深度搜索

1、深度搜索基础

深搜模板题。

本题要求为矩阵每个点赋值，可以定义函数，来为矩阵赋值，比如：定义函数 dfs(int x,int y)，含义是为 x、y 点赋值。

调用函数时，首先为 1,1 点赋值为 1，也就是 dfs(1,1)，既然函数能够为 1,1 点赋值，也就能为 1,2 点赋值，以此类推，是典型的递归思想，要注意，每次递归，所赋的值要 +1。

本题需要掌握下列内容：

1、理解赋值的顺序，任意给定一个 n*m 的矩阵，能够写出赋值的结果；

2、理解递归的过程，任意给定一个 n*m 的矩阵，能够画出递归的过程；

3、理解程序编写的过程，和实现的原理；

4、理解函数递归调用的过程；

5、理解迷宫类递归的几种不同写法；

解法一：将要填写的位置 x,y，以及要递归的值，作为递归参数

注意：将要填写的值 k 作为递归参数，本题是对的，因为本题从 1,1 点开始，如果本题修改成从任意点开始深搜，那么是不能将要填的值 k 作为递归参数的（参考解法二），大家可以测试一下。

```
#include <bits/stdc++.h>//1-1586-1 javacn
using namespace std;
int n,m;
int a[20][20];
// 为二维数组赋值
// 为 x,y 点赋值为 k
void fun(int x,int y,int k){
	a[x][y] = k;// 赋值
	// 优先向右，其次向下，再次向左，再次向上
	// 递归的条件：不能出边界，且不能访问已经赋值的点
	if(y+1<=m && a[x][y+1]==0) fun(x,y+1,k+1);	// 向右尝试
	if(x+1<=n && a[x+1][y]==0) fun(x+1,y,k+1);	// 向下尝试
	if(y-1>=1 && a[x][y-1]==0) fun(x,y-1,k+1);	// 向左尝试
	if(x-1>=1 && a[x-1][y]==0) fun(x-1,y,k+1);	// 向上尝试
}
```

```
int main()
{
	//n 行 m 列
	cin>>n>>m;
	// 为 1,1 点赋值为 1
	fun(1,1,1);

	// 输出
	int i,j;
	for(i = 1;i <= n;i++)
	{
		for(j = 1;j <= m;j++)
		{
			cout<<setw(3)<<a[i][j];
		}
		cout<<endl;
	}
}
```

解法二：将 x,y 作为递归参数，将要填写的值 k，作为全局变量。

```
#include <bits/stdc++.h>//1-1586-2 javacn
using namespace std;
int a[20][20],n,m,k = 1;
// 为 x,y 点填值
void dfs(int x,int y){
	a[x][y] = k;
	k++;// 每填一个值，值要自增

	//dfs 函数既然能为 xy 点填值，必定能为 xy 右侧点 x,y+1 填值
	// 递归条件：不能出右边界，右侧点没有走过
	if(y+1<=m && a[x][y+1]==0) dfs(x,y+1);
	// 右侧不能走，向下
	if(x+1<=n && a[x+1][y]==0) dfs(x+1,y);
	// 向下不能走，向左
	if(y-1>=1 && a[x][y-1]==0) dfs(x,y-1);
	// 左侧不能走，向上
	if(x-1>=1 && a[x-1][y]==0) dfs(x-1,y);
}

int main()
{
	cin>>n>>m;
	dfs(1,1);// 从 1,1 点开始赋值
	// 输出
	for(int i = 1;i <= n;i++)
	{
		for(int j = 1;j <= m;j++)
		{
			cout<<setw(3)<<a[i][j];
		}
		cout<<endl;
	}
	return 0;
}
```

思路：从出发点开始，探测所有可探测的点，看是否有目标点，如果有，就表示可达，否则表示不可达！

走到终点后，我们要将程序直接停掉，来避免不必要的递归：

（1）return：停止当前的函数，如果函数是递归产生的，不会停止所有的递归，只是停止本次函数的递归，退到上一次调用的地方；

（2）exit(0)：停止程序，无论是否有递归，全部停止。

```
#include <bits/stdc++.h>//2-1430-1  javacn
using namespace std;
/*
1. 判断如果起止点有 1，就不能走；
2. 从起点开始搜索，如果走到过终点，标记；
*/
int a[110][110];
int n;
int ha,la,hb,lb;

// 搜索所有可行的点，走过标记
void dfs(int x,int y)
{
	//cout<<x<<" "<<y<<endl;
	a[x][y] = 1;// 走过的点标记
	// 判断是否到达终点
	if(x==hb&&y==lb)
	{
		cout<<"YES";
		exit(0);
	}

	// 判断四个方向，是否有能走的点，如果有，直接递归执行
	if(y+1<=n&&a[x][y+1]==0) dfs(x,y+1);
	if(x+1<=n&&a[x+1][y]==0) dfs(x+1,y);
	if(y-1>=1&&a[x][y-1]==0) dfs(x,y-1);
	if(x-1>=1&&a[x-1][y]==0) dfs(x-1,y);
}
```

```
int main()
{
	cin>>n;
	for(int i = 1;i <= n;i++)
	{
		for(int j = 1;j <= n;j++)
		{
			cin>>a[i][j];
		}
	}
	cin>>ha>>la>>hb>>lb;

	// 如果起止点不能走，输出 NO
	if(a[ha][la]==1 || a[hb][lb]==1){
		cout<<"NO";
		return 0;
	}

	dfs(ha,la);// 从 ha,la 开始搜索，走过的点标记为 1
	cout<<"NO";
	return 0;
}
```

```cpp
#include<iostream>//2-1430-2   bfs 广搜   jiangyf70
using namespace std;
int a[110][110], r[100100][3];
int fx[] = {0, 1, 0, -1};
int fy[] = {1, 0, -1, 0};
int main()
{
    int n;
    cin >> n;
    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= n; j++) //题目坐标从 1,1 开始的。
            cin >> a[i][j];
    }
    int x1, y1, x2, y2;
    cin >> x1 >> y1 >> x2 >> y2;
    if(a[x1][y1] || a[x2][y2])
    {
        cout << "NO";
        return 0;
    }

    int head = 0, tail = 0;
    r[head][0] = x1;
    r[head][1] = y1;
    a[x1][y1] = 1;
    bool f = false; //是否到终点
```

```
    while(head <= tail && f == false)
    {
        for(int i = 0; i < 4; i++)
        {
            int dx = r[head][0] + fx[i];
            int dy = r[head][1] + fy[i];
            if(dx == x2 && dy == y2)
            {
                f = true;
            }
            else if(dx >= 1 && dx <= n && dy >= 1 && dy <= n && a[dx][dy] == 0)
            {
                tail++;
                r[tail][0] = dx;
                r[tail][1] = dy;
                a[dx][dy] = 1;
                    //cout <<head <<" " << tail <<" "<<  r[tail][0] << " " <<
r[tail][1] << endl;

            }
        }
        head++;
    }
    if(f) cout << "YES";
    else cout << "NO";
    return 0;
}
```

循环每个点，如果当前的点是'W'，则计数器自增1；然后从当前的i，j点开始递归，将上下左右中相邻的为W的点全部标记为'.'。

深搜求解：

```
#include <bits/stdc++.h>//3-1434-1    javacn
using namespace std;

/*
  思路：当遇到积水格，计数器加1，并将该区域的池塘抽干
*/
int n,m;
char a[110][110]; // 默认初值为'\0'
int fx[5] = {0,0,1,0,-1};
int fy[5] = {0,1,0,-1,0};

// 深搜：将xy及相邻的积水点全部标记为.
void dfs(int x,int y)
{
	// 将递归到的有效的点（相邻的有积水的点）标记为.
	a[x][y] = '.';

	int tx,ty;
	// 递归尝试4个方向
	for(int i = 1;i <= 4;i++)
	{
		tx = x + fx[i];
		ty = y + fy[i];

		// 如果该点有效（本题判断相邻格是W，就不用判断是否出边界了）
		if(a[tx][ty] == 'W')
		{
			dfs(tx,ty);
		}
	}
}
```

```
int main()
{
	cin>>n>>m;
	int i,j,c = 0;
	// 读入地图
	for(i = 1;i <= n;i++)
	{
		for(j = 1;j <= m;j++)
		{
			cin>>a[i][j];
		}
	}

	// 依次遍历每个点，如果是池塘，就将计数器 +1，并将相邻池塘单元格全部标记为点
	for(i = 1;i <= n;i++)
	{
		for(j = 1;j <= m;j++)
		{
			// 如果该点是池塘
			if(a[i][j] == 'W')
			{
				c++;
				dfs(i,j);
			}
		}
	}

	cout<<c;
}
```

```cpp
#include <bits/stdc++.h>//3-1434-2  javacn   不使用方向数组的参考解法
using namespace std;

int n,m,c = 0;
char a[110][110];// 存储地图数据

// 将从 x,y 点开始的相邻的 W 标记为 .
void dfs(int x,int y)
{
	a[x][y] = '.';

	// 尝试四个相邻的方向
	// 由于字符数组出了边界不可能是 W, 因此字符数组可以不判断出边界
	if(a[x][y+1]=='W') dfs(x,y+1);
	if(a[x+1][y]=='W') dfs(x+1,y);
	if(a[x][y-1]=='W') dfs(x,y-1);
	if(a[x-1][y]=='W') dfs(x-1,y);
}
```

```
int main()
{
	// 读入
	cin>>n>>m;
	for(int i = 1;i <= n;i++)
	{
		for(int j = 1;j <= m;j++)
		{
			cin>>a[i][j];
		}
	}

	// 从每个点开始，遇到 W，C++，并将相邻的能走到的 W 全标记为 .
	for(int i = 1;i <= n;i++)
	{
		for(int j = 1;j <= m;j++)
		{
			if(a[i][j] == 'W')
			{
				c++;
				// 从该点开始搜索，将相邻的 W 标记为 .
				dfs(i,j);
			}
		}
	}

	cout<<c<<endl;
    return 0;
}
```

```cpp
#include<bits/stdc++.h>//3-1434-3    bfs 广搜    jiangyf70
using namespace std;
int n, m;
char a[110][110];
int r[100100][3];
int fx[] = {0, 1, 0, -1};
int fy[] = {1, 0, -1, 0};
void bfs(int x, int y)
{
    int head = 0, tail = 0;
    memset(r, 0, sizeof(r));
    r[head][0] = x;
    r[head][1] = y;
    a[x][y] = '.';
    while(head <= tail)
    {
        for(int i = 0; i < 4; i++)
        {
            int dx = r[head][0] + fx[i];
            int dy = r[head][1] + fy[i];
            if(a[dx][dy] == 'W')
            {
                tail++;
                r[tail][0] = dx;
                r[tail][1] = dy;
                a[dx][dy] = '.';
            }
        }
        head++;
    }

}
```

```
void print()
{
    for(int i = 0; i < n; i++)
    {
        cout << a[i] << endl;
    }
    cout << endl;
}

int main()
{
    cin >> n >> m;
    for(int i = 0; i < n; i++) cin >> a[i];
    int cnt = 0;
    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < m; j++)
        {
            if(a[i][j] == 'W')
            {
                cnt++;
                bfs(i, j);
                //print();
            }
        }
    }
    cout << cnt;
    return 0;
}
```

```cpp
#include<bits/stdc++.h>//4-1435   wsjszvip
using namespace std;
int n, m;
char a[101][101];
int fx[8] = {-1, 0, 1, 0,-1,-1,1,1};
int fy[8] = {0, 1, 0, -1,-1,1,1,-1};
void dfs(int i, int j)
{
	//标记为点
	a[i][j] = '.';
	int oi, oj, k;
	for (k = 0 ; k < 8 ; k ++)
	{
		oi = i + fx[k];
		oj = j + fy[k];
		// (i-1,0)  (i,j+1) (i+1,j) (i,j-1)
		if (oi >= 1 && oi <= n && oj >= 1 && oj <= m && a[oi][oj] == 'W')
		{
			dfs(oi,oj);
		}
	}
}
```

```
int main()
{
	cin >> n >> m;
	int i, j, c = 0;
	for (i = 1; i <= n ; i ++)
	{
		for (j = 1; j <= m ; j ++)
		{
			cin >> a[i][j];
		}
	}

	for (i = 1; i <= n ; i ++)
	{
		for (j = 1 ; j <= m ; j ++)
		{
			if (a[i][j] == 'W')
			{
				dfs(i, j);
				c ++;
			}
		}
	}
	cout<<c;
	return 0;
}
```

