

复制 Markdown　展开　进入 IDE 模式

按照字典序输出自然数 1 到 n 所有不重复的排列，即 n 的全排列，要求所产生的任一数字序列中不允许出现重复的数字。

输入格式

一个整数 n。

输出格式

由 $1 \sim n$ 组成的所有不重复的数字序列，每行一个序列。

每个数字保留 5 个场宽。

输入输出样例

输入 #1　复制

```
3
```

输出 #1　复制

```
    1    2    3
    1    3    2
    2    1    3
    2    3    1
    3    1    2
    3    2    1
```

说明/提示

$1 \leq n \leq 9$。

```cpp
int n;
int ans[10];
int vis[10];
void dfs(int x) {
    if (x == n) {
        for (int i = 1; i <= n; i++) {
            cout << "    " << ans[i];
        }
        cout << endl;
        return;
    }
    for (int i = 1; i <= n; i++) {
        if (vis[i] == 0) {
            vis[i] = 1;
            ans[x + 1] = i;
            dfs(x + 1);
            vis[i] = 0;
        }
    }
}
signed main() {
std:ios::sync_with_stdio(0);
    std::cin.tie(0);
    cin >> n;
    dfs(0);
    return 0;
}
```

模板

```
void dfs(int k) {
    if ( 到达目的 ) {
        输出解 ;
        return;                    // 注意点一
    }
    for (int i = 1; i <=  算符种数 ; i++) {
        if ( 满足条件 ) {     // 注意点二
            保存结果 ;                    // 注意点三
            dfs(k + 1);
            恢复 ;       // 注意点四
        }
    }
}
```

dfs 能解决什么样的问题

- 路径搜索问题
- 连通性分析

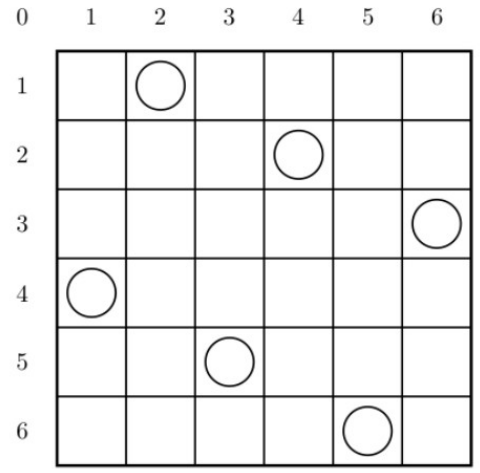
八皇后 https://www.luogu.com.cn/problem/P1219

```
void dfs(int k) {
    if (到达目的) {
        输出解;
        return;                    //注意点一
    }
    for (int i = 1; i <= 算符种数; i++) {
        if (满足条件) {            //注意点二
            保存结果;              //注意点三
            dfs(k + 1);
            恢复;          //注意点四
        }
    }
}
```

题目描述

复制 Markdown　展开　进入 IDE 模式

一个如下的 6×6 的跳棋棋盘，有六个棋子被放置在棋盘上，使得每行、每列有且只有一个，每条对角线（包括两条主对角线的所有平行线）上至多有一个棋子。



洛谷

上面的布局可以用序列 $2\ 4\ 6\ 1\ 3\ 5$ 来描述，第 i 个数字表示在第 i 行的相应位置有一个棋子，如下：

行号 $1\ 2\ 3\ 4\ 5\ 6$

列号 $2\ 4\ 6\ 1\ 3\ 5$

这只是棋子放置的一个解。请编一个程序找出所有棋子放置的解。
并把它们以上面的序列方法输出，解按字典顺序排列。
请输出前 3 个解。最后一行是解的总个数。

输入格式

一行一个正整数 n，表示棋盘是 $n \times n$ 大小的。

输出格式

前三行为前三个解，每个解的两个数字之间用一个空格隔开。第四行只有一个数字，表示解的总数。

输入输出样例

输入 #1 复制	输出 #1 复制
6	2 4 6 1 3 5 3 6 2 5 1 4 4 1 5 2 6 3 4

说明/提示

【数据范围】
对于 100% 的数据，$6 \leq n \leq 13$。

```cpp
void dfs(int x) {
    if (x == n) {
        num++;
        if (num <= 3) {
            for (int i = 1; i <= n; i++) {
                cout << ans[i] << ' ';
            }
            cout << endl;
        }
        return;
    }
    for (int i = 1; i <= n; i++) {
        if (vis[x][i] == 0) {
            for (int j = 1; j <= n; j++) {
                vis[j][i]++;
            }
            for (int j = 1; j <= n; j++) {
                int y = j - x + i;
                if (y >= 1 && y <= n)vis[j][y]++;
            }
            for (int j = 1; j <= n; j++) {
                int y = x + i - j;
                if (y >= 1 && y <= n)vis[j][y]++;
            }
            ans[x + 1] = i;
            dfs(x + 1);
            for (int j = 1; j <= n; j++) {
                vis[j][i]--;
            }
            for (int j = 1; j <= n; j++) {
                int y = j - x + i;
                if (y >= 1 && y <= n)vis[j][y]--;
            }
            for (int j = 1; j <= n; j++) {
                int y = x + i - j;
                if (y >= 1 && y <= n)vis[j][y]--;
            }
        }
    }
}
```

