## 困难的串

如果一个字符串包含两个相邻的重复子串, 则称它是“容易的串”, 其他串称为“困难的串”.

容易的串: BB、ABCDACABCAB、ABCDABCD.

困难的串: D、DC、ABDAB、CBABCBA.

输入正整数n和L, 输出由前L个字符组成的、字典序第n小的困难串. 例如, 当L=3, 前7个困难的串分别为: A、AB、ABA、ABAC、ABACA、ABACAB、ABACABA. 输入保证答案不超过80个字符.

| Input | Output |
|---|---|
| 7 3<br>30 3 | ABACABA<br>ABACABCACBABCABACABCACBACABA |

## 题目分析

- 利用递归构造(搜索), 同时进行判断;
- 由于是递归构造, 在判断是否是困难的串时, 只需要判断必须具有当前字符作为后缀的子串即可, 因为前面的其他子串已经在当前递归层次前就判断过了;
- 值得注意的是, 当找到一个困难的串时, 由于是按字典序寻找, 因此不需要回溯, 只需要通过继续加长寻找下一个困难的串即可. 只有不能通过加长字符串得到下一个困难的串才需要回溯.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| A | B | A | C | A | B | A | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

## 参考程序

```
#include <stdio.h>
#define N 1000
int n, L, cnt, s[N];
int dfs(int cur);//返回值为1表示已找到解
int main()
{
    freopen("1.in", "r", stdin);
    freopen("1.out", "w", stdout);
    while(scanf("%d%d", &n, &L) == 2)
    {
        cnt = 0;
        dfs(0);
    }
    return 0;
}
```

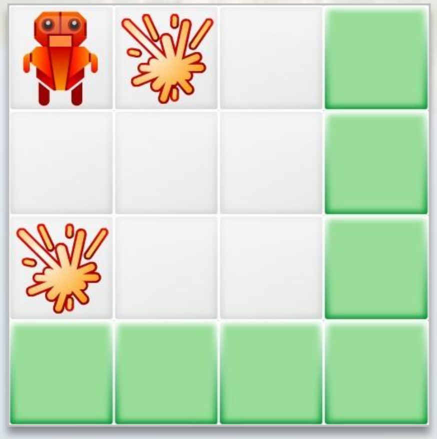
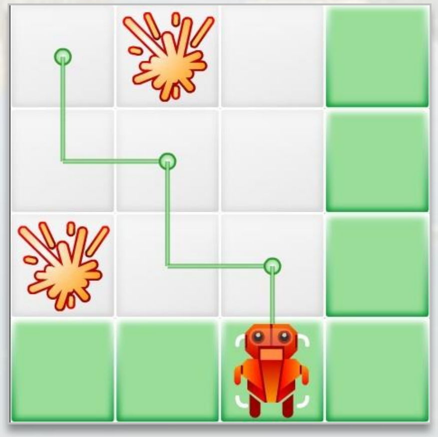
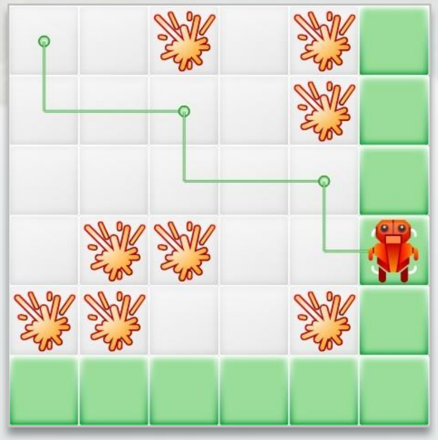
```
int dfs(int cur) {
    int i, j, k;
    if(cnt == n) {
        for(i = 0; i < cur; i++)   printf("%c", s[i] + 'A');
        printf("\n");
        return 1;
    }
    for(i = 0; i < L; i++) { //试探着在下标cur处放字符形成困难的串
        s[cur] = i;
        for(j = 1; j <= (cur+1)/2; j++) { //检查长度为j的子串
             for(k = 0; k < j; k++)  if(s[cur-k] != s[cur-j-k])   break;
             if(k == j)      break; //存在长度为j的子串相等
        }
        if(j > (cur+1)/2) { //说明是困难串
            cnt++;
            if(dfs(cur+1) == 1)      return 1;
        }
    }
    return 0;
}
```