反过来考虑这个问题，判断在圈里面不容易，但判断在圈外面很容易

沿着最外层遍历一圈，从最外层的 0 开始搜索，可以访问到所有封闭的圈以外的 0，标记为 3；

打印：3 打印为 0,1 还是打印为 1,0 打印为 2；

```
#include <bits/stdc++.h>//5-1802-1    javacn
using namespace std;

int n;
int a[40][40];
// 方向值的变化数组
int fx[5] = {0,0,1,0,-1};
int fy[5] = {0,1,0,-1,0};

// 深搜：从 xy 点开始搜索，将所有相邻的 0 标记为 3
void dfs(int x,int y)
{
	a[x][y] = 3;
	// 遍历四方向
	int tx,ty;
	for(int i = 1;i <= 4;i++)
	{
		tx = x + fx[i];
		ty = y + fy[i];
		// 判断在方阵内，且要去的点是 0
		if(tx>=1&&tx<=n&&ty>=1&&ty<=n&&a[tx][ty]==0) dfs(tx,ty);
	}
}
```

```
int main()
{
	cin>>n;
	for(int i = 1;i <= n;i++)
	{
		for(int j = 1;j <= n;j++)	{	cin>>a[i][j];	}
	}
	// 遍历最外层的值，找到 0，深搜
	// 第 1 行和最后一行，一定在最外层，其余每行只有第 1 个和最后一个在最外层
	// 循环每一行
	for(int i = 1;i <= n;i++)
	{
		// 只有第 1 行和最后一行要循环每一列
		if(i == 1 || i == n)
		{
			for(int j = 1;j <= n;j++)	{	if(a[i][j] == 0) dfs(i,j);	}
		}
		else{
			// 看第 1 个和最后一个值
			if(a[i][1] == 0) dfs(i,1);
			if(a[i][n] == 0) dfs(i,n);
		}
	}
	for(int i = 1;i <= n;i++)	// 打印
	{
		for(int j = 1;j <= n;j++)
		{
			if(a[i][j] == 3) cout<<0<<" ";
			else if(a[i][j] == 0) cout<<2<<" ";
			else cout<<1<<" ";
		}
		cout<<endl;
	}
	return 0;
}
```

```cpp
#include <bits/stdc++.h>//5-1802-2   R_S
using namespace std;
int a[35][35];
int n;
void dfs(int x ,int y)
{
	a[x][y] = 2;
	int xx[5] = {0,0,1,0,-1};
	int yy[5] = {0,1,0,-1,0};
	for(int i=1;i<=4;i++)
	{
		int tx = x + xx[i];
		int ty = y + yy[i];
		if(tx>=1 && tx<=n-1 && ty>=1 && ty<=n-1 && a[tx][ty]==0)
		{
			dfs(tx,ty);
		}
	}
}
void p()
{
	for(int i=1;i<=n;i++)
	{
		for(int j=1;j<=n;j++)
		{
			cout <<  a[i][j] << ' ';
		}
		cout << endl;
	}
}
```

```
int main()
{
	cin >> n;
	for(int i=1;i<=n;i++)
	{
		for(int j=1;j<=n;j++)
		{
			cin >> a[i][j];
		}
	}

	// 开始找第一个为 1 的点
	for(int i=1;i<=n;i++)
	{
		for(int j=1;j<=n;j++)
		{
			if(a[i][j]==1)
			{
				dfs(i+1,j+1);
				p();
				return 0;
			}
		}
	}

	return 0;
}
```

```
/*
1、从第一个点开始找，找到第一个为 1 的点开始深搜，
    因为数字 1 构成圆环，那么最后一个遍历点会回到第一个 1 的位置。
    构成位置后，从倒数第一个点的右边开始填色，填色同样采用深搜的方式。
*/
#include<bits/stdc++.h>//5-1802-3    jiangyf70
using namespace std;
int n;
int a[40][40];
int fx[] = {0, 1, 0, -1};
int fy[] = {1, 0, -1, 0};

void dfs(int x, int y, int k)
{
    a[x][y] = k;
    for(int i = 0; i < 4; i++)
    {
        int dx = x + fx[i];
        int dy = y + fy[i];
        if(a[dx][dy] == 0 && dx >= 1 && dx <= n && dy >= 1 && dy <= n)
            dfs(dx, dy, k);
    }
}
```

```cpp
int main()
{
    cin >> n;
    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= n; j++)
        {
            cin >> a[i][j];
        }
    }

    for(int i = 1; i <= n; i++)
    {
        if(a[1][i] == 0) dfs(1, i, 3);
        if(a[i][1] == 0) dfs(i, 1, 3);
        if(a[n][i] == 0) dfs(n, i, 3);
        if(a[i][n] == 0) dfs(i, n, 3);
    }

    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= n; j++)
        {
            if(a[i][j] == 3) a[i][j] = 0;
            else if(a[i][j] == 0) a[i][j] = 2;
            cout << a[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

```cpp
#include<iostream>//5-1802-4  jiangyf70
using namespace std;
int n, a[30][30];
int fx[4] = {0, 1, 0, -1};
int fy[4] = {1, 0, -1, 0};
void dfs(int x, int y)
{
    a[x][y] = 3;
    for(int i = 0; i < 4; i++)
    {
        int dx = x + fx[i];
        int dy = y + fy[i];
        if(dx <= n && dx >= 1 && dy <= n && dy >= 1 && a[dx][dy] == 0)
            dfs(dx, dy);
    }
}
```

```cpp
int main()
{
    cin >> n;
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= n; j++)
            cin >> a[i][j];

    for(int i = 1; i <= n; i++)
    {
        if(i == 1 || i == n)
        {
            for(int j = 1; j <= n; j++)
            {
                if(a[i][j] == 0)dfs(i, j);
            }
        }
        else
        {
            if(a[i][1] == 0) dfs(i, 1);
            if(a[i][n] == 0) dfs(i, n);
        }
    }

    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= n; j++)
        {
            if(a[i][j] == 0) a[i][j] = 2;
            if(a[i][j] == 3) a[i][j] = 0;
            cout << a[i][j] <<" ";
        }
        cout << endl;
    }
    return 0;
}
```

```cpp
#include<iostream>//6-1383 anselxu
#include<cmath>
using namespace std;
char a[33][33];
int n,m;
void dfs(int x,int y)
{
	a[x][y]='.'// 改变值来记录已经搜索过的 #
	if(a[x+1][y]=='#') dfs(x+1,y);
	if(a[x-1][y]=='#') dfs(x-1,y);
	if(a[x][y+1]=='#') dfs(x,y+1);
	if(a[x][y-1]=='#') dfs(x,y-1);
	return;
}

int main()
{
	cin>>n>>m;
	int tot=0;
	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]=='#')
			{
				tot++;// 找到 #, 累计加 1
				dfs(i,j); 搜索相邻的 #
			}
		}
	cout<<tot;
	return 0;
}
```

1、读入数据，读入的同时记录 @ 的位置（下标）

2、从 @ 的位置出发，深搜，每搜过一个点，通过公共计数器 c 来统计搜索的次数

注意：为了防止死循环，走过一个点，就将该点从 . 改成 #

```
#include<bits/stdc++.h>//7-1897-1  javacn
using namespace std;
int n,m,s1,s2;//s1,s2 代表出发点的位置
char a[50][50];
int c = 0;
// 深搜从出发点开始能够访问到的点，并标记为 #
void dfs(int x,int y)
{
	a[x][y] = '#';// 标记走过，防止死循环
	c++;// 走过一个点计数一次
	// 尝试四方向
	// 不需要判断出边界，因为出了边界不可能是 .
	if(a[x][y+1]=='.') dfs(x,y+1);
	if(a[x+1][y]=='.') dfs(x+1,y);
	if(a[x][y-1]=='.') dfs(x,y-1);
	if(a[x-1][y]=='.') dfs(x-1,y);
}
```

```
int main()
{
	// 先读入列，再读入行
	cin>>m>>n;
	for(int i = 1;i <= n;i++)
	{
		for(int j = 1;j <= m;j++)
		{
			cin>>a[i][j];
			// 记录出发点的坐标
			if(a[i][j] == '@')
			{
				s1 = i;
				s2 = j;
			}
		}
	}
	dfs(s1,s2); // 从出发点开始搜索
	cout<<c;
	return 0;
}
```

```cpp
#include<iostream>//7-1897-2  bfs广搜    jiangyf70
using namespace std;

int n, m;
char a[30][30];
int r[1000][3];
int fx[] = {0, 1, 0, -1};
int fy[] = {1, 0, -1, 0};

int main()
{
    cin >> m >> n;
    int x, y;
    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < m; j++)
        {
            cin >> a[i][j];
            if(a[i][j] == '@')
            {
                x = i;
                y = j;
            }
        }
    }

    int head = 1, tail = 1;
    r[head][0] = x;
    r[head][1] = y;
    a[x][y] = '#';
```

```
    while(head <= tail)
    {
        for(int i = 0; i < 4; i++)
        {
            int dx = r[head][0] + fx[i];
            int dy = r[head][1] + fy[i];
            if(a[dx][dy] == '.')
            {
                tail++;
                r[tail][0] = dx;
                r[tail][1] = dy;
                a[dx][dy] = '#';

            }
        }
        head++;
    }
    cout << tail;

    return 0;
}
```

```cpp
#include <bits/stdc++.h>//8-1907-1    javacn 深搜解法
using namespace std;

int n,m,c;
char a[110][110];
int fx[5] = {0,0,1,0,-1};
int fy[5] = {0,1,0,-1,0};

void dfs(int x,int y)
{
	a[x][y] = '0';

	int tx,ty;
	for(int i = 1;i <= 4;i++)
	{
		tx = x + fx[i];
		ty = y + fy[i];

		if(a[tx][ty]>='1' && a[tx][ty]<='9')
		{
			dfs(tx,ty);
		}
	}
}
```

```
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]>='1' && a[i][j]<='9')
			{
				c++;
				dfs(i,j);
			}
		}
	}

	cout<<c;
	return 0;
}
```

```cpp
#include<bits/stdc++.h>//8-1907-2    bfs广搜    jiangyf70
using namespace std;
int n, m;
char a[110][110];
int  r[100100][3];
int fx[] = {0, 1, 0, -1};
int fy[] = {1, 0, -1, 0};

void bfs(int x, int y)
{
    memset(r, 0, sizeof(r));
    int head = 0, tail = 0;
    r[head][0] = x;
    r[head][1] = y;
    a[x][y] = '0';
    while(head <= tail)
    {
        for(int i = 0; i < 4; i++)
        {
            int dx = r[head][0] + fx[i];
            int dy = r[head][1] + fy[i];
            if(a[dx][dy] > '0' && a[dx][dy] <= '9')
            {
                tail++;
                r[tail][0] = dx;
                r[tail][1] = dy;
                a[dx][dy] =  ‘0’ ;
            }
        }
        head++;
    }

}
```

```
void print()
{
    for(int i = 0; i < n; i++) cout << a[i] << endl;
    cout << endl;
}

int main()
{
    cin >> n >> m;
    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < m; j++)
        {
            cin >> a[i][j];
        }
    }
    int cnt = 0;
    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < m; j++)
        {
            if(a[i][j] > '0' && a[i][j] <= '9')
            {
                cnt++;
                bfs(i, j);
                //print();
            }
        }
    }
    cout << cnt;
    return 0;
}
```

2、最少步数问题

1、准备一个整数数组，记录从出发点到每个点至少需要多少步，初始化为 INT_MAX；

2、从出发点开始探测，顺时针探测，如果该点可达，且到该点的步数更少，则替换 d 数组的步数；

3、最终 d 数组记录了到每个点至少需要多少步，d[n][m] 就是最终结果；

```
#include <bits/stdc++.h>//1-1432-1  javacn
using namespace std;
int n,m;
char a[50][50];// 地图
int d[50][50];// 存储走到每个点最少需要多少步
// 方向值变化的数组
int fx[5] = {0,0,1,0,-1};
int fy[5] = {0,1,0,-1,0};

// 递归探索地图，求到走到每个点最少需要多少步
void dfs(int x,int y,int dep)
{
	d[x][y] = dep;

	int tx,ty;
	// 循环数组，得到 4 个新的方向，递归探索 4 个方向
	for(int i = 1;i <= 4;i++)
	{
		tx = x + fx[i];
		ty = y + fy[i];

		// 如果 tx,ty 可以探索（该点在地图内，且该点是 .，且走到该点的步数更少）
		if(a[tx][ty]=='.'&&dep+1<d[tx][ty])
		{
			dfs(tx,ty,dep+1);
		}
	}
}
```

```
int main()
{
	int i,j;
	cin>>n>>m;
	for(i = 1;i <= n;i++)
	{
		for(j = 1;j <= m;j++)
		{
			cin>>a[i][j];
			// 将最少步数初始值设为 INT_MAX
			d[i][j] = INT_MAX;
		}
	}

	// 调用函数
	dfs(1,1,1);

	cout<<d[n][m];

}
```

题目数据范围要求 r,c<=40，实际时超过的，开始我提交开的数组 a 为 45，报运行错误。开到 55 提交就 A 了。

```
#include<iostream>//1-1432 -2   广搜解  anselxu
using namespace std;
struct bfs_arr{
	int x,y,t,pr;
};
bfs_arr b[5000];
char a[55][55];
int main()
{
	int dx[]={0,1,0,-1};
	int dy[]={1,0,-1,0};
	bool f=0;
	int n,m,h,l;
	cin>>n>>m;
	for(int i=1;i<=n;i++)
	{
		for(int j=1;j<=m;j++)
		{
			cin>>a[i][j];
		}
	}
	int head,tail;
	head=1;
	tail=1;
	b[head].x=1;
	b[head].y=1;
	b[head].t=1;
```

