信息学奥赛笔记02

关于排列组合问题的去重问题

当题目变成了对自定义数组排列组合，而不是对1-n这些元素排列组合的时候，会出现一个问题——假如数组的值是`(1, 1, 2)`那么通过先前的排列组合代码写出的结果会导致计算机认为第一个1和第二个1不是同一个数，会产生相同的结果。

那么该如何避免这个问题呢?

之所以会产生重复的答案，是因为我们在第一个数填了1后，此时回到填写第一个数的时候，我们填写了第二个1，这样之后走出的路径结果是完全一样的，就产生了重复的答案。

我们再细化一下这个问题

现在我们的数组值是`[1, 1, 2]`

对于dfs来说，它会先尝试把第一位填写第一个`1`，那么数组的使用情况如下[**1**, 1, 2],

那么接下来会生成`[1, 1]`，`[1, 2]`这两个答案

那么这一次的dfs就算走到了尽头。

接下来回到选择第一个数的部分，我们在第一个数选择了数组里的第二个`1`，此时数组的被选择情况为[1, **1**, 2]。

那么选择结果也会是`[1, 1]`，`[1, 2]`这两个答案。就产生了重复结果。

所以想要根本的去处这个情况，我们可以对数组**排序**，这样相同的数就被放在了一起。

那如果当前元素 == 上一个元素 && 上一个元素没有被选中装填。是不是就说明，已经完成了优先选择上一个元素的情况，而对于当前这个数来说，已经不再是第一次选择了。所以就要continue掉。用代码表示的话应该是:

设vis数组表示每个数是否被选用，用a数组表示题目中的数组。

```
if (vis[i] || (i >= 0 && a[i] == a[i - 1] && !vis[i - 1])) continue;
```

[U410632] 排列组合问题——全排列(H)https://www.luogu.com.cn/problem/U410632

题目描述

输入一个长度为n的可能含有重复数字的数组，求这个数组元素的全排列。

输入格式

第一行，一个正整数n。

第二行，n个整数，表示数组的值。

输出格式

若干行，一行输出一个排列，用空格隔开。

**请按照字典序输出！！

样例 #1

样例输入 #1

```
3
1 1 2
```

样例输出 #1

```
1 1 2
1 2 1
2 1 1
```

样例 #2

样例输入 #2

```
3
1 2 3
```

样例输出 #2

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

提示

对于100%的数据，有$1 \leq n \leq 10$，$1 \leq a_i \leq n$。

```cpp
#include <bits/stdc++.h>
using namespace std;
int n, nums[11], ans[11], vis[11];
void dfs(int idx) {
    if (idx == n) {
        for (int i = 0; i < n; i++) {
            cout << ans[i] << " ";
        }
        cout << endl;
        return;
    }
    for (int i = 0; i < n; i++) {
```

```
        if (vis[i] || (i && nums[i] == nums[i - 1] && !vis[i - 1]))
continue;  //去重！！！
        vis[i] = 1;
        ans[idx] = nums[i];
        dfs(idx + 1);
        vis[i] = 0;
    }
}
int main() {
    cin >> n;
    for (int i = 0; i < n; i++) cin >> nums[i];
    sort(nums, nums + n);
    dfs(0);
    return 0;
}
```

[U410639]排列组合问题——全组合(H)https://www.luogu.com.cn/problem/U410639

题目描述

输入两个正整数n，k，输入n个可能含有重复元素的数组，请输出这个数组中k个元素的组合情况。

输入格式

第一行，两个正整数n, k。

第二行，n个整数，表示数组的值。

输出格式

若干行，一行输出一个排列，用空格隔开。

请按照字典序输出！！

样例 #1

样例输入 #1

```
4 2
1 1 2 2
```

样例输出 #1

```
1 1
1 2
2 2
```

提示

对于100%的数据，有$1 \leq n, k \leq 30, 1 \leq a_i \leq n$。

> 只在基础的组合问题上进行了改动，考虑到去重问题。

```cpp
#include <bits/stdc++.h>

using namespace std;

int n, k, a[40], s[40];
bool v[40];

void dfs(int l, int last) {
    if(l == k+1) {
        for(int i = 1; i <= k; i++) {
            cout << a[i] << ' ';
        }
        cout << endl;
        return ;
    }
    for(int i = last; i <= n; i++) {
        if(v[i] || i > 1 && s[i] == s[i - 1] && !v[i - 1]) continue;
        v[i] = 1;
        a[l] = s[i];
        dfs(l+1, i+1);
        v[i] = 0;
    }
}

int main() {
    cin >> n >> k;
    for(int i = 1; i <= n; i++) cin >> s[i];
    sort(s + 1, s + n + 1);
    dfs(1, 1);
    return 0;
}
```