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| A | B | A | C | A | B | A | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

# 大逃亡

love8909遇到危险了!!! 他被困在一个迷宫中, 彷徨而无助. 现在需要你来帮助他走出困境. 他只能记住指定长度的指令(指令的长度由MinLen和MaxLen限定), 并循环执行, 而且他只会向下或向右(很奇怪吧^_^). 他在地图的左上角, 你需要告诉他一个运动序列, 即向下(D)或向右(R), 使他能够成功走出这个图且不碰到陷阱.

如果还不明白, 可以参看图片. 图片1,2对应样例的第1组, 图片3对应样例的第2组.

- **Standard Input**

第一行为1个整数T, 表示有T组测试数据.

第二行为4个整数Height, Width, MinLen,MaxLen, 分别表示地图的高,宽, 命令序列的最小和最大长度. 3 <= Height, Width <= 60, 2 <= MinLen <= MaxLen <= 35.

第三行至第Height+2行为地图信息. 其中'.'表示空地, 'X'表示陷阱.

- **Standard Output**

只有一行, 为命令序列(只含D, R).

注意: 如果有多解, 输出命令长度最短的; 依然有多解, 输出字典序最小的(D的字典序比R小), 数据保证一定存在一组解.

字典序: 字符串从前往后依次比较, 第一个字符不同的位置, 字符较小的字符串字典序较小, 详情参见字典.

## Samples

| Input | Output |
|---|---|
| <pre>2<br>3 3 2 2<br>. X.<br>. . .<br>X. .<br>5  5  2  3<br>. . X. X<br>. . . . X<br>. . . . .<br>. XX. .<br>XX. . X</pre> | <pre>DR<br>DRR</pre> |

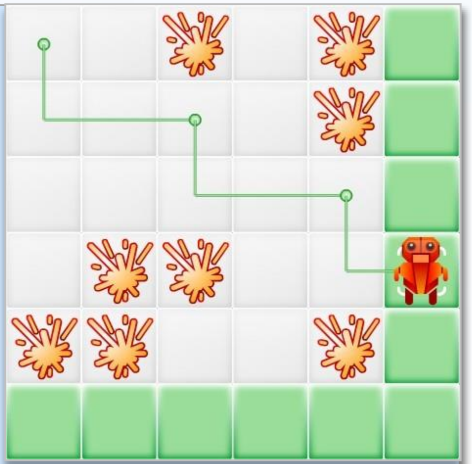
## 题目分析

在给定的指令序列下, 如何判断能否走出迷宫?

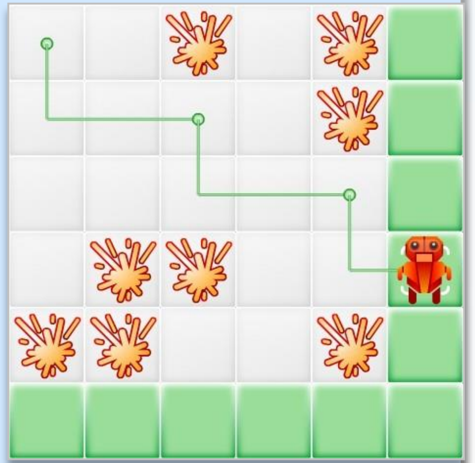
- 由于love8909是按照指令循环执行, 所以第一个循环走的路径确定后, 以后循环的路径是完全是确定的.
- 可以在第一次循环走到某步时, 就同时判断后面循环在此步是否落入陷阱, 以尽快确定该指令序列是否可行.
- 题目要求指令最短, 最短情形下还要求字典序最小.
- 指令可以从短到长枚举, 指令序列中优先枚举向下, 再枚举向右即可.