```
	while(head<=tail)
	{
		h=b[head].x;
		l=b[head].y;
		a[h][l]='#';
		for(int i=0;i<4;i++)
		{
			if(a[h+dx[i]][l+dy[i]]=='.')
			{
				a[h+dx[i]][l+dy[i]]='#';
				b[++tail].t=b[head].t+1;
				b[tail].x=h+dx[i];
				b[tail].y=l+dy[i];
				b[tail].pr=head;
				if(h+dx[i]==n&&l+dy[i]==m)
				{
					cout<<b[tail].t;
					f=1;
					break;
				}
			}
		}
		head++;
		if(f) break;
	}
	return 0;
}
```

最少步数问题，走到每个点的最小危险系数的和就是本题的最少步数：

```
#include <bits/stdc++.h>//2-1541      javacn
using namespace std;

int n,m;
int a[50][50];
// 走到每个点危险系数和的最小值
int d[50][50];
// 方向值变化的数组
int fx[5] = {0,0,1,0,-1};
int fy[5] = {0,1,0,-1,0};

// 深搜求走到每个点经过的危险系数和的最小值
void dfs(int x,int y,int sum)
{
	d[x][y] = sum;

	// 循环四方向
	int tx,ty;
	for(int i = 1;i <= 4;i++)
	{
		tx = x + fx[i];
		ty = y + fy[i];

		// 如果该点可行：走到该点的危险系数的和更小
		// 走到 tx,ty 点的危险系数 = 走到 x,y 点的危险系数 + tx,ty 点的危险系数
		if(sum+a[tx][ty]<d[tx][ty])
		{
			dfs(tx,ty,sum+a[tx][ty]);
		}
	}
}
```

```
int main()
{
	cin>>n>>m;
	for(int i = 1;i <= n;i++)
	{
		for(int j = 1;j <= m;j++)
		{
			cin>>a[i][j];
			d[i][j] = INT_MAX;
		}
	}

	dfs(1,1,a[1][1]);
	cout<<d[n][m];
	return 0;
}
```

```
include<bits/stdc++.h>//3-1433-1  走出迷宫的最少步数 2     Mihui0705
using namespace std;
int n,m,s1,s2,e1,e2;
char a[50][50];
int d[50][50];
int fx[4]={0,1,0,-1};
int fy[4]={1,0,-1,0};
void dfs(int x,int y,int k)
{

      d[x][y]=k;
      int tx,ty;
      for(int i=0;i<4;i++)
      {
            tx=x+fx[i];
            ty=y+fy[i];
            if((a[tx][ty]=='.'||a[tx][ty]=='T' )&& k+1<d[tx][ty])//  条件 1：是空地 条件 2：新获得的路径值比之前存储的最小值小（大的就不用更改了）
            {
                  dfs(tx,ty,k+1);//     每次往后面走都是前面一格最小值 +1（因为是通路）
            }
      }
}
```

```
int main()
{

	cin>>n>>m;
	for(int i=1;i<=n;i++)
	{
		for(int j=1;j<=m;j++)
		{
			cin>>a[i][j];
			if(a[i][j]=='S')
			{
				s1=i;
				s2=j;
			}
			if(a[i][j]=='T')
			{
				e1=i;
				e2=j;
			}
		}
	}
	for(int i=1;i<=n;i++)
	{
		for(int j=1;j<=m;j++)
		{
			d[i][j]=INT&#95;MAX;// 要更改存储最小值，有一个比较存在，就得让其初始值处于无限大的状态
		}
	}
	dfs(s1,s2,0);
	cout<<d[e1][e2];
	return 0;
}
```

```cpp
#include <iostream>//3-1433-2  走出迷宫的最少步数 2    kevinh
#include <climits>
using namespace std;

int n,m,r[110][110];
char a[110][110];
int bx,by,ex,ey;
int fx[4][2] = {{-1,0},{0,1},{1,0},{0,-1}};

void dfs(int x,int y,int dep)
{
    r[x][y]=dep;
    int tx,ty;
    for(int i=0;i<4;i++)
   {
        tx = x+fx[i][0];
        ty = y+fx[i][1];

        if(a[tx][ty]==' .' &&dep+1<r[tx][ty])
        {
            dfs(tx,ty,dep+1);
        }
    }
}
```

```
int main()
{
    int i,j;
    cin>>n>>m;
    for(i=1;i<=n;i++)
   {
        for(j=1;j<=m;j++)
       {
            cin>>a[i][j];
            r[i][j] = INT&#95;MAX;
            // 找出起点和出口，并设置为.
            if(a[i][j]=='S')
            {
                bx=i;
                by=j;
                a[i][j]='.';
            }
            else if(a[i][j]=='T')
            {
                ex=i;
                ey=j;
                a[i][j]='.';
            }

        }
    }
    dfs(bx,by,0);
    cout<<r[ex][ey];
    return 0;
}
```

```cpp
#include<bits/stdc++.h>//4-1900    jiangyf70
using namespace std;
char a[30][30];
int b[30][30];
int n, m;
int dx[4] = {0, 1, 0, -1};
int dy[4] = {1, 0, -1, 0};

void dfs (int x, int y, int dep)
{
    b[x][y] = dep;
    for(int i = 0; i < 4; i++)
    {
        int fx = x + dx[i];
        int fy = y + dy[i];
        if((a[fx][fy] == '.'|| a[fx][fy] =='*') && b[x][y] + 1 < b[fx][fy])
        {
            dfs(fx, fy, dep+1);
        }
    }
}
```

```
int main()
{
    int x1, x2, y1, y2;
    cin >> n >> m;
    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= m; j++)
        {
            cin >> a[i][j];
            b[i][j] = INT&#95;MAX;
            if(a[i][j] == '@')
            {
                x1 = i;
                y1 = j;
            }
            if(a[i][j] =='*')
            {
                x2 = i;
                y2 = j;
            }
        }

    }
    dfs(x1, y1, 1);
    if(b[x2][y2] < INT&#95;MAX)        cout << b[x2][y2] - 1; //不算最后一步。
    else cout << -1;


    return 0;
}
```

```
#include <bits/stdc++.h>//5-1901  javacn
using namespace std;
最少步数问题，关键信息如下：
1. 道路（@）、墙壁（#）、和守卫（x）骑士（r）、公主（a）
2. 移动一个位置，需要一个单位时间杀死守卫，需要额外多一个单位的时间
3. 不可能成功，输出”Impossible”
char a[30][30];// 迷宫
int d[30][30];// 最少步数
int n,m;
int s1,s2,e1,e2;// 起止点的坐标
int fx[5] = {0,0,1,0,-1};// 方向
int fy[5] = {0,1,0,-1,0};
void dfs(int x,int y,int sum)// 求走到每个点的最少时间
{
	d[x][y] = sum;
	// 尝试四方向
	int tx,ty;
	for(int i = 1;i <= 4;i++)
	{
		tx = x + fx[i];
		ty = y + fy[i];
		// 如果是@或者a，且时间更少（+1）
		if((a[tx][ty]=='@'||a[tx][ty]=='a') && sum+1<d[tx][ty])
		{
			dfs(tx,ty,sum+1);
			// 如果是x，且时间更少（+2）
		}
		else if(a[tx][ty]=='x'&&sum+2<d[tx][ty])
		{
			dfs(tx,ty,sum+2);
		}
	}
}
```

```
int main()
{
	// 读入、初始化
	cin>>n>>m;
	for(int i = 1;i <= n;i++)
	{
		for(int j = 1;j <= m;j++)
		{
			cin>>a[i][j];

			if(a[i][j] =='r')
			{
				s1 = i;
				s2 = j;
			}
			else if(a[i][j] =='a')
			{
				e1 = i;
				e2 = j;
			}

			d[i][j] = INT_MAX;
		}
	}

	dfs(s1,s2,0);

	// 如果能走到
	if(d[e1][e2]!=INT_MAX)		cout<<d[e1][e2];
	else cout<<"Impossible";
	return 0;
}
```

深搜求走到每个点的最少步数，注意本题要先读列，再读行：

```
#include <bits/stdc++.h>//6-1441-1     javacn
using namespace std;

//s1,s2:出发点，e1,e2:终点
int n,m,s1,s2,e1,e2;
char a[200][200];//迷宫
int d[200][200];//走到每个点的最少步数
//方向数组
int fx[10]={0,-2,-2,-1,1,2,2,1,-1};
int fy[10]={0,-1,1,2,2,1,-1,-2,-2};

//深搜求走到每个点的最少步数
void dfs(int x,int y,int step)
{
	d[x][y]=step;
	int tx,ty;
	for(int i=1;i<=8;i++)
	{
		tx=x+fx[i];
		ty=y+fy[i];
		if((a[tx][ty]=='.'||a[tx][ty]=='H')&&step+1<d[tx][ty])
		{
			dfs(tx,ty,step+1);
		}
	}
}
```

```
int main()
{
	// 本题先读列，再读行
	cin>>m>>n;
	for(int i=1;i<=n;i++)
	{
		for(int j=1;j<=m;j++)
		{
			cin>>a[i][j];
			d[i][j]=INT_MAX;
			if(a[i][j]=='K')
			{
				s1=i;
				s2=j;
			}
			else if(a[i][j]=='H')
			{
				e1=i;
				e2=j;
			}
		}
	}
	dfs(s1,s2,0);
	cout<<d[e1][e2];
}
```

```cpp
#include<iostream>//6-1441-2   广搜    anselxu
using namespace std;
struct pp{int a,b,t,pr;};
int dx[]={2,2,1,1,-2,-2,-1,-1};
int dy[]={1,-1,2,-2,1,-1,2,-2};
char b[180][180];
pp a[40001];
int main()
{
	int n,m,x,y;
	cin>>m>>n;
	for(int i=1;i<=n;i++)
	{
		for(int j=1;j<=m;j++)
		{
			cin>>b[i][j];
			if(b[i][j]=='K')
			{
				x=i;
				y=j;
			}
		}
	}
	int head,tail;
	head=1;
	tail=1;
	a[head].a=x;
	a[head].b=y;
	a[head].t=0;
	b[x][y]='*';
```

```
        while(head<=tail)
        {
                int h,l;
                x=a[head].a;
                y=a[head].b;
                for(int i=0;i<8;i++)
                {
                        h=x+dx[i];
                        l=y+dy[i];
                        if(h<=n&&h>0&&l<=m&&l>0&&b[h][l]!='*')
                        {
                                tail++;
                                a[tail].a=h;
                                a[tail].b=l;
                                a[tail].t=a[head].t+1;
                                a[tail].pr=head;
                                if(b[h][l]==' H' )
                                {
                                        cout<<a[tail].t;
                                        return 0;
                                }
                                b[h][l]='*';
                        }
                }
                head++;
        }
        return 0;
}
```

```cpp
#include <bits/stdc++.h>//6-1441-3  jiangyf70
using namespace std;
char a[155][155];
int b[155][155];
int fx[8] = {-2, -2, -1, -1, 1, 1, 2, 2};
int fy[8] = {1, -1, 2, -2, 2, -2, 1, -1};
int n, m;
void dfs(int x, int y, int k)
{
    b[x][y] = k;
    for(int i = 0; i < 8; i++)
    {
        int dx = x + fx[i];
        int dy = y + fy[i];
        if((a[dx][dy] =='.'|| a[dx][dy] =='H') && k + 1 < b[dx][dy])
            dfs(dx, dy, k+1);
    }
}
int main()
{
    int x1, y1, x2, y2;
    cin >> n >> m;                  //注意 n, m 的输入是颠倒的
    for(int i = 1; i <= m; i++)
    {
        for(int j = 1; j <= n; j++)
        {   cin >> a[i][j];
            b[i][j] = INT_MAX;
            if(a[i][j] == 'H')x2 = i, y2 = j;
            if(a[i][j] == 'K')x1 = i, y1 = j;
        }
    }
    dfs(x1, y1, 0);
    cout << b[x2][y2];
    return 0;
}
```

3、回溯与路径打印

重点：将路径数组的下标 k 作为递归的参数，这样在递归后退时，k 的值会跟着后退，从而覆盖之前的路径点。

解法一：提供三个递归参数

dfs(x,y,k)，根据当前的点坐标 x,y 计算下一个点的坐标。

```
#include <bits/stdc++.h>//1-1431-1  javacn
using namespace std;
// 左、上、右、下顺序进行搜索
char a[30][30];
int r[410][3];// 记录正确的第一条路径
int n;
// 方向的变化
int fx[5] = {0,0,-1,0,1};
int fy[5] = {0,-1,0,1,0};

// 打印 r 数组中存储的第一条路线
void print(int k)
{
	for(int i = 1;i <= k;i++)
	{
		cout<<"("<<r[i][1]<<","<<r[i][2]<<")";
		// 如果不是最后一个点，打印连接的 ->
		if(i != k)
		{
			cout<<"->";
		}
	}

	exit(0);// 停止程序
}
```

```
// 向路径数组下标为 k 的位置，记录一个坐标 x,y
void dfs(int x,int y,int k)
{
	// 记录探索到的点的坐标
	r[k][1] = x;
	r[k][2] = y;
	// 将走过的点，标记为 1，防止死循环
	a[x][y] = '1';