P1123 取数游戏 https://www.luogu.com.cn/problem/P1123

题目描述

复制 Markdown 展开 进入 IDE 模式

一个 $N \times M$ 的由非负整数构成的数字矩阵，你需要在其中取出若干个数字，使得取出的任意两个数字不相邻（若一个数字在另外一个数字相邻 8 个格子中的一个即认为这两个数字相邻），求取出数字和最大是多少。

输入格式

第一行有一个正整数 T，表示了有 T 组数据。

对于每一组数据，第一行有两个正整数 N 和 M，表示了数字矩阵为 N 行 M 列。

接下来 N 行，每行 M 个非负整数，描述了这个数字矩阵。

输出格式

共 T 行，每行一个非负整数，输出所求得的答案。

输入输出样例

输入 #1 复制

```
3
4 4
67 75 63 10
29 29 92 14
21 68 71 56
8 67 91 25
2 3
87 70 85
10 3 17
3 3
1 1 1
1 99 1
1 1 1
```

输出 #1 复制

```
271
172
99
```

说明/提示

样例解释

对于第一组数据，取数方式如下：

$$\begin{matrix} [67] & 75 & 63 & 10 \\ 29 & 29 & [92] & 14 \\ [21] & 68 & 71 & 56 \\ 8 & 67 & [91] & 25 \end{matrix}$$

数据范围及约定

- 对于20%的数据，$1 \le N, M \le 3$；
- 对于40%的数据，$1 \le N, M \le 4$；
- 对于60%的数据，$1 \le N, M \le 5$；
- 对于100%的数据，$1 \le N, M \le 6$，$1 \le T \le 20$，$a_{i,j} \le 10^5$。

```cpp
void dfs(int x, int y, int sum) {
    if (x == n + 1 && y == 1) {
        ans = max(ans, sum);
        return;
    }
    int nextx, nexty;
    if (y == m) {
        nextx = x + 1;
        nexty = 1;
    }
    else {
        nextx = x;
        nexty = y + 1;
    }
    if (vis[x][y] == 0) {
        vis[x][y]++;
        for (int i = 0; i < 8; i++) {
            vis[x+dx[i]][y+dy[i]]++;
        }
        dfs(nextx, nexty, sum + a[x][y]);
        vis[x][y]--;
        for (int i = 0; i < 8; i++) {
            vis[x + dx[i]][y + dy[i]]--;
        }
        dfs(nextx, nexty, sum);
    }
    else {
        dfs(nextx, nexty, sum);
    }
}
```

P1135 奇怪的电梯 https://www.luogu.com.cn/problem/P1135

题目描述

复制 Markdown 中文 展开 进入 IDE 模式

为了对抗附近恶意国家的威胁，R 国更新了他们的导弹防御系统。

一套防御系统的导弹拦截高度要么一直严格单调上升要么一直严格单调下降。

例如，一套系统先后拦截了高度为 3 和高度为 4 的两发导弹，那么接下来该系统就只能拦截高度大于 4 的导弹。

给定即将袭来的一系列导弹的高度，请你求出至少需要多少套防御系统，就可以将它们全部击落。

输入格式

输入包含多组测试用例。

对于每个测试用例，第一行包含整数 n，表示来袭导弹数量。

第二行包含 n 个不同的整数，表示每个导弹的高度。

当输入测试用例 $n = 0$ 时，表示输入终止，且该用例无需处理。

输出格式

对于每个测试用例，输出一行，一个整数，表示所需的防御系统数量。

输入输出样例

输入 #1	复制	输出 #1	复制
5 3 5 2 4 1 0		2	

说明/提示

样例解释

对于样例，需要两套系统。一套击落 $3, 4$ 号导弹，另一套击落 $5, 2, 1$ 号导弹。

数据规模与约定

$1 \leq n \leq 50$。

- 1、可以用某个上升拦截系统拦截。
- 2、可以用某个下降拦截系统拦截。
- 3、需要新建一个上升拦截系统拦截。
- 4、需要新建一个下降拦截系统拦截。

```
void dfs(int x) {
    if (up.size() + down.size() >= ans)return;
    if (x == n + 1) {
        ans = min(ans, (int)(up.size() + down.size()));
        return;
    }
    int flag = -1;
    for (int i = 0; i < up.size(); i++) {
        if (a[x] > up[i]) {
            if (flag == -1) {
                flag = i;
            }
            else {
                if (up[i] > up[flag])flag = i;
            }
        }
    }
    if (flag == -1) {
        up.push_back(a[x]);
        dfs(x + 1);
        up.pop_back();
    }
    else {
        int tem = up[flag];
        up[flag] = a[x];
        dfs(x + 1);
        up[flag] = tem;
    }
    flag = -1;
    for (int i = 0; i < down.size(); i++) {
        if (a[x] < down[i]) {
            if (flag == -1) {
                flag = i;
            }
            else {
                if (down[i] < down[flag])flag = i;
            }
        }
    }
    if (flag == -1) {
        down.push_back(a[x]);
        dfs(x + 1);
        down.pop_back();
    }
    else {
        int tem = down[flag];
        down[flag] = a[x];
        dfs(x + 1);
        down[flag] = tem;
    }
}
```