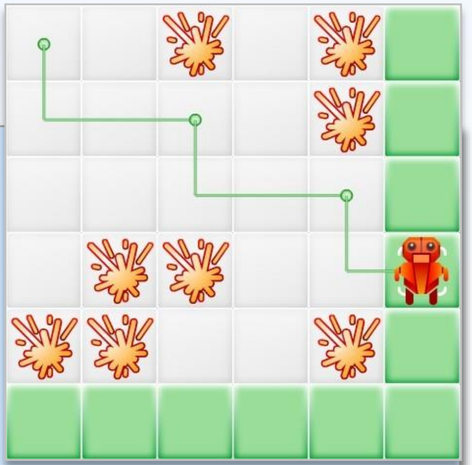
## 参考程序

```
#include <iostream>
#include <cstdio>
int Height, Width, MinLen, MaxLen, sH, sW;
char MAP[64][64];
bool in(int x, int y) //检查当前点在图内否
{
    return x >= 0 && x < Height && y >= 0 && y < Width;
}
bool check(int x, int y) //检查在数个矩形
{                        //重复执行时对应位置是否有障碍
    while (in(x, y))
    {
        if (MAP[x][y] == 'X')  return false;
        x += sH; y += sW; //确保重复执行时当前位置不会遇障碍
    }
    return true;
}
```



```
char res[128];
bool dfs(int x,int y,int D,int R)//探索在向下和向右命令序列
{                                //长度给定下，能否走出去
        if (check(x, y) == false)  return false;
        if (D == 0 && R == 0) {
                res[x + y] = '\0';
                printf("%s\n", res);
                return true;
        }
        if (D > 0) {//由于字典序，先考虑向下再考虑向右
                res[x + y] = 'D';
                if (dfs(x + 1, y, D - 1, R)) return true;
        }
        if (R > 0) {
                res[x + y] = 'R';
                if (dfs(x, y + 1, D, R - 1)) return true;
        }
        return false;
}
```





```
int main()
{
    int T;
    scanf("%d", &T);
    while (T--)
    {
        scanf("%d %d %d %d", &Height, &Width, &MinLen, &MaxLen);
        for (int i = 0; i < Height; i++)
            scanf("%s", MAP[i]);
        bool find = false;
           //先序列总长度枚举，再按向下的命令长度枚举
        for (int L = MinLen; !find && L <= MaxLen; L++)
            for (sH = L; !find && sH >= 0; sH--)
                find = dfs(0, 0, sH, sW = L - sH);
    }
    return 0;
}
```

# Restore Equations

An interstellar expedition team has recently discovered on planet Mars some multiplication equations, which are believed to be the proof that Mars has once been the home of some intellectual species. However these equations are incomplete where some digits are missing because of the eroding power of Mars' nature. Here is one example.

$$\begin{array}{cccc} & 1 & 2 & \square \\ \times & & 4 & 5 \\ \hline \square & 5 & \square & 5 \end{array}$$

One way to restore this equation is to assume the 3 missing digits are 3, 5, 3, respectively, obtaining 123 x 45 = 5535, which indicates this equation may indeed be the intellectual output of Mars' ancient habitants, rather than just some random numbers.

There has been hot debate over this, because people aren't sure whether they can restore all the equations discovered on Mars, i.e. restore the missing digits such that the multiplication equation holds. As a programmer in NASA, you are assigned the task of checking if all the equations are restorable.

## Standard Input

The first line is an integer T, number of equations to check. Next T lines each contains three non-empty strings A, B and C, separated by spaces. Each string contains digits('0'-'9') and/or asterisks '*' only, where an asterisk stands for a missing digit. No string begins with the digit '0'.

## Standard Output

For each test case, output "Yes"(Quotes for clarity only) on a line if one can replace the asterisks with digits such that afterwards the numbers represented by A, B and C satisfy A x B = C. Otherwise output "No" instead. Note that an asterisk must be replaced with exactly one digit('0'-'9') and the resulting numbers A, B and C can't start with zeros(See sample input/output for more clarification).