	// 判断 xy 如果是终点，则打印路径
	if(x == n && y == n)
	{
		print(k);
	}

	int tx,ty;
	// 探索四个方向
	for(int i = 1;i <= 4;i++)
	{
		tx = x + fx[i];
		ty = y + fy[i];

		// 如果该点可达（没走过，且不是障碍，且没有出迷宫）
		if(a[tx][ty] == '0')
		{
			// 向 r 数组的下标为 k+1 那一行，记录 tx,ty 点
			dfs(tx,ty,k+1);
		}
	}
}
```

```
int main()
{
	int i,j;
	cin>>n;
	for(i = 1;i <= n;i++)
	{
		for(j = 1;j <= n;j++)
		{
			cin>>a[i][j];
		}
	}

	// 调用函数，探索出第一条路径
	// 向 r 数组的下标为 1 的那一行，记录 1,1 点
	dfs(1,1,1);
}
```

解法二：通过数组记录走过的点，从而计算下一个点。

```
#include <iostream>//1-1431-2  javacn
using namespace std;
int n;
int r[400][3];// 记录路径
char s[30][30];// 字符数组，记录迷宫
int fx[5] = {0,0,-1,0,1};
int fy[5] = {0,-1,0,1,0};
// 打印路径
void show(int k)
{
	int i;
	for(i = 1; i <= k; i++)
	{
		cout<<"("<<r[i][1]<<","<<r[i][2]<<")";
		// 如果不是最后一个点，加 ->
		if(i != k)
		{
			cout<<"->";
		}
		else
		{
			cout<<endl;
		}
	}
}
```

```
//x y 表示要探测的点，k 表示要存储的路径数组的下标
void fun(int k)
{
	int tx,ty;
	for(int i = 1; i <= 4; i++)
	{
		tx = r[k-1][1] + fx[i];
		ty = r[k-1][2] + fy[i];

		if(tx >= 1 && tx <= n && ty >= 1 && ty <= n && s[tx][ty] == '0')
		{
			r[k][1] = tx;
			r[k][2] = ty;
			s[tx][ty]='1';
			if(tx==n&&ty==n) {	show(k);	}
			else		 {	fun(k+1);	}
		}
	}
}
int main() {
	cin>>n;
	int i,j;
	for(i = 1; i <= n; i++)
	{
		for(j = 1; j <= n; j++)
		{
			cin>>s[i][j];
		}
	}
	// 从 1,1 点开始探测，从下标为 1 开始记录路径
	r[1][1] = 1;
	r[1][2] = 1;
	s[1][1] = '1';// 标记出发点走过
	fun(2);
}
```

解法一：参照迷宫的第一条路，深搜出迷宫的所有路径

```
#include <bits/stdc++.h>//2-1360-1    javacn
using namespace std;
// 只能向下或者向右走：优先向下，其次向右
int n,m;
int r[20][3];// 存储行走路径
// 方向的变化
int fx[3] = {0,1,0};
int fy[3] = {0,0,1};
int c;// 计数器

void print(int k)
{
	c++;
	cout<<c<<":";
	// 除了最后一个点以外
	for(int i = 1;i < k;i++)
	{
		cout<<r[i][1]<<","<<r[i][2]<<"->";
	}
	cout<<n<<","<<m<<endl;
}
```

```
// 向 r 数组下标为 k 的那一行，记录 x,y 点
void dfs(int x,int y,int k)
{
	// 记录坐标
	r[k][1] = x;
	r[k][2] = y;

	// 如果走到了终点，打印路径
	if(x == n && y == m)
	{
		print(k);
		// 停止递归函数，到了终点打印，就不需要继续递归了
		return;
	}

	int tx,ty;
	for(int i = 1;i <= 2;i++)
	{
		tx = x + fx[i];
		ty = y + fy[i];

		// 判断 tx,ty 有效
		if(tx>=1&&tx<=n&&ty>=1&&ty<=m)
		{
			dfs(tx,ty,k+1);
		}
	}
}

int main()
{
	cin>>n>>m;
	// 向 r 数组下标为 1 的那一行，记录 1,1 点
	dfs(1,1,1);
}
```

解法二：利用 r 数组存储的上一个点的坐标，求解下一个点的坐标

```
#include <bits/stdc++.h>//2-1360-2     javacn
using namespace std;
// 只能向下或者向右走：优先向下，其次向右
int n,m;
int r[20][3];// 存储行走路径
// 方向的变化
int fx[3] = {0,1,0};
int fy[3] = {0,0,1};
int c;// 计数器

void print(int k)
{
	c++;
	cout<<c<<":";
	// 除了最后一个点以外
	for(int i = 1;i < k;i++)
	{
		cout<<r[i][1]<<","<<r[i][2]<<"->";
	}
	cout<<n<<","<<m<<endl;
}
```

```
// 向 r 数组下标为 k 的那一行存经过的点的坐标
void dfs(int k)
{
	int tx,ty;
	for(int i = 1;i <= 2;i++)
	{
		// 得到新点坐标
		tx = r[k-1][1] + fx[i];
		ty = r[k-1][2] + fy[i];
		if(tx>=1&&tx<=n&&ty>=1&&ty<=m)		// 判断 tx,ty 有效
		{
			// 存储 tx,ty
			r[k][1] = tx;
			r[k][2] = ty;

			if(tx == n && ty == m)  // 如果到了终点，则打印，否则递归
			{
				print(k);
			}
			else
			{
				dfs(k+1);
			}
		}
	}
}

int main(){
	cin>>n>>m;
	// 第 1 个点的坐标直接存入
	r[1][1] = 1;
	r[1][2] = 1;
	// 向 r 数组下标为 2 的那一行存坐标
	dfs(2);
}
```

思路：深搜，从 0,0 点出发，按题目要求尝试四个方向，哪个方向能去就将坐标存入结果数组。

注意：本题 0,0 点在左下角，向上 x 会变小。

解法一：将当前点的坐标 x,y 传入递归函数，利用当前坐标 x,y 的值计算可能跳转到的 4 个坐标值。

```
#include <bits/stdc++.h>//3-1362-1    javacn
using namespace std;
//a 存储所有的路径，c 存储路径数量
int a[100][3],c=0;
// 四种移动规则
int fx[5]= {0,2,1,-1,-2},fy[5]= {0,1,2,2,1};

// 输出结果
void print(int k)
{
	c++;
	cout<<c<<":";
	// 最后一个点在循环结束打印
	for(int i=1; i<k; i++)
	{
		cout<<a[i][1]<<","<<a[i][2]<<"->";
	}
	cout<<"4,8"<<endl;
}
```

```
// 深搜找所有路径
void dfs(int x,int y,int k)
{
	a[k][1] = x;
	a[k][2] = y;
	if(x == 4 && y == 8)
	{
		print(k);
		// 到了终点不需要继续递归，开始后退尝试其他路径
		return;
	}

	int tx,ty;
	// 往 4 个方向跳
	for (int i=1; i<=4; i++)
	{
		// 跳的新点，从上当前点开始加上方向值的变化，得到 4 个新点
		tx = x+fx[i];
		ty = y+fy[i];
		// 判断马不越界
		if (tx>=0&&tx<=4&&ty>=0&&ty<=8)
		{
			dfs(tx,ty,k+1);
		}
	}
}

int main()
{
	// 从 0,0 点开始递归，存入数组的下标为 1 的那行
	dfs(0,0,1);
}
```

解法二：先将 0,0 点存储，然后根据 a 数组的上一个点计算下一个点可能的 4 个坐标值。

```
#include <bits/stdc++.h>//3-1362-2  javacn
using namespace std;
//a 存储所有的路径，c 存储路径数量
int a[100][3],c=0;
// 四种移动规则
int fx[5]= {0,2,1,-1,-2},fy[5]= {0,1,2,2,1};

// 输出结果
void print(int k)
{
	c++;
	cout<<c<<":";
	// 最后一个点在循环结束打印
	for(int i=1; i<k; i++)
	{
		cout<<a[i][1]<<","<<a[i][2]<<"->";
	}
	cout<<"4,8"<<endl;
}
```

```
// 深搜找所有路径
void dfs(int k)
{
	int tx,ty;
	// 往 4 个方向跳
	for (int i=1; i<=4; i++)
	{
		// 跳的新点，从上 1 个点开始加上方向值的变化，得到 4 个新点
		tx = a[k-1][1]+fx[i];
		ty = a[k-1][2]+fy[i];
		// 判断马不越界
		if (tx>=0&&tx<=4&&ty>=0&&ty<=8)
		{
			// 保存当前马的位置
			a[k][1]=tx;
			a[k][2]=ty;
			// 如果到了终点
			if (tx==4&&ty==8)
			{
				print(k);
			}
			else
			{
				dfs(k+1);// 搜索下一步
			}
		}
	}
}
int main()
{
	a[1][1]=0;
	a[1][2]=0;
	// 从坐标 (0,0) 开始往右跳第二步
	dfs(2);
}
```

```cpp
#include<iostream>//3-1362-3  jiangyf70
using namespace std;
bool a[5][9]; //标记坐标是否走过
int b[50][2]; //保存步骤坐标
int c= 0;
int fx[4] = {2, 1, -1, -2};
int fy[4] = {1, 2, 2, 1};
void print(int k)
{
    c++;
    printf("%d:", c);
    for(int i = 1; i < k; i++)
    {
        printf("%d,%d->", b[i][0], b[i][1]);
    }
    printf("%d,%d\n", b[k][0], b[k][1]);
}
```

```
void dfs(int x, int y, int k)
{
    b[k][0] = x;
    b[k][1] = y;
    for(int i = 0; i < 4; i++)
    {
        int dx = x + fx[i];
        int dy = y + fy[i];
        if(dx >= 0 && dx <= 4 && dy >= 0 && dy <= 8 && a[dx][dy] == false)
        {
            b[k+1][0] = dx;
            b[k+1][1] = dy;
            a[dx][dy] = true;
            if(dx == 4 && dy == 8)
            {
                print(k+1);
            }
            else
            {
                dfs(dx, dy, k+1);
            }
            a[dx][dy] = false;
        }
    }
}


int main()
{
    a[0][0] = true;
    dfs(0, 0, 1);
    return 0;
}
```

本题提供三种参考解法，注意看解法三和前两种解法的差异！

解法一： 用正在走的点 xy 计算出下一个点，标记将要去的点

```
#include <bits/stdc++.h>//4-1739-1  javacn
using namespace std;
// 右、下、左、上
/*
  思路：沿着右、下、左、上的顺序深度优先搜索，走过的点标记为 true
        递归其他方向，递归结束，后退到上一步，撤销标记，回溯到前一个状态
*/
int n,c;
int r[30][3];// 存储路径
bool f[10][10];// 标记点是否走过
int fx[5] = {0,0,1,0,-1};
int fy[5] = {0,1,0,-1,0};

// 输出路径
void print(int k)
{
      c++;
      cout<<c<<":";
      for(int i = 1;i < k;i++)
      {
            cout<<r[i][1]<<","<<r[i][2]<<"->";
      }
      cout<<n<<","<<n<<endl;
}
```

```
// 深搜存储路径：将 xy 点存储到 r 数组下标为 k 的位置
void dfs(int x,int y,int k){
	r[k][1] = x;
	r[k][2] = y;
	// 如果到了终点，打印
	if(x == n && y == n)
	{
		print(k);
		return;
	}
	// 尝试不同的方向
	int tx,ty;
	for(int i = 1;i <= 4;i++)
	{
		tx = x + fx[i];
		ty = y + fy[i];

		// 判断新的方向可达，且没有访问过
		if(tx>=1&&tx<=n&&ty>=1&&ty<=n&&f[tx][ty]==false)
		{
			// 标记 tx，ty 点走过了
			f[tx][ty] = true;
			dfs(tx,ty,k+1);
			// 递归结束，后退，回溯到前一个状态
			f[tx][ty] = false;
		}
	}
}
int main(){
	cin>>n;
	// 从 1,1 点开始递归，记录到 r 数组下标为 1 的那一行
	f[1][1] = true;// 标记 1,1 点走路
	dfs(1,1,1);
	return 0;
}
```

解法二：用数组中存储的上一个点，计算下一个点，标记将要去的点

```
#include <bits/stdc++.h>//4-1739-2  javacn
using namespace std;
// 右、下、左、上
/*
  思路：沿着右、下、左、上的顺序深度优先搜索，走过的点标记为 true
        递归其他方向，递归结束，后退到上一步，撤销标记，回溯到前一个状态
*/
int n,c;
int r[30][3];// 存储路径
bool f[10][10];// 标记点是否走过
int fx[5] = {0,0,1,0,-1};
int fy[5] = {0,1,0,-1,0};