二维迷宫类DFS

我们把问题从线性的搜索拓展成在一个面上搜索，那这就涉及到一个问题——从当前这一层到下一层的途径，那么对于一个二维平面来说，我们最基本的行为方式是`上下左右`移动，此时需要引入`方向数组`的概念。

方向数组(4向移动)

这一层到达下一层的途径的方式，它描绘了横坐标，纵坐标的变化趋势，以体现出坐标的变化量，用该变化量去推测移动点。

接下来我们来列举几个比较基础的方向数组写法：

双一维数组形

```
//当前坐标为x, y
int dx[] = {1, -1, 0, 0};
int dy[] = {0, 0, 1, -1};
for (int d = 0; d < 4; d++) {
    int nx = x + dx[d], ny = y + dy[d];
}
```

单二维数组形

```
//当前坐标为x, y
int dir[4][2] = {{-1, 0}, {1, 0}, {0, 1}, {0, -1}};
for (int d = 0; d < 4; d++) {
    int nx = x + dir[d][0], ny = y + dir[d][1];
}
```

压缩单一维数组形

关于这种方法，我们可以看一下相邻的两个位置的变化状态。

`dir[0], dir[1]` 表示的坐标变化是：(0, 1)

`dir[1], dir[2]` 表示的坐标变化是：(1, 0)

`dir[2], dir[3]` 表示的坐标变化是：(0, -1)

`dir[3], dir[4]` 表示的坐标变化是：(-1, 0)

这样也能够描绘出4向移动的方向变化情况。

```
//当前坐标为x, y
int dir[] = {0, 1, 0, -1, 0};
for (int d = 0; d < 4; d++) {
    int nx = x + dir[d], ny = y + dir[d + 1];
}

```

方向数组(8方移动)

```
//当前坐标为x, y
int dx[] = {1, -1, 0, 0, -1, -1, 1, 1};
int dy[] = {0, 0, 1, -1, -1, 1, -1, 1};
for (int d = 0; d < 8; d++) {
    int nx = x + dx[d], ny = y + dy[d];
}
```

这样写8向数组的好处是什么呢？如果我们只需要四向移动，我们可以遍历方向数组的0-3。

如果我们只需要斜向移动，我们可以只遍历数组的(4, 7)

如果我们既需要十字移动，也需要斜向移动，我们就可以遍历方向数组的(0, 7)。

[U412188] 岛屿数量 https://www.luogu.com.cn/problem/U412188

题目描述

给定一个$[0,1]$矩阵，其中`0`代表水域，`1`代表陆地，求这块地岛屿的数量。

注意：一个陆地四个方向上如果仍然有陆地，则表示这块陆地是连续的区域。

此外，你可以假设该网格的四条边均被水域包围。

输入格式

第一行两个正整数n,m，表示这块地区的面积。

接下来n行m列，表示这块地区的水域陆地情况。

输出格式

一个整数，表示岛屿的数量。

样例 #1

样例输入 #1

```
3 3
1 0 0
0 1 1
0 1 1
```

样例输出 #1

```
2
```

提示

对于100%的数据，有$1 \leq n,m \leq 300$。$a[i][j]$的值为`0`或`1`。

> 这道题目，我们做一个dfs函数，把附近相同的1全部消除，那么我们进行了几次消除，就意味着一共有多少块岛屿。

```cpp
#include <bits/stdc++.h>
using namespace std;

int m, n, ans;
int dx[] = {1, -1, 0, 0};
int dy[] = {0, 0, 1, -1};
vector<vector<int> > a;
void dfs(int x, int y) {
```

```
    if (x < 0 || x == m || y < 0 || y == n || a[x][y] == 0) {      //一旦发现当前的格子不符合要求，则跳过
        return;
    }
    a[x][y] = 0;                        //把当前这个岛屿的情况化为0
    for (int d = 0; d < 4; d++) {
        int nx = x + dx[d];
        int ny = y + dy[d];
        dfs(nx, ny);
    }
}
int main() {
    cin >> m >> n;
    a = vector<vector<int> > (m, vector<int>(n));
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            cin >> a[i][j];
        }
    }

    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            if (a[i][j] == 0) continue;
            dfs(i, j);
            ans++;
        }
    }
    cout<<ans;
    return 0;
}
```