- The length of string A and B are at most 3, and the length of C is at most 6.
- The length of each string is greater than zero.
- At least one '*' will be present in each equation.

## Samples

| Input | Output |
|---|---|
| 2<br>12*   45   *5*5<br>11   11   *121 | Yes<br>No |

## 题目分析

- 先通过深度优先搜索完成第一个乘数的枚举;
- 再通过深度优先搜索完成第二个乘数的枚举;
- 接着计算检查是否满足等式;
- 如果满足, 则完成; 否则回溯继续刚才的过程.

## 参考程序

```
#include <bits/stdc++.h>
string A,B,C;
int lena,lenb,lenc;
int a[3],b[3];
bool flag = false;
int main() {
    int ncase; scanf("%d",&ncase);
    while(ncase--){
        scanf("%s %s %s", A, B, C);
        lena = A.length(); lenb = B.length(); lenc = C.length();
        reverse(A.begin(), A.end());
        reverse(B.begin(), B.end());
        reverse(C.begin(), C.end());
        for(int i = 0; i < 3; ++i) a[i] = 0, b[i] = 0;
        flag = false; dfs(0);
        if(flag) printf("Yes\n");
        else printf("No\n");
    }
    return 0;
}
```

```
void check()
{
    int x = a[0] + a[1]*10 + a[2]*100;
    int y = b[0] + b[1]*10 + b[2]*100;
    int z = x * y;
    for(int i = 0; i < lenc-1; ++i)
    {
        if(C[i] == '*')
        {
            z /= 10;continue;
        }
        if(z%10+'0' != C[i])return;
        z /= 10;
    }
    if ((z==0 && C[lenc-1]!='0') || z>=10)
        return;
    if (z==C[lenc-1]-'0' || C[lenc-1]=='*')
        flag = true;
}
```

```
void dfs1(int n) //对第二个乘数搜索
{
    if(flag) return;
    if(n == lenb)
    {
         check();   return;
    }
    if(B[n] == '*')
    {
        for(int j = (n==lenb-1); j <= 9; ++j)
        {
              b[n] = j;     dfs1(n+1);
        }
    }
    else
    {
        b[n] = B[n]-'0';
        dfs1(n+1);
    }
}
```

```
void dfs(int n) //对第一个乘数搜索
{
    if(flag) return;
    if(n == lena)
    {
      dfs1(0); return;
    }
    if(A[n] == '*')
        for(int j = (n==lena-1); j <= 9; ++j)
        {
            a[n] = j;
            dfs(n+1);
        }
    else
    {
        a[n] = A[n]-'0';
        dfs(n+1);
    }
}
```

## 五、宽度优先搜索

- 宽度优先搜索算法(又称广度优先搜索)是最简便最常用的图的搜索算法之一, 这一算法也是很多重要的图的算法的原型. 图论中的Dijkstra单源最短路径算法和Prim最小生成树算法都采用了和宽度优先搜索类似的思想.
- 其别名又叫BFS, 属于一种盲目搜寻法, 目的是系统地展开并检查图中的所有节点, 以找寻结果. 换句话说, 它并不考虑结果的可能位置, 彻底地搜索整张图, 直到找到结果为止.

## Find The Multiple

**Given a positive integer n, write a program to find out a nonzero multiple m of n whose decimal representation contains only the digits 0 and 1. You may assume that n is not greater than 200 and there is a corresponding m containing no more than 100 decimal digits.**

- **Standard Input**

**The input file may contain multiple test cases. Each line contains a value of n (1 <= n <= 200). A line containing a zero terminates the input.**