// 输出路径
void print(int k)
{
      c++;
      cout<<c<<":";
      for(int i = 1;i < k;i++){
            cout<<r[i][1]<<","<<r[i][2]<<"->";
      }
      cout<<n<<","<<n<<endl;
}
```

```
// 深搜存储路径：将 xy 点存储到 r 数组下标为 k 的位置
void dfs(int k){
	// 尝试不同的方向
	int tx,ty;
	for(int i = 1;i <= 4;i++)
	{
		tx = r[k-1][1] + fx[i];
		ty = r[k-1][2] + fy[i];
		// 判断新的方向可达，且没有访问过
		if(tx>=1&&tx<=n&&ty>=1&&ty<=n&&f[tx][ty]==false)
		{
			// 存储路径
			r[k][1] = tx;
			r[k][2] = ty;

			// 标记 tx，ty 点走过
			f[tx][ty] = true;
			// 如果到了终点
			if(tx == n && ty == n)  {	print(k);	}
			else                    {	dfs(k+1);	}
			// 递归结束，回溯到前一个状态
			f[tx][ty] = false;
		}
	}
}
int main(){
	cin>>n;
	// 存储第 1 个点
	r[1][1] = 1;
	r[1][2] = 1;
	f[1][1] = true;// 标记 1,1 点走路
	// 从 r 数组下标为 2 的那一行开始存储路径
	dfs(2);
	return 0;
}
```

解法三：标记正在访问的点

```
#include <bits/stdc++.h>//4-1739-3  javacn
using namespace std;
int a[50][3];// 存放所有走过的路径
bool f[50][50];// 标记某个点是否走过
int n,c;//n 行，m 列的棋盘

//fx 代表 x 坐标可能的变化
int fx[5] = {0,0,1,0,-1};
//fy 代表 y 坐标可能的变化
int fy[5] = {0,1,0,-1,0};

//x 代表 a 数组中元素的个数
void print(int x)
{
	c++;
	cout<<c<<":";
	int i;
	for(i = 1;i <= x;i++)
	{
	  // 如果不是最后一个解
		if(i != x)
		{
			cout<<a[i][1]<<","<<a[i][2]<<"->";
		}
		else
		{
			cout<<a[i][1]<<","<<a[i][2]<<endl;
		}
	}

}
```

```
void dfs(int x,int y,int k)
{
	a[k][1] = x;
	a[k][2] = y;

	if(x == n && y == n)
	{
		print(k);
		return;
	}

	// 标记当前正在走的 xy
	f[x][y] = true;

	int tx,ty;
	for(int i = 1;i <= 4;i++)
	{
		tx = x + fx[i];
		ty = y + fy[i];

		if(tx>=1&&tx<=n&&ty>=1&&ty<=n&&f[tx][ty]==false)
		{
			dfs(tx,ty,k+1);
		}
	}

    // 当前点 x,y 的四方向讨论结束，回溯时，撤销标记
	f[x][y] = false;
}

int main()
{
	cin>>n;
	dfs(1,1,1);
}
```

```
#include<iostream>//5-1411-1  迷宫的路径？ anselxu
#include<cstring>
using namespace std;
int n,mn,t=0,a[160][2];
char m[19][19];
int dx[]={0,1,0,-1};
int dy[]={1,0,-1,0};
void p(int c){
	printf("%d:",t);
	for(int i=1;i<=c;i++){
		printf("%d,%d->",a[i][0],a[i][1]);
	}
	printf("%d,%d\n",n,mn);
}
void dfs(int x,int y,int c){
	if(x==n&&y==mn){
		t++;// 到达终点，增加一个路径
		p(c-1);
		return;
	}
	m[x][y]='#';// 到达某点，标记
	a[c][0]=x;
	a[c][1]=y;
	int h,l;
	for(int i=0;i<4;i++)
	{
		h=x+dx[i];
		l=y+dy[i];
		if(h>0&&h<=n&&l>0&&l<=mn&&m[h][l]=='o')
		{
			dfs(h,l,c+1);
		}
	}
	m[x][y]='o';// 回溯
}
```

```
int main()
{
	cin>>n>>mn;
	for(int i=1;i<=n;i++)
	{
		for(int j=1;j<=mn;j++)
		{
			cin>>m[i][j];
		}
	}
	dfs(1,1,1);
	if(t==0)
	cout<<"no";
	return 0;
}
```

```cpp
#include <iostream>//5-1411-2  迷宫的路径？  kevinh
using namespace std;

int n,m,c;
char a[15][15];
int r[15*15][2];
int f[15][15];
int fx[4][2] = {{0,1},{1,0},{0,-1},{-1,0}};

void print(int k)
{
	c++;
	cout<<c<<":";
	for(int i=1;i<=k;i++)
	{
		cout<<r[i][0]<<","<<r[i][1];
		if(i!=k)
		{
			cout<<"->";
		 }
	}
    cout<<endl;
}
```

```
void dfs(int x,int y,int k){
	r[k][0] = x;
	r[k][1] = y;
	if(x==n&&y==m)
	{
	  print(k);
	  return;
	}
	int tx,ty;
	for(int i=0;i<4;i++)
	{
		tx = x+fx[i][0];
		ty = y+fx[i][1];

		if(a[tx][ty]=='o'&&f[tx][ty]==false)
		{
			f[tx][ty] = true;
			dfs(tx,ty,k+1);
			f[tx][ty] = false;
		}
    }
}

int main() {
	int i,j;
	cin>>n>>m;
	for(i=1;i<=n;i++)
	{
		for(j=1;j<=m;j++)	{	cin>>a[i][j];	}
	}
    f[1][1] = true;
    dfs(1,1,1);
    if(c==0)	{	cout<<"no"<<endl;	}
    return 0;
}
```

4、回溯与全排列

```
#include<iostream>//1-1654   jiangyf70
using namespace std;
int n, a[10];
bool b[10];

void print()
{
    for(int i = 1; i <= n; i++)    {    cout << a[i];   }
    cout << endl;
}

void dfs(int k) //将a[k]赋值
{
    for(int i = 1; i <= n; i++)
    {
        if(b[i] == false)
        {
            a[k] = i;
          //  b[i] = true;
            if(k == n)      {      print();     }
            else            {      dfs(k+1);    }
            //b[i] = false;
        }
    }
}

int main()
{
    cin >> n;
    dfs(1);
    return 0;
}
```

全排列基础题。

```
#include <bits/stdc++.h>//2-1308    javacn
using namespace std;
int n;
int a[10];// 存放全排列的结果
bool f[10];// 标记哪些数使用过
void print(){
	for(int i = 1;i <= n;i++)
	{
		cout<<a[i];
		if(i != n)  {     cout<<" ";        }
		else        {     cout<<endl;       }
	}
}
void fun(int k)// 递归函数：为 a 数组每个元素赋值
{
	for(int i = 1;i <= n;i++)
	{
		// 如果 i 这个数没有被用过，则填写到下标为 k 的位置
		if(f[i] == false)
		{
			a[k] = i;
			f[i] = true; // 标记数字 i 被选用了
			// 如果 a 中存储了 n 个元素，输出结果，否则递归为 k+1 的下标赋值
			if(k == n)  {     print();    }
			else        {     fun(k+1);   }
			f[i] = false; // 回溯到前一个状态，标记 i 没有被用过
		}
	}
}
int main(){
	cin>>n;
	// 为 a 数组的下标为 1 的位置赋值
	fun(1);
}
```

```cpp
#include<iostream>//3-1358    jiangyf70
using namespace std;
int n, c = 0;
int a[11];
bool b[11];
bool prime(int n)
{
    if(n == 1) return 0;
    for(int i = 2; i * i <= n; i++)
    {
        if(n % i == 0)        return 0;
    }
    return 1;
}

void print()
{
    cout << ++c << ':';
    for(int i = 1; i <= n; i++)
    {
        cout << a[i] << " ";
    }
    cout << endl;
}
```

```
void dfs(int k)
{
    for(int i = 1; i <= n; i++)
    {
        if(b[i] == false)
        {
            if(k == 1 || prime(a[k-1] + i))
            {
                a[k] = i;
                if(k == n && prime(a[k] + a[1]))
                {
                    print();
                }
                else
                {
                    b[i] = true;
                    dfs(k+1);
                }
                b[i] = false;
            }
        }
    }
}


int main()
{
    cin >> n;
    dfs(1);
    cout << "total:" << c;
    return 0;
}
```

```cpp
#include<iostream>//4-1361    jiangyf70
using namespace std;
int n, m;
int a[10];
bool b[10];
void print()
{
    for(int i = 1; i <= m; i++)     {         cout << a[i] <<"    ";    }
    cout << endl;
}

void dfs(int k)
{
    for(int i = 1; i <= n; i++)
    {
        if(b[i] == false && k <= m)
        {
            a[k] = i;
            if(k == m)            {    print();            }
            else
            {
                b[i] = true;
                dfs(k+1);
            }
            b[i] = false;
        }
    }
}

int main()
{
    cin >> n >> m;
    dfs(1);
    return 0;
}
```

```cpp
#include<iostream>//5-1685    jiangyf70
using namespace std;
int n;
int a[10]; // 原始数字
bool b[10]; // 标志是否用过
int r[10]; // 存放结果
void print()
{
    for(int i = 1; i <= n; i++)    {        cout << r[i] <<"   ";    }
    cout << endl;
}
void dfs(int k)
{
    for(int i = 1; i <= 9; i++)
    {
        if(a[i] != 0 && b[i] == false && k <= n )
        {
            r[k] = i; // 将该下标写入第 k 个值
            b[i] = true;
            if(k == n)   print();
            else        {      dfs(k+1);     }
            b[i] = false;
        }
    }
}
int main()
{
    cin >> n;
    int x;
    for(int i = 1; i <= n; i++)
    {   cin >> x;
        a[x]++;  // 用桶的办法，说明该下标有一个值，从小到大字典序看，
    }
    dfs(1);
}
```

解题要求：

1. 第 1 个数必须是 1
2. 素数环：相邻两个数的和是素数
3. 如果超过 10 个解，只要输出前 10 个解

解题思路：

1. 先输出第 1 个数为 1 的全排列
2. 加条件使得结果满足环中，任意相邻的 2 个数的和是素数
3. 判断，如果超过 10 组解，只输出前 10 组解

特别注意：如果 n 是奇数，没有解；因为如果要相邻 2 数的和是素数，必须一个奇数和一个偶数间隔排开。

如果 n 是奇数，会导致首尾两个数的和是偶数。

```
#include <bits/stdc++.h>//6-1439-1  javacn
using namespace std;
int a[30];// 存储素数环
bool f[30];// 标记某个数是否被使用
int n,cnt = 0;// 统计素数环的数量
// 判断素数
bool prime(int n){
	if(n <= 1) return false;
	for(int i = 2;i <= sqrt(n);i++)
	{
		if(n % i == 0) return false;
	}
	return true;
}
// 输出素数环
void print(){
	for(int i = 1;i <= n;i++)
	{
		cout<<a[i]<<" ";
	}
	cout<<endl;
	cnt++;
	if(cnt == 10) exit(0);
}
```

```
// 为 a 数组下标为 k 的位置填值
void dfs(int k)
{
	// 循环所有的可能
	for(int i = 1;i <= n;i++)
	{
		// 如果 i 没有被用过，且和上一个数的和是素数
		if(f[i]==false&&prime(i+a[k-1]))
		{
			f[i] = true;// 标记用过
			a[k] = i;
			// 判断是否填满 n 个数，且首尾的和是素数
			if(k==n&&prime(a[n]+a[1])) print();// 输出素数环的结果
			else dfs(k+1);// 填下一个位置
			f[i] = false;// 回溯：撤销对于 i 这个数的占用
		}
	}
}

int main()
{
	cin>>n;
	// 如果 n 是奇数，没有解
	// 因为如果要相邻 2 数的和是素数，必须一个奇数和一个偶数间隔排开
	// 如果 n 是奇数，会导致首尾两个数的和是偶数
	if(n % 2 == 1) return 0;

	a[1] = 1;
	f[1] = true;//1 标记占用
	dfs(2);// 从 a 数组的第 2 个位置开始填值

	return 0;
}
```

```cpp
#include<iostream>//6-1439-2  jiangyf70
using namespace std;
int n, c = 0;
int r[25];
bool b[25];

bool prime(int n)
{
    if(n <= 1) return 0;
    for(int i = 2; i * i <= n; i++)
    {
        if(n % i == 0) return 0;
    }
    return 1;
}

void print()
{
    for(int i = 1; i <= n; i++)
    {
        cout << r[i] << " ";
    }
    cout << endl;
    c++;
    if(c == 10)exit(0);
}
```

```
void dfs(int k)
{
    for(int i = 1; i <= n; i++)
    {

        if(b[i] == false && k <= n && prime(i + r[k-1]))
        {
            r[k] = i;
            b[i] = true;
            if(k == n && prime(r[n] + r[1]) && c <= 10) print();
            else dfs(k+1);
            b[i] = false;
        }
    }
}

int main()
{
    cin >> n;
    if(n % 2 == 1) return 0;  //不加这句会超时
    r[1] = 1;
    b[1] = true;
    dfs(2);
    return 0;
}
```

思路：第 1 个单词确定的情况下，剩余的单词全排列，求能接龙的最长长度

解法一：使用 r 数组存储接龙的结果，求能够为 r 数组赋值的最长长度（其实没必要存储接龙的结果，参考解法二）