岛屿的最大面积

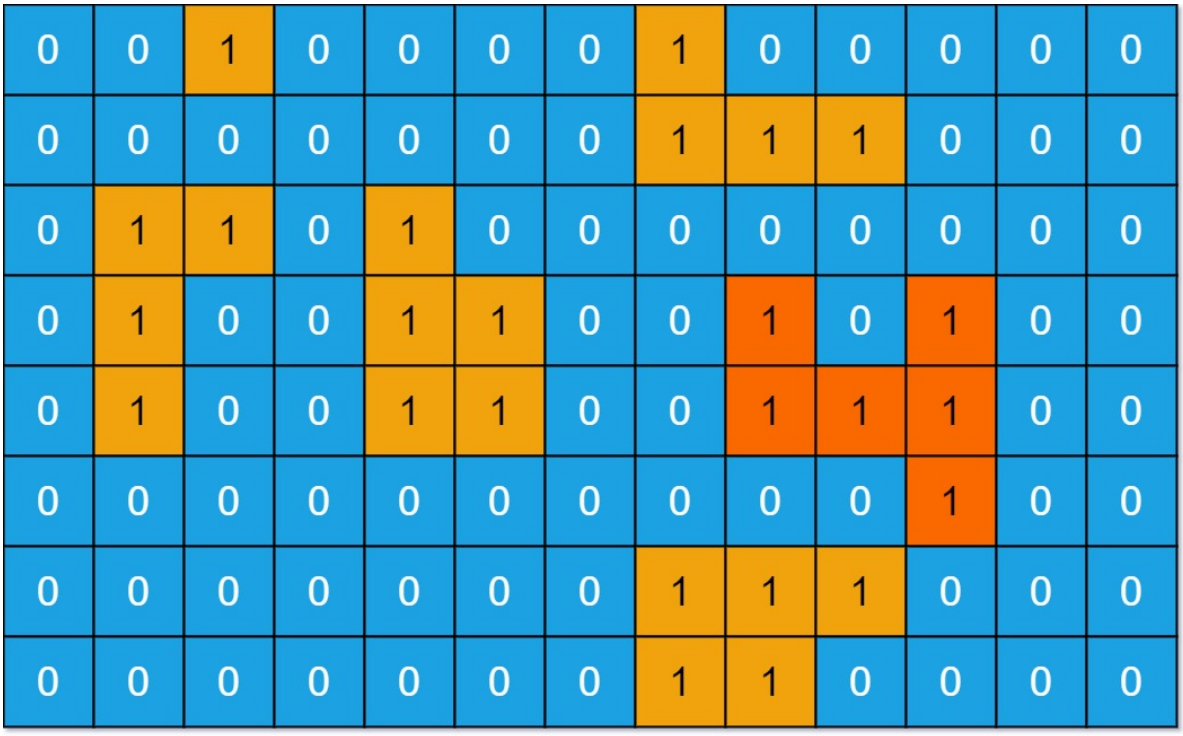
题目描述

给你一个大小为 `m x n` 的二进制矩阵 `a` 。

岛屿 是由一些相邻的 `1` (代表土地) 构成的组合，这里的「相邻」要求两个 `1` 必须在 **水平或者竖直的四个方向上** 相邻。你可以假设 `a` 的四个边缘都被 `0` （代表水）包围着。

岛屿的面积是岛上值为 `1` 的单元格的数目。

计算并返回 `grid` 中最大的岛屿面积。如果没有岛屿，则返回面积为 `0` 。



输入格式

第一行两个正整数m, n，表示这块地区的面积。

接下来m行n列，表示这块地区的水域陆地情况。

输出格式

一个正整数，表示岛屿的最大面积。

样例 #1

样例输入 #1

```
8 13
0 0 1 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 0 0 0
0 1 1 0 1 0 0 0 0 0 0 0 0
0 1 0 0 1 1 0 0 1 0 1 0 0
0 1 0 0 1 1 0 0 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1 1 1 0 0 0
0 0 0 0 0 0 0 1 1 0 0 0 0
```

样例输出 #1

```
6
```

提示

对于100%的数据，有$1 \leq n, m \leq 300$。$a[i][j]$的值为`0`或`1`。

思路分析

> 和上一道题不同的是，这道题我们并不关心进入了多少次岛屿，而是具体一个岛屿的面积，所以我们可以在进入岛屿前将计数器置零，每次修改岛屿的时候，累加计数器，这样当我们消除完一块岛屿后就可以成功获得它的面积，再与答案比较即可。

```cpp
#include <bits/stdc++.h>
using namespace std;

int m, n, ans, cnt;
int dx[] = {1, -1, 0, 0};
int dy[] = {0, 0, 1, -1};
vector<vector<int> > a;
void dfs(int x, int y) {
    if (x < 0 || x == m || y < 0 || y == n || a[x][y] == 0) {
        return;
    }
    a[x][y] = 0;
    cnt++;
    for (int d = 0; d < 4; d++) {
        int nx = x + dx[d];
        int ny = y + dy[d];
        dfs(nx, ny);
    }
}
int main() {
    cin >> m >> n;
    a = vector<vector<int> > (m, vector<int>(n));
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            cin >> a[i][j];
        }
    }

    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            if (a[i][j] == 0) continue;
            cnt = 0;
            dfs(i, j);
            ans = max(ans, cnt);
        }
    }
    cout << ans;
    return 0;
}
```