- **Standard Output**

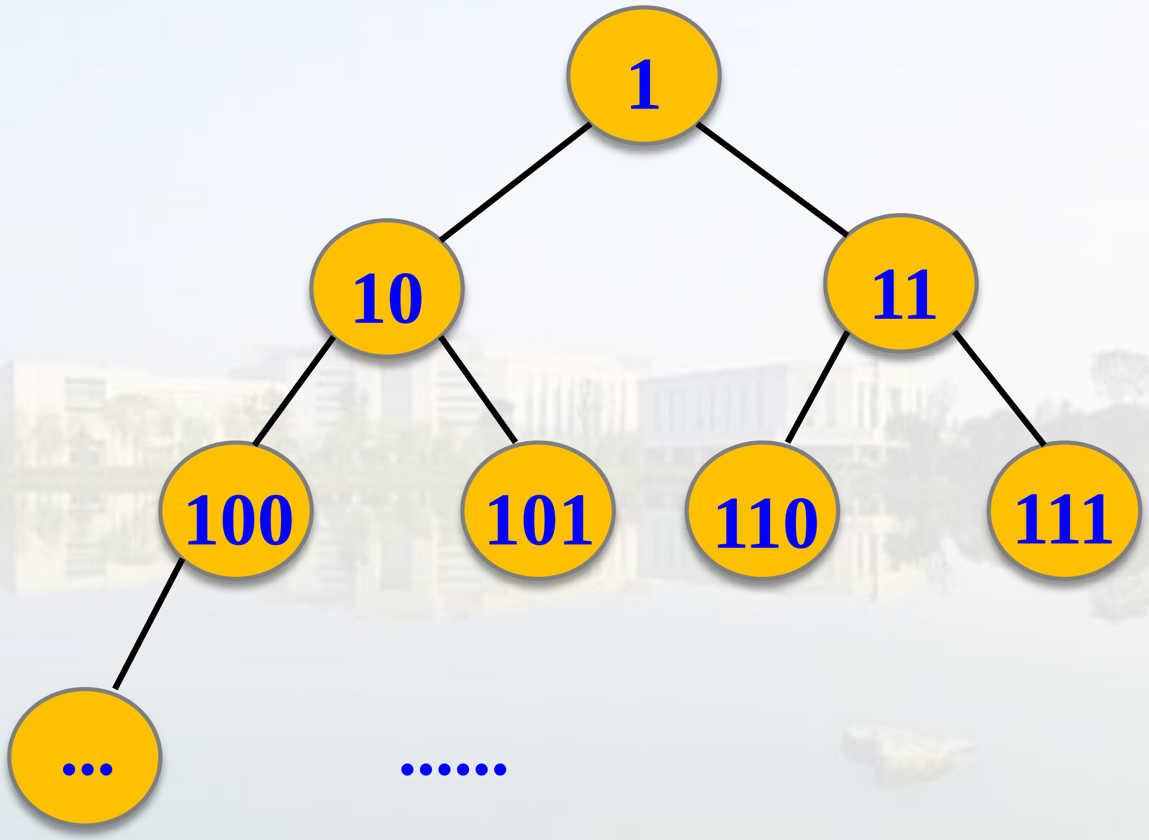
**For each value of n in the input print a line containing the corresponding value of m. The decimal representation of m must not contain more than 100 digits. If there are multiple solutions for a given value of n, any one of them is acceptable.**

- **Samples**

| Input | Output |
|---|---|
| 2<br>6<br>19<br>0 | 10<br>100100100100100100<br>1111111111111111111 |

## 题目分析

- 本题等同于构造一颗二叉树，然后按层次去遍历这颗树.



## 参考程序

```
#include<bits/stdc++.h>
using namespace std;
int n;
void BFS(){
    long long top;
    queue<long long> q;
    q.push(1);
    while(!q.empty()) {
        top = q.front();
        q.pop();
        if(top % n == 0) break;
        q.push(top * 10);
        q.push(top * 10 + 1);
    }
    printf("%lld\n", top);
}
int main() {
    while(scanf("%d",&n)&&n) BFS();
    return 0;
}
```

## Prime Path

**The ministers of the cabinet were quite upset by the message from the Chief of Security stating that they would all have to change the four-digit room numbers on their offices.**