```
#include<bits/stdc++.h>//7-1590-1   javacn
using namespace std;

/*
思路：第 1 个单词确定的情况下，剩余的单词全排列，求能接龙的最长长度
*/
const int N = 60;
string a[N];// 读入的字符串
string r[N];// 表示接龙的结果
bool f[N];// 标记第 i 个字符串是否被使用
int n,ma = 1;

// 为 r 数组第 k 个位置填字符串
void dfs(int k)
{
	for(int i = 2;i <= n;i++)
	{
		// 第 i 个单词没有被使用，且该单词的词头 == 上一个单词的词尾
		if(!f[i]&&a[i][0]==r[k-1][1])
		{
			f[i] = true;// 标记占用
			r[k] = a[i];
			ma = max(ma,k);

			// 如果没填满，尝试填写下一位置
			if(k<n) dfs(k+1);
			f[i] = false;// 回溯，撤销标记
		}
	}
}
```

```
int main()
{
	cin>>n;
	for(int i = 1;i <= n;i++)
	{
		cin>>a[i];
	}

	r[1] = a[1];// 第 1 个单词确定
	f[1] = true;
	dfs(2);// 从第 2 个单词开始填写
	cout<<ma;
    return 0;
}
```

解法二：递归时将上一个单词的词尾作为递归的参数

```
#include<bits/stdc++.h>//7-1590-2  javacn
using namespace std;
// 思路：第 1 个单词确定的情况下，剩余的单词全排列，求能接龙的最长长度
const int N = 60;
string a[N];// 读入的字符串
bool f[N];// 标记第 i 个字符串是否被使用
int n,ma = 1;
// 讨论第 k 个位置可以放哪个单词
//char tail：上一个单词的词尾
void dfs(int k,char tail)
{
	for(int i = 2;i <= n;i++)
	{
		// 第 i 个单词没有被使用，且该单词的词头 == 上一个单词的词尾
		if(!f[i]&&a[i][0]==tail)
		{
			f[i] = true;// 标记占用
			ma = max(ma,k);

			// 如果没填满，尝试填写下一位置
			dfs(k+1,a[i][1]);
			f[i] = false;// 回溯，撤销标记
		}
	}
}
int main(){
	cin>>n;
	for(int i = 1;i <= n;i++)	{	cin>>a[i];	}

	f[1] = true;
	dfs(2,a[1][1]);// 从第 2 个单词开始填写
	cout<<ma;
    return 0;
}
```

```cpp
#include<iostream>//7-1590-3    jiangyf70
#include<cstring>
using namespace std;
char a[60][5];
bool f[60];
char r[60][5];
int l = 1, n;  //l 初始长度要为 1 啊，调试了很久。

void dfs(int k)
{
    for(int i = 2; i <= n; i++)
    {
        if(f[i] == false && a[i][0] == r[k-1][1])
        {
            f[i] = true;
            strcpy(r[k],a[i]);
            l = max(k, l);
            dfs(k+1);
            f[i] = false;
        }
    }
}

int main()
{
    cin >> n;
    for(int i = 1; i <=n; i++)    {       cin >> a[i];    }
    f[1] = true;
    strcpy(r[1],a[1]);
    dfs(2);
    cout << l;
    return 0;
}
```

生成 L 位密码：

1. 至少有一个元音，至少两个辅音
2. 按字母表的顺序
3. 如果结果超过 25000 个，只需要前 25000 个

读入 n 位小写字母，从中选取若干位，构成 L 位的密码

思路：求字母的组合；

解法一：组合的写法

```
#include <bits/stdc++.h>//8-1823-1    javacn
using namespace std;

char a[30],r[30];
bool f[30];
int len,c,ans = 0;

// 输出
void print()
{
	// 检验是否包含至少 1 个元音 2 个辅音
	int cnt = 0;
	for(int i = 1;i <= len;i++)
	{
		if(r[i]=='a'||r[i]=='e'||r[i]=='i'||r[i]=='o'||r[i]=='u') cnt++;
	}

	// 有至少 1 个元音，2 个辅音
	if(cnt>=1&&len-cnt>=2)
	{
		ans++;
		for(int i = 1;i <= len;i++) cout<<r[i];
		cout<<endl;

		if(ans == 25000)		exit(0);
	}
}
```

```
// 从 c 个字符中搜索出 len 个满足条件的字符
void dfs(int x,int cnt)
{
	if(cnt == len)
	{
		print();
		return;
	}
	if(x > c) return;

	r[cnt+1] = a[x];
	dfs(x+1,cnt+1);
	dfs(x+1,cnt);
}

int main()
{
	cin>>len>>c;
	for(int i = 1;i <= c;i++)
	{
		cin>>a[i];
	}

	sort(a+1,a+c+1);// 排序

	// 从第 1 个位置开始填字符
	dfs(1,0);
	return 0;
}
```

解法二：改写排列，效率不如解法一

```
#include <bits/stdc++.h>//8-1823-2     javacn
using namespace std;

char a[30],r[30];
bool f[30];
int len,n,cnt = 0;//cnt 统计结果的数量

// 输出
void print()
{
    // 检验是否有至少一个元音 2 个辅音
    int c = 0;
    for(int i = 1;i <= len;i++)
    {
        if(r[i]=='a'||r[i]=='e'||r[i]=='i'||r[i]=='o'||r[i]=='u')
        {
            c++;
        }
    }
    // 至少一个元音，2 个辅音
    if(c>=1&&len-c>=2)
    {
        cnt++;
        for(int i = 1;i <= len;i++)        cout<<r[i];
        cout<<endl;

        if(cnt == 25000) exit(0);
    }
}
```

```
// 为 r 数组填写小写字母，直到 len 位
void dfs(int k)
{
    // 下标为 k 的这一位有 n 种不同的填写可能
    for(int i = 1;i <= n;i++)
    {
        // 如果该字符没有用过，且比前一个大
        if(!f[i] && a[i] > r[k-1])
        {
            f[i] = true;// 第 i 个字母标记占用
            r[k] = a[i];
            if(k == len) print();// 尝试打印
            else dfs(k+1);
            f[i] = false;// 回溯，撤销占用
        }
    }
}

int main()
{
    cin>>len>>n;
    // 读入小写字母
    for(int i = 1;i <= n;i++)
    {
        cin>>a[i];
    }

    sort(a+1,a+n+1);

    dfs(1);// 从 r 数组的第 1 位开始填写

      return 0;
}
```

从后向前搜索，按题意优先搜索不包含的情况：

```
#include<bits/stdc++.h>//9-1950-1    javacn
using namespace std;
int a[40];// 存储每个数
int f[40];// 判断第 i 个数是否被选中
int n,t,sum=0,c=0;
// 深搜
void dfs(int k)
{
      int i,j,flag=0;
      if(k<0)
      {
            sum=0;
            for(i=0; i<n; i++)
            {
                  if(f[i]==1)       {     flag=1;     sum+=a[i];  }
            }
            // 如果和为 t，且不是全没选的情况
            if(sum==t && flag != 0)
            {
                  c++;
                  for(i=0; i<n; i++)
                  {
                        if(f[i]==1)       {     cout<<a[i]<<" ";  }
                  }
                  cout<<endl;
            }
            return ;
      }
      // 分两种情况讨论
      f[k]=0;
      dfs(k-1);
      f[k]=1;
      dfs(k-1);
}
```

```
int main()
{
	int i,j;
	cin>>n;
	for(i=0; i<n; i++)
	{
		cin>>a[i];
	}
	cin>>t;

	//按题意，从后往前搜索
	dfs(n-1);
	cout<<c;
	return 0;
}
```

```cpp
#include<iostream>//9-1850-2    jiangyf70    暴力做法和深搜做法        暴力做法
using namespace std;
long long a[30];
int main()
{
    int n, cnt = 0;
    cin >> n;
    for(int i = 0; i < n; i++)    {        cin >> a[i];    }
    long long t;
    cin >> t;
     int m = 1<<n; //暴力做法就是每个数都有取和不取两个状态，排列组合就是 2 的 n 次方。构造一个 2 的 n 次方的数，用位运算判断它第 j 位是不是 1，若是 1 就取这个数。
    long long sum;
    for(int i = 1; i < m; i++)
    {
        sum = 0;
        long long b[30] = {0};
        for(int j = 0; j < n; j++)
        {
            if(i >> j & 1) //用位运算判断 i 的第 j 位是不是 1，若是 1 就取这个数。
            {
                sum += a[j];
                b[j] = a[j];
            }
        }
        if(sum == t)
        {
            cnt++;
            for(int i = 0; i < n; i++)      {  if(b[i]) cout << b[i] <<" ";    }
            cout <<endl;
        }
    }
    cout << cnt;
    return 0;
}
```

深搜

```
#include<bits/stdc++.h>//9-1850-3    jiangyf70
using namespace std;
long long n, m, t, a[25], b[25], cnt = 0;
void print(int k)
{
    cnt++;
    for(int i = k; i >= 1; i--)
    {
        printf("%d ", b[i]);
    }
    printf("\n");
}
// 选择 a[i] 放到 b[pos+1] 的位置
void dfs(long long i, long long sum, long long pos)
{
    if(sum == t && i < 1 && pos > 0)       print(pos);
    if(i < 1)       return;
    dfs(i-1, sum, pos); // 不选
    b[pos+1] = a[i];
    dfs(i-1, sum + a[i], pos + 1); // 选

}

int main()
{
    cin >> n;
    for(int i = 1; i <= n; i++)      cin >> a[i];
    cin >> t;
    dfs(n, 0, 0); // 先 a[n] 元素不选，这样就可以保证输出结果不先是最后一个元素
    cout << cnt;
    return 0;
}
```

```cpp
#include<iostream>//9-1850-4    anselxu
#include<cstring>
#include<algorithm>
#include<cmath>
#include<cstdio>
using namespace std;
int n,a[25],l,ans[25],ab[10001][25];
long long t,sum;
bool b[25];
void arr_print(int x)
{// 打印当前结果
      for(int i=x;i>0;i--)
      {                          // 结果是倒着存的，输出反向
            cout<<ans[i]<<' ';
      }
      cout<<endl;
}
```

```
void dfs(int x,int y)
{
	for(int i=1;i<=y;i++)
	{
		ans[x]=a[i];// 存储当前选择的数
		b[i]=1;// 标记当前数被使用了
		sum+=a[i];// 累计和
		if(sum==t)
		{
			l++;// 记录符合的集合个数
			arr_print(x);//x 记录了集合内数的个数
			dfs(x+1,i-1);// 当前和满足条件，后面可能有相互求和为 0 的组合，所以要继续搜索
		}
		else
		{
			dfs(x+1,i-1);	// 不满足田间，继续搜索
		}

		b[i]=0;// 回溯
		sum-=a[i];// 当前和也回溯到上一步
	}
}
int main()
{
	cin>>n;
	for(int i=1;i<=n;i++)
		cin>>a[i];
	cin>>t;
	dfs(1,n);// 搜索
	cout<<l;
	return 0;
}
```

两个单词合并时，合并部分取的是最小重叠部分 比如 abababab ababab c 这种情况，重叠部分应该是 ab。

2
abababab
abababc
a
相邻的两部分不能存在包含关系。

每个单词最多出现两次.

还要注意所有单词都无法拼接的情况，如：
8
no
new
name
never
national
necessary
ever
me
n

```cpp
#include<bits/stdc++.h>//10-1864     jiangyf70
using namespace std;
string s[60], a[60], r;  //a[] 存放拼接数组 , s 初始读入字符串
char c; // 起始字符
int b[60];  // 判断是否使用 2 次
int n, len = 0;
void print(int k) // 拼接字符串并求最大长度
{
    string s = "";
    for(int i = 1; i <= k; i++)
    {
        s += a[i];
    }
    if(len < s.size())
    {
        len = s.size();  //len 存最大长度
        r = s; //r 保存最长字符串，本题不是必要的。
    }
}
int  congh(string a, string b)  // 判断重复，返回重复长度。
{
    int l = min(a.size(), b.size());
    int la = a.size();
    for(int i = 1; i < l; i++) //要从小到大判断,避免出现 abababab  ababab 这种情况，
判断最小重复长度
    {
        if(a.substr(la-i, i) == b.substr(0, i))
        {
            return i;
        }
    }
    return 0;

}
```

```
void dfs(int k) //k 表示拼接的个数
{
    for(int i = 0; i <  n; i++)
    {

        if(b[i] < 2) // 该串是否使用 2 次
        {
            if(k == 1 && s[i][0] == c) // 如果是第一个且首字母是 c
            {
                a[k] = s[i];
                print(k); // 调用判断长度
                b[i]++; // 使用一次
                dfs(k+1);
                b[i]--; // 回溯
            }
            else
            {
                int j = congh(a[k - 1], s[i]); // 是否重复
                if(j)
                {
                    a[k] = s[i].substr(j);// 把重复部分以后的加入数组。
                    print(k);
                    b[i]++;
                    dfs(k+1);
                    b[i]--;

