写在最前面

2021.ICPC.澳门.A

有一个 n 行 n 列的迷宫，其中，每一格高度$h_{i,j}$,高度是1到 n^2 的排列

找到一条**通过每个单元格的路径**，使向上爬的次数，不多于向下爬的次数。

$(1 \le T \le 100)$

$(2 \le n \le 64)$

```
1
2
4 3
2 1

4 3 1 2
```

说说大家的想法（这里找个同学回答一下？）

n小

方阵，找特定的连通块

好，如果是搜索，时间复杂度是多少?

那么可以是搜索吗?

B. Matrix of Differences

MarkDown视图 Copy 已翻译

For a square matrix of integers of size $n \times n$, let's define its **beauty** as follows: for each pair of side-adjacent elements x and y, write out the number $|x - y|$, and then find the number of different numbers among them.

For example, for the matrix $\begin{pmatrix} 1 & 3 \\ 4 & 2 \end{pmatrix}$ the numbers we consider are $|1-3|=2$, $|1-4|=3$, $|3-2|=1$ and $|4-2|=2$; there are 3 different numbers among them (2, 3 and 1), which means that its beauty is equal to 3.

You are given an integer n. You have to find a matrix of size $n \times n$, where each integer from 1 to n^2 occurs exactly once, such that its **beauty** is the maximum possible among all such matrices.

DeepL 翻译

对于大小为 $n \times n$ 的整数正方形矩阵，我们可以这样定义它的美：对于每一对边相邻的元素 x 和 y ，写出数字 $|x-y|$ ，然后求出其中不同数字的个数。

例如，对于矩阵 $\begin{pmatrix} 1 & 3 \\ 4 & 2 \end{pmatrix}$ ，我们考虑的数是 $|1-3|=2$ 、 $|1-4|=3$ 、 $|3-2|=1$ 和 $|4-2|=2$ ；其中有 3 个不同的数（ 2 、 3 和 1 ），这意味着它的美等于 3 。

给你一个整数 n 。你必须找到一个大小为 $n \times n$ 的矩阵，其中从 1 到 n^2 的每个整数都恰好出现一次，使得它的美是所有这样的矩阵中可能出现的最大值。

MarkDown视图 Copy 翻译

Input

The first line contains a single integer t $(1 \le t \le 49)$ – the number of test cases.

The first (and only) line of each test case contains a single integer n $(2 \le n \le 50)$.

MarkDown视图 Copy 翻译

Output

For each test case, print n rows of n integers — a matrix of integers of size $n \times n$, where each number from 1 to n^2 occurs exactly once, such that its beauty is the maximum possible among all such matrices. If there are multiple answers, print any of them.

Example

input Copy

```
2
2
3
```

output Copy

```
1 3
4 2
1 3 4
9 2 7
5 8 6
```

GNU G++20 11.2.0 (64 bit, winlibs) 字体大小: 15 全屏 固定到底部 固定到右侧

1 洛谷

这个是我当时第一周训练的时候，打CF遇到的题目。

我一看这个数据规模，就开始搜索了。。。。

它是搜索吗？搜索的时间复杂度是多少？

$O(n^2!)$

排列

但是需要这么做吗?

我们注意到，最大的beauty是多少?

n^2-1

能不能达到?

$[n^2,1,n^2-1,2,n^2-2,3,...]$

这一部分，我想表达的是：**搜索不能代替思考。**

不要被知识套牢了。

搜索是重要的基础，这个毋庸置疑。

但是，搜索不意味着，放弃思考。

有的题就是会故意把数据范围写的很小，但是搜索又过不去。

要主动思考。

这里可能会涉及一些，**构造**，这个是后话了

指数搜索

以下我想讲一下DFS的基本套路。

- 终止条件
- 转移过程

题目描述

你有 n 种可支配的配料。

对于每一种配料，各自有酸度 s 和苦度 b。

总的酸度为每一种配料的**酸度总乘积**；总的苦度为每一种配料的**苦度的总和。**

我们希望选取配料，以使得**酸度和苦度的绝对差最小。**

必须添加至少一种配料

有 $1 \leq n \leq 10$，酸度和苦度不同时为 1 和 0。

```
2
3 8
5 8

1
```

我们发现，每种配料，只有两种状态，要么选，要么不选。

由此，DFS的流程是：

由此，我们可以这样理解搜索

搜索是一种按一定策略的枚举

“一定策略”，可以解释为：“相邻状态的转移”

代码

```
void DFS(int pos, int x, int y)
{
    if(pos == n + 1)//结束条件
    {
        if(x == 1 && y == 0)//一个都没有选，直接结束，不更新答案
            return ;

        ans = std::min(std::abs(x - y), ans);//更新答案
        return ;
    }

    DFS(pos + 1, x * a[pos], y + b[pos]);
    DFS(pos + 1, x, y);
}
...
...
DFS(1, 1, 0);//搜索的初始条件，从第1个开始，初始酸度是1，苦度是0
```

我们来总结一下这种搜索的大概模板。。

首先，要有一个（多个）递归边界，递归结束的条件。

后面就是递归的转移：

我们以序号，为转移状态，同时记录它的过程量。

方格与特定连通块

关于这类搜索的转移

这里我们顺便理以下BFS的大概流程

首先先是比较经典的，方格上的转移

```
using namespace std;
int dx[8] = {-2, -2, -1, -1, 1, 1, 2, 2};
int dy[8] = {-1, 1, -2, 2, -2, 2, -1, 1};
int val[305][305];
int l, sx, sy, ex, ey;
```



```

void BFS()
{
    queue<pair<int, int>> q;
    q.push({sx, sy});
    while(!q.empty()) {
        int px = q.front().first, py = q.front().second;
        q.pop();
        for(int i = 0; i < 8; i++) {
            int nx = px + dx[i];
            int ny = py + dy[i];
            if(nx >= 0 && ny >= 0 && nx < l && ny < l && !val[nx][ny])
            {
                val[nx][ny] = val[px][py] + 1;
                q.push({nx, ny});
            }
        }
    }

}
```