[U412192] 消除包围的X https://www.luogu.com.cn/problem/U412192

题目描述

给你一个 `m x n` 的矩阵 `a` ，由若干字符 `'X'` 和 `'O'` ，找到所有被 `'X'` 围绕的区域，并将这些区域里所有的 `'O'` 用 `'X'` 填充。

将修改后的矩阵输出。

输入格式

第一行两个正整数m, n，表示这块地区的面积。

接下来m行n列，表示这块矩阵。

输出格式

m行n列，表示修改后的矩阵。

样例 #1

样例输入 #1

```
4 4
XXXX
XOOX
XXOX
XOXX
```

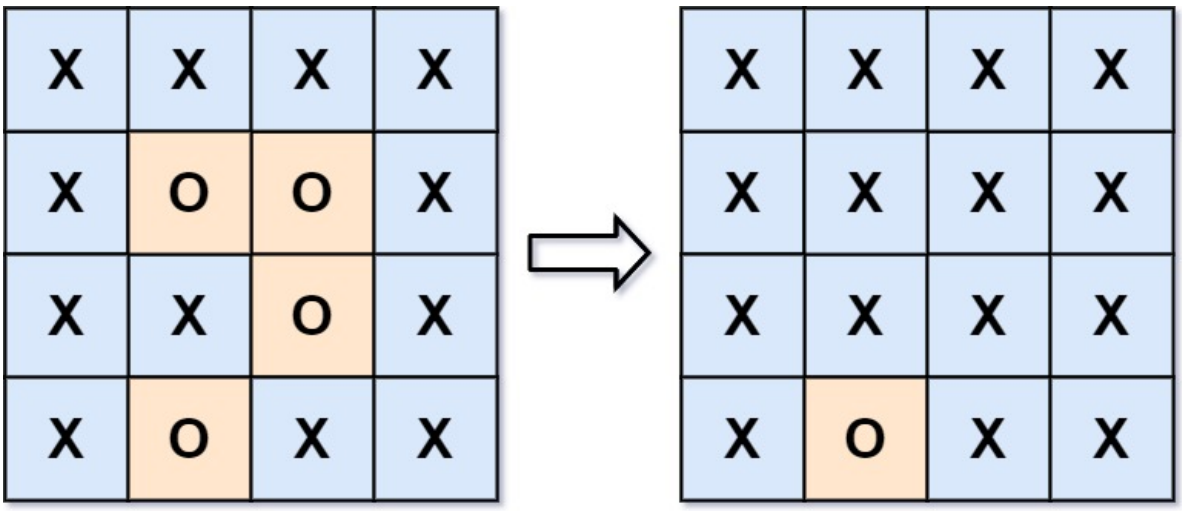
样例输出 #1

```
XXXX
XXXX
XXXX
XOXX
```

提示

对于100%的数据，有$1 \leq n, m \leq 300$。$a[i][j]$的值为`X`或`O`。

样例解释：



思路分析

> 该题涉及到了封闭区域和非封闭区域的问题，具体什么是非封闭区域呢，大家可以想想，如果一块区域连通道了地图的边界，是不是一定不可以被包围，那如果它整个区域都无法走到地图的边界，也就意味着它被包裹在了地图内，所以我们可以分多次dfs，先从地图边界寻找未被包围的区域，对这块区域进行第一次染色，然后再对区域内的岛屿进行第二次染色，最后将两次染色的结果合并即可。

```cpp
#include <bits/stdc++.h>
using namespace std;
int m, n;
int dx[] = {1, -1, 0, 0};
int dy[] = {0, 0, 1, -1};
vector<string> a;
void dfs(int x, int y) {
    if (x < 0 || x == m || y < 0 || y == n || a[x][y] != 'O') {
        return;
    }
    a[x][y] = 'A';
    for (int d = 0; d < 4; d++) {
        int nx = x + dx[d], ny = y + dy[d];
        dfs(nx, ny);
    }
}
int main() {
    cin >> m >> n;
    a.resize(m);
    for (int i = 0; i < m; i++) cin >> a[i];
    for (int i = 0; i < m; i++) {
        dfs(i, 0);
        dfs(i, n - 1);
    }
    for (int j = 1; j < n - 1; j++) {
        dfs(0, j);
        dfs(m - 1, j);
    }
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            if (a[i][j] == 'A') {
                a[i][j] = 'O';
            } else if (a[i][j] == 'O') {
                a[i][j] = 'X';
            }
        }
    }
    for (int i = 0; i < m; i++) cout << a[i] << endl;
    return 0;
}
```