                }
            }
        }
    }

}
```

```
int main()
{

    cin >> n;
    for(int i = 0; i < n; i++)
    {
        cin >> s[i];
    }
    cin >> c;
    dfs(1);
    cout << len;
//    cout << r; //可以输出最大长度的字符串
    return 0;
}
```

枚举第 1 位可能的数：2 3 5 7，搜索枚举出后续每一位可能的值：1 3 7 9。

```
#include<bits/stdc++.h>//11-1943-1  javacn
using namespace std;
int a[5] = {1,3,7,9};// 非首位的数
int num[5] = {2,3,5,7};// 首位可能的数
int n;
bool prime(int n){
	if(n <= 2) return false;
	for(int i = 2;i <= sqrt(n);i++)
	{
		if(n % i == 0)	{	return false;	}
	}
	return true;
}
// 当前这个数及位数
void dfs(int num,int cnt){
	if(cnt == n)
	{
		cout<<num<<endl;
		return;
	}
	for(int i = 0;i < 4;i++)// 可以向当前数上追加的数
	{
		// 如果追加上来是素数
		if(prime(num * 10 + a[i]))	{	dfs(num*10+a[i],cnt+1); }
	}
}
int main() {
	cin>>n;
	for(int i = 0;i < 4;i++)
	{
		dfs(num[i],1);// 当前这个数，当前这个数的位数
	}
	return 0;
}
```

```cpp
#include<bits/stdc++.h>//11-1943-2     jiangyf70   暴力会超时只能 60 分
using namespace std;
int n;
bool isp(int n)
{
    if(n == 1) return 0;
    for(int i = 2;i * i <= n; i++)          {   if(n % i == 0) return 0;  }
    return 1;
}

void  dfs (int a, int l)
{
    if(isp(a))
    {
        if(l == n)
        {
            cout << a << endl;
            return;
        }
        for(int i = 1; i <= 9; i +=2)
        {
            dfs(a * 10 + i, l + 1);
        }
    }
}
int main()
{
    cin >> n;
    dfs(2, 1);
    dfs(3, 1);
    dfs(5, 1);
    dfs(7, 1);
    return 0;
}
```

暴力会超时

```
#include<bits/stdc++.h>//11-1943-3    jiangyf70
using namespace std;
bool isp(int n)
{
    if(n == 1) return 0;
    for(int i = 2; i * i <= n; i++)
    {
        if(n % i == 0) return 0;
    }
    return 1;
}

bool ism(int n)
{
    while(n)
    {
        if(isp(n)) n /= 10;
        else return 0;
    }
    return 1;
}

int main()
{
    int n;
    cin >> n;
    int x = pow(10, n-1);
    int y = pow(10, n);
    for(int i = x + 1; i < y; i += 2)
    {
        if(ism(i))        cout << i << endl;
    }
    return 0;
}
```

5、深搜综合

先求出每个连通块的大小，再求最大连通块大小。

```
#include <iostream>//1-1380-1    javacn
using namespace std;

int n,m,ma,c;
char a[110][110];// 地图
bool f[110][110];// 标记是否走过
// 方向
int fx[5] = {0,0,1,0,-1};
int fy[5] = {0,1,0,-1,0};

// 求连通块的大小
void dfs(int x,int y)
{
	int tx,ty;
	f[x][y] = true;// 走过标记
	c++;// 求大小
	for(int i = 1; i <= 4; i++)
	{
		tx = x + fx[i];
		ty = y + fy[i];
		// 没走过、没出迷宫（出了迷宫不可能和当前值相等）、要去的点的值和当前值相等
		if(f[tx][ty] == false && a[tx][ty] == a[x][y])
		{
			dfs(tx,ty);
		}
	}
}
```

```
int main()
{
	int i,j;
	cin>>n>>m;
	//读入
	for(i = 1; i <= n; i++)
	{
		for(j = 1; j <= m; j++)
		{
			cin>>a[i][j];
		}
	}

	//从每个没有走过的点开始求连通块大小
	for(i = 1; i <= n; i++)
	{
		for(j = 1; j <= m; j++)
		{
			c = 0;
			if(f[i][j] == false) dfs(i,j);
			ma = max(ma,c);//求最大连通块大小
		}
	}

	cout<<ma;
	return 0;
}
```

```cpp
#include<iostream>//1-1380-2    jiangyf70
using namespace std;
int n, m, cnt = 0;;
char a[110][110];
int fx[] = {0, 1, 0, -1};
int fy[] = {1, 0, -1, 0};

void dfs(int x, int y, char c) // 找到这一片水深为 c 的坐标，将坐标 x, y 标记为 0.
{
    a[x][y] = '0' ;
    cnt++;
    for(int i = 0; i < 4; i++)
    {
        int dx = x + fx[i];
        int dy = y + fy[i];

        if(a[dx][dy] == c)
        {
            dfs(dx, dy, c);
        }
    }
}
```

```
int main()
{
    cin >> n >> m;
    int maxn = 0;
    for(int i = 0; i < n; i++)
    {
        cin >> a[i];
    }
    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < m; j++)
        {
            if(a[i][j] != '0')
            {
                cnt = 0;
                dfs(i, j, a[i][j]);
                if(maxn < cnt) maxn = cnt;
            }
        }
    }
    cout << maxn;
    return 0;
}
```

```cpp
#include <iostream>//1-1380-3 【提高】小 X 学游泳 kevinh
#include <climits>
using namespace std;

int n,m;
char a[110][110];
int fx[4][2] = {{-1,0},{0,1},{1,0},{0,-1}};
int c;
int ma=INT&#95;MIN;

void dfs(int x,int y,char k)
{
    a[x][y]=0;
    c++;
    int tx,ty;
    for(int i=0;i<4;i++)
    {
        tx = x+fx[i][0];
        ty = y+fx[i][1];

        if(a[tx][ty]==k)
        {
            dfs(tx,ty,k);
        }
    }
}
```

```
int main()
{
    cin>>n>>m;
    int i,j;
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=m;j++)
        {
            cin>>a[i][j];
        }
    }

    for(i=1;i<=n;i++)
    {
        for(j=1;j<=m;j++)
        {
            if(a[i][j]!=0)
            {
                c=0;
                dfs(i, j, a[i][j]);

                if(c>ma)
                {
                    ma = c;
                }
            }
        }
    }
    cout<<ma<<endl;

    return 0;
}
```

注意 bfs 要特判起点 == 终点的情况，而 dfs 是不需要特判这种情况的。

```
#include<bits/stdc++.h>//2-1819-1    javacn
using namespace std;
int n;
int q[210][3];// 走到每个点及最少步数
int h,t;
int fx[3] = {0,1,-1};
int a[210];
bool f[210];// 标记走过
int s,e;// 起止点

int main()
{
	cin>>n>>s>>e;
	for(int i = 1;i <= n;i++)
	{
		cin>>a[i];
	}

	// 特判起点 == 终点的情况
	if(s == e)
	{
		cout<<0;
		return 0;
	}
```

```
	//bfs 求从 s 点到 e 点的最少步数
	h = 1;
	t = 1;
	q[1][1] = s;
	q[1][2] = 0;
	f[s] = true;// 走过标记
	int tx,ty;// 表示要去的点
	while(h <= t)
	{
		for(int i = 1;i <= 2;i++)
		{
			tx = q[h][1] + fx[i] * a[q[h][1]];
			// 如果该层可以走
			if(tx>=1&&tx<=n&&f[tx]==false)
			{
				// 入队，标记，判终点
				t++;
				q[t][1] = tx;
				q[t][2] = q[h][2] + 1;
				f[tx] = true;
				// 判终点
				if(tx == e)
				{
					cout<<q[t][2];
					return 0;
				}
			}
		}

		h++;
	}
	cout<<-1;
	return 0;
}
```

本题本质上来说还是最少步数问题，按几次按钮，相当于最少要走多少步，只是本题不同于迷宫类问题，本题只有 2 个方向，上或者下。

```
#include<bits/stdc++.h>//2-1819-2  javacn
using namespace std;
int a[210];
int d[210];// 表示走到每一层最少要按几次按钮
int n;
int s,e;// 代表出发楼层和要到达的楼层
int fx[3] = {0,1,-1};
// 走到 x 楼，最少需要 step 步
void dfs(int x,int step){
	d[x] = step;
	int to;
	for(int i = 1;i <= 2;i++)	// 尝试两个方向
	{
		to = x + fx[i] * a[x];
		// 如果该楼层可以去
		if(to>=1&&to<=n&&step+1<d[to])
		{
			dfs(to,step+1);
		}
	}
}
int main() {
	cin>>n>>s>>e;
	for(int i = 1;i <= n;i++){
		cin>>a[i];
		d[i] = INT_MAX;
	}
	// 从 s 点开始搜索
	dfs(s,0);
	if(d[e] == INT_MAX) cout<<-1;
	else cout<<d[e];
	return 0;
}
```

思路：

遍历每个点，如果是 #，判断从该 # 开始的连通的 # 是矩形还是非矩形

难点：

判断连通块是否是矩形

判断连通块是否是矩形的方法：

求出连通块中所有点的 xy 的最小值：lx,ly 以及 最大值：rx,ry

如果是矩形：lx,ly 必然是左上角的坐标，rx,ry 必然是右下角的坐标

矩形的面积 = (rx-lx+1) * (ry-ly+1)

如果面积 = 连通块中 # 的数量，说明是矩形

```
#include <bits/stdc++.h>//3-1440      javacn
using namespace std;

int n,m;
char a[100][100];
// 方向的变化
int fx[5] = {0,0,1,0,-1};
int fy[5] = {0,1,0,-1,0};
//lx,ly 连通块的坐标的最小值
//rx,ry 连通块的坐标的最大值
//c 每个连通块有几个 #
int lx,ly,rx,ry,c;

// 搜索所有的连通的 #，求出 # 的数量，以及 lx,ly,rx,ry 的值
```

```
// 搜索所有的连通的 #，求出 # 的数量，以及 lx,ly,rx,ry 的值
void dfs(int x,int y)
{
	a[x][y] = '.';
	c++;
	lx = min(x,lx);
	ly = min(y,ly);
	rx = max(x,rx);
	ry = max(y,ry);

	int tx,ty;
	// 尝试四方向
	for(int i = 1;i <= 4;i++)
	{
		tx = x + fx[i];
		ty = y + fy[i];

		// 判断在迷宫内，且是 #
		if(a[tx][ty] == '#')
		{
			dfs(tx,ty);
		}
	}
}
```

```
int main()
{
	cin>>n>>m;
	for(int i = 1;i <= n;i++)
	{
		for(int j = 1;j <= m;j++)
		{
			cin>>a[i][j];
		}
	}

	// 谷仓和奶牛的数量
	int gc = 0,nn = 0;
	for(int i = 1;i <= n;i++)
	{
		for(int j = 1;j <= m;j++)
		{
			// 如果该点是 #
			if(a[i][j] == '#')
			{
				// 初始化
				lx=i;ly=j;rx=i;ry=j;c=0;
				dfs(i,j);
				//cout<<lx<<","<<ly<<" "<<rx<<","<<ry<<" "<<c<<endl;
				// 判断如果是矩形
				if((rx-lx+1)*(ry-ly+1)==c) gc++;
				else nn++;
			}
		}
	}

	// 输出结果
	cout<<gc<<endl<<nn;
	return 0;
}
```

已知条件：

1、可以从任意方格出发

2、可以向上下左右移动

3、移动到的方格中的数字必须比当前方格中的数字更大

求：经过的数字和的最大值

解题步骤：

1、求出从某个点出发，能够经过的所有的数字和 sum

2、在所有数字和中求一个最大值

3、从每个点出发，打擂台求出数字和的最大值

```
#include <bits/stdc++.h>//4-1381  javacn
using namespace std;
int n,m,s,a[110][110];
int fx[5] = {0,0,1,0,-1};
int fy[5] = {0,1,0,-1,0};
int ma;
// 求从 x,y 出发，经过的可能的数字和
void dfs(int x,int y,int sum)
{
	//cout<<x<<","<<y<<" sum="<<sum<<endl;
	ma = max(sum,ma);// 求从某个点开始经过的数字和的最大值
	// 尝试四个方向
	int tx,ty;// 表示将要尝试的坐标
	for(int i = 1;i <= 4;i++)
	{
		tx = x + fx[i];
		ty = y + fy[i];
		// 如果那个点可以去（在迷宫内，且比当前点大）
		if(tx>=1&&tx<=n&&ty>=1&&ty<=m&&a[tx][ty]>a[x][y])
		{
			dfs(tx,ty,sum+a[tx][ty]);
		}
	}
}
```

```
int main()
{
	cin>>n>>m>>s;
	// 生成数组元素的值
	for(int i = 1;i <= n;i++)
	{
		for(int j = 1;j <= m;j++)
		{
			s=(s*345)%19997;
			a[i][j] = (s%10)+1;
			//cout<<a[i][j]<<" ";
		}
	}