**— It is a matter of security to change such things every now and then, to keep the enemy in the dark.**

**— But look, I have chosen my number 1033 for good reasons. I am the Prime minister, you know!**

**— I know, so therefore your new number 8179 is also a prime. You will just have to paste four new digits over the four old ones on your office door.**

**— No, it's not that simple. Suppose that I change the first digit to an 8, then the number will read 8033 which is not a prime!**

**— I see, being the prime minister you cannot stand having a non-prime number on your door even for a few seconds.**

**— Correct! So I must invent a scheme for going from 1033 to 8179 by a path of prime numbers where only one digit is changed from one prime to the next prime**

Now, the minister of finance, who had been eavesdropping, intervened.

— No unnecessary expenditure, please! I happen to know that the price of a digit is one pound.

— Hmm, in that case I need a computer program to minimize the cost. You don't know some very cheap software gurus, do you?

— In fact, I do. You see, there is this programming contest going on... Help the prime minister to find the cheapest prime path between any two given four-digit primes! The first digit must be nonzero, of course. Here is a solution in the case above.

1033
1733
3733
3739
3779
8779
8179

The cost of this solution is 6 pounds. Note that the digit 1 which got pasted over in step 2 can not be reused in the last step – a new 1 must be purchased.

- **Standard Input**

**One line with a positive number: the number of test cases (at most 100). Then for each test case, one line with two numbers separated by a blank. Both numbers are four-digit primes (without leading zeros).**

- **Standard Output**

**One line for each case, either with a number stating the minimal cost or containing the word Impossible.**

- **Samples**

| Input | Output |
|---|---|
| 3<br>1033 8179<br>1373 8017<br>1033 1033 | 6<br>7<br>0 |

## 题目分析

- 首先区分the prime minister 和prime number中的prime的不同意思.
- 从起始素数开始进行广搜, 每轮将所有可行的改变 (个位至千位, 每个位置进行一次改变) 放入搜寻队列一次进行素数判断.
- 用数组来记载转变路径, 每个结点指向其父结点. 达到目标之后向上寻找到祖先, 即可求出转变了多少次, 也可以得到转变路径.

## 参考程序

```
#include <bits/stdc++.h>
using namespace std;
int a, b;
int dis[9999] = { 0 };
int visited[9999] = { 0 };
bool isprime(int x) {
    for (int i = 2; i <= sqrt((double) x); i++)
        if (x % i == 0)
            return false;
    return true;
}
int bfs(int s, int r){
    if(s == r) return 0;
    queue<int> q;
    q.push(s);
    dis[s] = 0;  //距起点距离
    visited[s] = 1;  //访问标志
    while (!q.empty()) {
        int temp = q.front(); q.pop();
        for (int i = 0; i <= 9; i++) {
            int y1 = (temp / 10) * 10 + i;//改变个位
            if (isprime(y1) && !visited[y1]) {
                q.push(y1);
                dis[y1] = dis[temp] + 1; //距离增1
                visited[y1] = 1;
            }
            int y2 = temp%10 + (temp/100) * 100 + i*10;
            if (isprime(y2) && !visited[y2]) {
                q.push(y2);
                dis[y2] = dis[temp] + 1;
                visited[y2] = 1;
            }
            int y3 = temp%100 + (temp/1000) * 1000 + 100*i;
            if (isprime(y3) && !visited[y3]) {
                q.push(y3);
                dis[y3] = dis[temp] + 1;
                visited[y3] = 1;
            }
            if (i != 0) {
                int y4 = temp % 1000 + i * 1000;
                if (isprime(y4) && !visited[y4]) {
                    q.push(y4);
                    dis[y4] = dis[temp] + 1;
                    visited[y4] = 1;
            }   }
            if (visited[r])  return dis[r];
        }
    }
    return 0;
}
int main() {
      int n; scanf ("%d",&n);
      while (n--) {
            memset(visited,0,sizeof(visited));
            memset(dis,0,sizeof(dis));
            scanf ("%d %d",&a,&b); printf("%d\n", bfs(a, b));
      }
      return 0;
}
```

## 连通块

一个n*m的方格图, 一些格子被涂成了黑色, 在方格图中被标为1, 白色格子标为0. 问有多少个四连通的黑色格子连通块. 四连通的黑色格子连通块指的是一片同黑色格子组成的区域, 其中的每个黑色格子能通过四连通的走法 (上下左右), 只走黑色格子. 到达该连通达中的其他黑色格子.