或者这样写：

```
int n, m;
//int num = 0;
int d[4][2] = {{-1, 0}, {1, 0}, {0, 1}, {0, -1}};
int vis[N];
bool rule(int x, int y)
{
    if (x < 0 || x >= n || y < 0 || y >= m) return 0;
    if (s[x][y] == '#') return 0;
    return 1;
}
int cnt(int x, int y)
{
    int noblock = 0;
    for (int i = 0; i < 4; i++)
    {
        int tx = x + d[i][0];
        int ty = y + d[i][1];
        if (rule(tx, ty) == 1 && vis[ tx * m + ty ] == 0) noblock++;
    }
    return noblock;
}
```


千万不要写分支

一般来说，**先判断出边界**，再判断特定的连通。（RE）

我们拿山峰与山谷这题举个例子。

[山峰和山谷 Ridges and Valleys]

思路

找特定条件的连通块，对于每一个连通块，判断是山峰还是山谷

实现

```
bool H, L;
int numH = 0, numL = 0;

bool rule(int x, int y){return (x < 1 || x > n || y < 1 || y > n);}
void BFS(int x, int y)
{
    queue < pair < int , int > > q;
    vis[x][y] = 1;  q.push({x, y});
    while (!q.empty())
    {
        auto p = q.front();
        q.pop();
        for (int i = 0; i < 8; ++i)
        {
            int tx = p.fi + d[i][0], ty = p.se + d[i][1];
            if (rule(tx, ty))continue;
            if (a[tx][ty] > a[x][y])
                H = 0;
            if (a[tx][ty] < a[x][y])
                L = 0;

            if (vis[tx][ty] == 0 && a[tx][ty] == a[x][y])
            {
                vis[tx][ty] = 1;
                q.push({tx, ty});
            }
        }
    }
    return ;
}
int main()
{
    ...
    for (int i = 1; i <= n; ++i)
    {
        for (int j = 1; j <= n; ++j)
        {
            if (vis[i][j] == 0)
            {
                H = 1; L = 1;
                BFS(i, j);
                if (H == 1)++numH;
```

```
                if (L == 1)++numL;
            }
        }
    }
    ...
```

洪水填充

```
6
0 0 0 0 0 0
0 0 1 1 1 1
0 1 1 0 0 1
1 1 0 0 0 1
1 0 0 0 0 1
1 1 1 1 1 1

0 0 0 0 0 0
0 0 1 1 1 1
0 1 1 2 2 1
1 1 2 2 2 1
1 2 2 2 2 1
1 1 1 1 1 1
```

这里当时讲的比较快，我们又没有拉题目，这里仔细讲一下。

关于DFS和BFS

选择

- 找到最优解：BFS，因为我要全部状态的比较。
- 找到可行解，判断是否可行，DFS，因为我只要一个，不需要跑完全部的状态。
    - 如果，不可行，DFS会跑满

这里我们讲细一点，画出树形图

如果BFS和DFS都可以，一般BFS更保险，考虑栈溢出。

比如，洪水填充，DFS的是n^2的栈空间，BFS是$4n$的队列空间。

调试

输出中间量

加计数器，看看到底跑了多少次。

加了剪枝之后，复杂度变得很难算了。

或者说你写假了，跑不出来。

乘法逆元2

思路：

我看到大部分人都会先写一个，朴素算法

对于每个数求它的逆元。

OK，这个是什么时间复杂度?

$O(nlogn)$

为什么?

以上，我们得出了，**分别对于每个数求逆元再计算** 不可行。

那么，问题变成了**怎么降低复杂度**?

我们希望，它的复杂度是 $O(n)$ 的水平

但是，**逆元不可避免的有一个**$logn$

怎么办?

我们有什么工作是多余的?

“我们需要对所有的数，做逆元吗？”

“太累了，我们少做一点，我们只做一次，行不行？”

这里我们不妨想到这样一个性质

$$a_i{}^{-1} = (a_1 * a_2 * \cdots a_{i-1}) * (a_1 * a_2 * \cdots a_{i-1} * a_i)^{-1}$$

！！预处理

具体的说，是前缀积。

我们只要求所有元素的积的逆元。

在每一项拆出来，就是每个数的逆元。

```
#define MAXN 5000005
#define int long long
const int mod = 1e9 + 7 ;
const int M = 998244353;
int inv[MAXN], pre[MAXN], invpre[MAXN];
```

```
int qpow(int x)
{
    int res = 1;
    int b = mod - 2;
    while (b)
    {
        if (b & 1)
            res = res * x % mod;
        x = x * x % mod;
        b >>= 1;
    }
    return res % mod;
}
signed main()
{
    int n = readin();
    int ans = 0;
    pre[0] = 1;
    for (int i = 1; i <= n; ++i)
    {
        a[i] = readin();
        pre[i] = pre[i - 1] * a[i] % mod;
    }

    invpre[n] = qpow(pre[n]);
    for (int i = n - 1; i >= 0; --i)
        invpre[i] = invpre[i + 1] * a[i + 1] % mod;

    ...
    for (int i = 1; i <= n; ++i)
    {
        inv[i] = invpre[i] * pre[i - 1] % mod;
        ...

    }
}
```

复杂度就是 $O(n + logn) = O(n)$

（这里数组如果开的比较奢侈，洛谷上会MLE，LOJ没问题）