- **Standard Input**

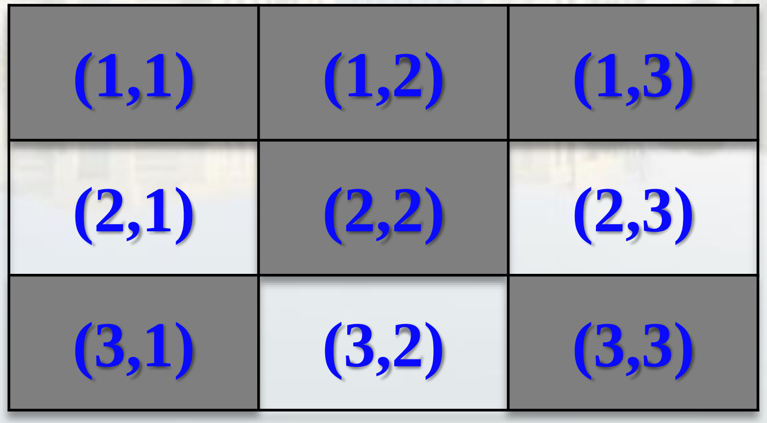
第1行, 两个整数n,m(1<=n,m<=100), 表达一个n*m的方格图; 下来n行, 每行m个整数, 分别为0或1, 表示这个格子是黑色还是白色.

- **Standard Output**

1行, 一个整数ans, 表示图中有ans个黑色格子连通块.

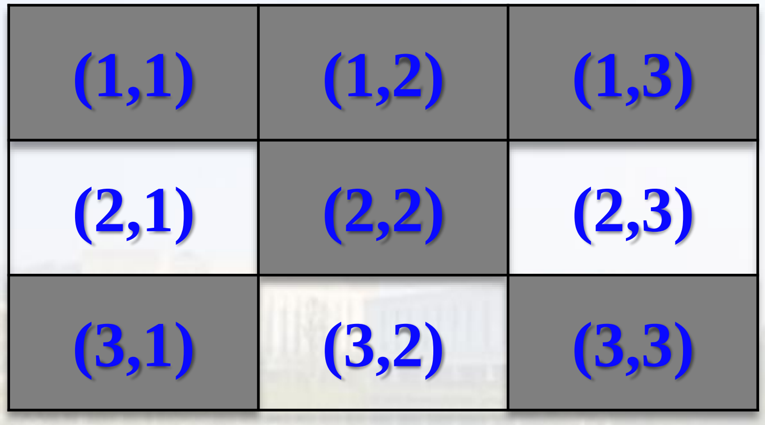
## Samples

| Input | Output |
|---|---|
| 3 3<br>1 1 1<br>0 1 0<br>1 0 1 | 3 |



## 题目分析

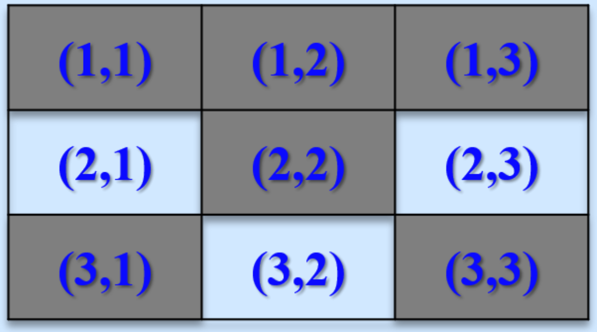
- 可以使用广度优先搜索.



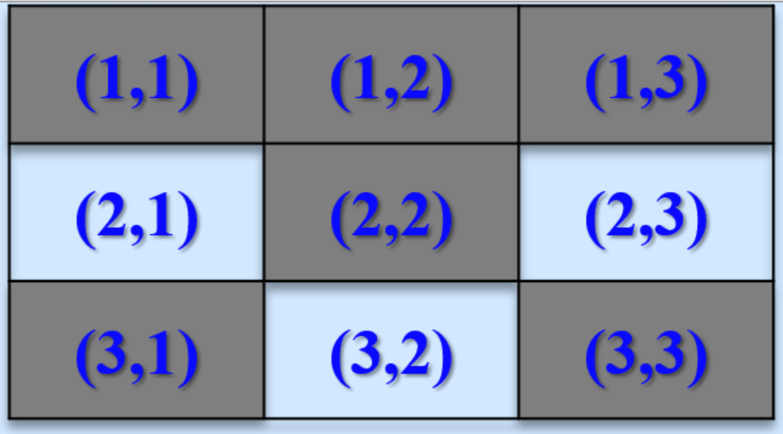
| | | |
|---|---|---|
| (1,1) | (1,2) | (1,3) |
| (2,1) | (2,2) | (2,3) |
| (3,1) | (3,2) | (3,3) |

## 参考程序

```
#include <iostream>
using namespace std;
const int N = 110;
const int flag[4][2] = {{0,1},{0,-1},{1,0},{-1,0}};
int n, m, ans, a[N][N], queue[N*N][2];
bool p[N][N];
void bfs(int x, int y) {
    int front = 1, rear = 2;
    queue[1][0] = x; queue[1][1] = y;
    while(front < rear) {
        x = queue[front][0];   y = queue[front++][1];
        for(int i = 0; i < 4; i++)   {
            int x1 = x + flag[i][0], y1 = y + flag[i][1];
            if(x1<1 || y1<1 || x1>n || y1>m || !a[x1][y1] || p[x1][y1])
                continue;
            p[x1][y1] = true;
            queue[rear][0] = x1;
            queue[rear++][1] = y1;
        }
    }
}
```



```
int main()
{
    cin >> n >> m;
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= m; j++)
            cin >> a[i][j];
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= m; j++)
            if(a[i][j] && !p[i][j])
            {
                ans++;
                bfs(i,j);
            }
    cout << ans << endl;
    return 0;
}
```

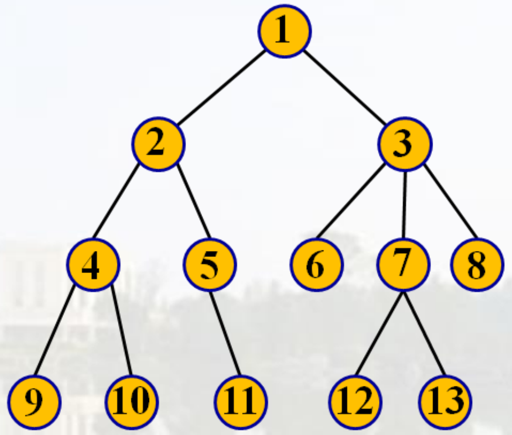


# 八数码问题

编号为1~8的8个正方形滑块被摆成3行3列(有一个格子留空). 每次可以把与空格相邻的滑块(有公共边才算相邻)移到空格中, 而它原来的位置就成为了新的空格. 给定初始局面和目标局面(用0表示空格), 你的任务是计算出最少的移动步数. 如果无法到达目标局面, 则输出-1.

| | | |
|---|---|---|
| 2 | 6 | 4 |
| 1 | 3 | 7 |
| | 5 | 8 |

| | | |
|---|---|---|
| 8 | 1 | 5 |
| 7 | 3 | 6 |
| 4 | | 2 |



9!=362880

| Input | Output |
|---|---|
| 1<br>2 6 4 1 3 7 0 5 8<br>8 1 5 7 3 6 4 0 2 | 31 |