	// 测验一下从 4,3 点出发经过可能的数字和
//	dfs(4,3,a[4][3]);
//	cout<<ma;

	// 从每个点开始打擂台，求最大
	for(int i = 1;i <= n;i++)
	{
		for(int j = 1;j <= m;j++)
		{
			dfs(i,j,a[i][j]);
		}
	}

	cout<<ma;
	return 0;
}
```

思路：

沿着最外层的一圈遍历一遍，如果有 '0' 就从该点开始深搜将所有经过的点都标记为 ''

被 '' 围住的 '0'，是一定搜索不到的

最后只要看还剩几个 '0'

```
#include<bits/stdc++.h>//5-1913     javacn
using namespace std;

char a[510][510];
int n,m;
// 方向的变化
int fx[5]= {0,0,1,0,-1};
int fy[5]= {0,1,0,-1,0};

// 深搜值为 '0' 的连通块，将其标记为 '*'
void dfs(int x,int y)
{
	a[x][y] = '*';
	int tx,ty;
	for(int i = 1;i <= 4;i++)
	{
		tx = x + fx[i];
		ty = y + fy[i];
		// 在迷宫内，且为 '0'（为了迷宫不可能是 '0'，值是 '\0' 也就是整数 0）
		if(a[tx][ty] == '0')
		{
			dfs(tx,ty);
		}
	}
}
```

```
int main() {
	cin>>n>>m;
	for(int i = 1;i <= n;i++)
	{
		for(int j = 1;j <= m;j++){
			cin>>a[i][j];
		}
	}
	// 沿着外层的一圈，找到 '0' 从该点深搜
	for(int i = 1;i <= n;i++)
	{
		// 如果是第 1 行，或者最后一行，遍历列
		if(i == 1 || i == n)
		{
			for(int j = 1;j <= m;j++)
			{
				if(a[i][j] == '0')	{	dfs(i,j);	}
			}
		}
		else
		{
			// 如果是该行的第 1 个元素、最后一个元素
			if(a[i][1] == '0') dfs(i,1);
			if(a[i][m] == '0') dfs(i,m);
		}
	}
	int c = 0;
	for(int i = 1;i <= n;i++)
	{
		for(int j = 1;j <= m;j++){
			if(a[i][j] == '0') c++;
		}
	}
	cout<<c;
}
```

```cpp
#include<bits/stdc++.h>//6-1379-1    javacn
using namespace std;
int num[50];// 存储所有的素数
int n;
int k=0;
int ans=0;// 存储结果
// 判素数
bool prime(int x)
{
	if(x <= 1) return false;
	for(int i = 2;i <= sqrt(x);i++)
	{
		if(x % i == 0) return false;
	}
	return true;
}
// 深搜求最多用的素数的数量
//d 为当前下标，sum 为总和，cnt 为使用的数字的个数
void dfs(int d,int sum,int cnt)
{
	if (sum == n)
	{
		ans = max(ans,cnt);
		return;
	}
	// 剪枝
	// 如果 sum 超过 n，或者下标大于素数个数结束
	if (sum > n || d > k)
	{
		return;
	}
	// 选当前数
	dfs(d+1,sum+num[d],cnt+1);
	dfs(d+1,sum,cnt);// 不选当前数
}
```

```
int main()
{
	cin>>n;
	for (int i=2; i<=n; i++)
	{
		// 如果是素数，将其存进 num 数组
		if (prime(i))
		{
			num[++k]=i;
		}
	}
	// 从 num 数组下标为 1 开始讨论，默认总和为 0，素数个数为 0
	dfs(1,0,0);
	cout<<ans;
	return 0;
}
```

回溯算法，选择路径时重要条件：当前将要使用的数要大于刚刚用过的数字，避免出现： 2，3，5， 11/2,3,11,5/2,5,3,11... 等情况， 否则会造成超时。

```
#include <iostream>//6-1379-2   hi0j

using namespace std;

// primes 所有可用素数
// primescnt primes 的长度
// used 标识已经被使用的数字
// path 已经选中的数字
// pathcnt 当前方案数字量
// cnt 最大数字量
int n, primes[205], primescnt ,used[205], path[205], pathcnt, cnt;

// 判断是否素数
bool isPrime(int n)
{
    if (n < 2) return false;
    for (int i = 2; i * i <= n; i++)
    {
        if (n % i == 0) return false;
    }
    return true;
}
```

```
void backtrack(int s){              // 回溯函数， 传入当前总和
    if (s == n)
    {
        if (pathcnt > cnt) cnt = pathcnt;
        return;
    }
    for (int i = 0; i < primescnt; i++)
    {        // 1. 没有被使用过        // 2. 加上后小于等于目标
        // 3. 新值要小于上次加过的值 —— 重要，否则会超时，避免出现 2,3,5,11 回溯后再检测 2,3,11,5/2,5,3,11... 等情况！
        if (!used[primes[i]]&&primes[i]+s<= n && primes[i]>path[pathcnt-1])
        {
            path[pathcnt++] = primes[i];
            used[primes[i]] = true;
            backtrack(s + primes[i]);
            path[pathcnt] = 0;
            pathcnt--;
            used[primes[i]] = false;
        }
    }
}
int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++ )     // 将小于 n 的素数都加入数组 primes
    {
        if (isPrime(i)){
            primes[primescnt++] = i;
        }
    }
    backtrack(0);
    cout << cnt;
    cout << endl;
    return 0;
}
```

```cpp
#include<bits/stdc++.h>//6-1379-3  jiangyf70
using namespace std;
int a[220], b[220], c[220], d[220];
int n, cnt = 0, k = 1, maxn = 0, sum = 0;
void prime(int n)  // 质数筛选法
{
    for(int i = 2; i * i <= n; i++)
    {
        int j = i + i;
        while(j <= n)
        {
            a[j] = 1;
            j += i;
        }
    }

    for(int i = 2; i <= n; i++)
    {
        if(a[i] == 0) b[k++] = i; // 将筛选出来的质数放到 b[], 下标从 1 开始  否则 dfs 初始参数不好弄 .
    }
}
void print(int x)
{
    if(maxn < x) maxn = x;
//    for(int i = 1; i <= x; i++) cout << d[i] << " ";  // 打印出来, 这两句可以不用
//    cout << endl;
}
```

```
void dfs(int x, int j, int sum, bool f)  // 选第 x 个数，从 b[j] 开始选， 和为 sum, f 为 b[j] 选还是不选
{
     if(f) d[x] = b[j]; // 将 b[j] 放到结果数组，便于打印  加个 f 判断一下是否要将 b[j] 加入 d[x]
    if(sum == n) print(x);
    else if(sum < n && j <= k && x < n)  //k 是 1-n 之间所有的质数个数
    {
        dfs(x + 1, j + 1, sum + b[j+1], 1); // 选 b[j+1] 这个数；
        dfs(x, j + 1, sum, 0); // 不选 b[j+1] 这个数.
    }
}

int main()
{

    cin >> n;
    prime(n);
    //for(int i = 1; i < k; i++) cout << b[i] << endl;  // 显示一下 1~n 之间的质数

    dfs(0, 0, 0, 0); // 选第 0 个数，从 b[0] 开始，和为 0，这就是 b[] 数组为什么要从 1 开始了. 这样第一个数也有选和不选两种可能.
    cout << maxn;
    return 0;

}
```

注意题目输出的结果的顺序是按列来遍历的，

```
#include<iostream>//7-1832  八皇后  jiangyf70
using namespace std;
int n;
int h[20][20], s[20], lx[40], rx[40]; //h[][]是存放排列的数组， s[]横行被占用，lx左斜  rx右斜行
int cnt = 0;
void print()
{
    cnt++;
    cout <<"No. "<< cnt << endl;
    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < n; j++)
        {
            cout << h[i][j] << " ";
        }
        cout << endl;
    }

}
```

```
void dfs(int x) // 放置第 x 列的皇后，从 0 遍历到 n-1 列，这里就是 0~7，仔细看题目结果，是按列排序输出结果的，刚开始按行输出结果不对。
{
    if(x == n) print();
    else if(x < n)
    {
        for(int i = 0; i < n; i++)
        {
            if(s[i] == 0 && lx[x + i] == 0 && rx[i - x + n] == 0) // 若 h[i][x] 的横 \ 斜都是 0，说明可以放置
            {
                h[i][x] = 1; // 为 i,x 放置皇后
                s[i] = lx[x+i] = rx[i - x + n] = 1; // 将标志设为 1
                dfs(x+1);
                h[i][x] = 0;
                s[i] = lx[x+i] = rx[i - x + n] = 0; // 将标志设为 0
            }
        }
    }


}


int main()
{
    //cin >> n;
    n = 8;
    dfs(0);

    return 0;
}
```

逐行讨论每一行的皇后可以放的位置，如果第 i 行第 j 列的位置放了一个皇后，那么：
第 i 行被占，但这个条件不用判断，因为第 i 行放好皇后会接着放第 i+1 行，因此不可能在同一行放 2 个皇后；
第 j 列被占用，通过布尔数组标记；
i,j 这个位置对应的 2 个斜角被占；通过规律分析可以发现，从右上角到左下角的斜角可以通过 i+j 来唯一表示，从左上角到右下角的斜角可以通过 i+8-j 来唯一表示。

```
#include <bits/stdc++.h>//8-1833-1  八皇后 javacn
using namespace std;

int ans, n, x1, x2, k;
int r[100],c[100];
bool f[30];// 列是否重复
bool d1[30];// 右对角是否重复
bool d2[30];// 左对角是否重复
vector<int> res[100];// 存储所有的解

// 从第 1 行开始找位置，由于是从第 1 行开始找位置，逐行找，因此不用判断是否有 2 个子在同一行
// 当第 row 行找到合适的位置，标记好，再去标记第 row+1 行，这样行上不会有重复
```

```
void queen(int row) {
	// 循环第 row 行的每一列
	for (int col=1; col<=8; col++)
	{
		if (f[col]==false && d1[row+col]==false && d2[row-col+8]==false)
		{
			r[row] = col;
			f[col] = true;
			d1[row+col] = true;
			d2[row-col+8] = true;

			if (row == 8)
			{
				k++;
				for (int i=1; i<=8; i++)	{  res[k].push_back(r[i]); }
			}
			else		{	queen(row+1);	}

			f[col] = false;
			d1[row+col] = false;
			d2[row-col+8] = false;
		}
	}
}
int main() {
	queen(1);	// 从第 1 列开始找位置
	cin >> n;
	int x;
	for (int i=1; i<=n; i++)
	{
		cin >> x;
		for(int j = 0; j < res[x].size(); j++)	{	cout<<res[x][j];  }
		cout<<endl;
	}
}
```

```
#include<bits/stdc++.h>//8-1833-2    dragoncatter
using namespace std;
int f1[110], f2[120], f3[120];
string s[100];
int n , t, cnt;

void dfs(int step, string str)
{
      if(step == 9)
      {
            s[++cnt] = str;
            return ;
      }
      else
      {
            for(int i = 1; i <= 8; i++)
            {
                  if(!f1[i] && !f2[step + i] && !f3[step - i + 20])
                  {
                        f1[i] = 1;
                        f2[step+i] = 1;
                        f3[step-i+20]=1;
                        dfs(step+1, str + char(i +  '0' ));
                        f1[i] = 0;
                        f2[step+i] = 0;
                        f3[step-i+20]=0;
                  }
            }
      }
}
```

```
int main()
{
	cin >> n;
	dfs(1, "");
	while(n--)
	{
		cin >> t;
		cout << s[t] << '\n';
	}
	return 0;
}
```

#3 这题又是按行来遍历的.

```
#include<iostream>//8-1833-3  jiangyf70
using namespace std;
int n;
int h[20][20], s[20], lx[40], rx[40]; //h[][] 是存放排列的数组， s[] 横行被占用，lx 左斜  rx 右斜行
int cnt = 0, r[100][10]; //r 存放第 i 个结果，每行就是对应的皇后串
void print()
{
    cnt++;
    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < n; j++)
        {
            if(h[i][j]) r[cnt][i] = j + 1; //j 从 0 开始计数的，所以要加 1
        }

    }

}
```

```
void dfs(int x) // 放置第 x 列的皇后，从 0 遍历到 n-1 列，这里就是 0~7，仔细看题目结果，是按列排序输出结果的，刚开始按行输出结果不对。
{
    if(x == n) print();
    else if(x < n)
    {
        for(int i = 0; i < n; i++)
        {
            if(s[i] == 0 && lx[x + i] == 0 && rx[i - x + n] == 0) // 若 h[i][x] 的横 \ 斜都是 0，说明可以放置
            {
                h[x][i] = 1; // 为 x,i 放置皇后 按行遍历
                s[i] = lx[x+i] = rx[i - x + n] = 1; // 将标志设为 1
                dfs(x+1);
                h[x][i] = 0;
                s[i] = lx[x+i] = rx[i - x + n] = 0; // 将标志设为 0
            }
        }
    }
}
int main()
{
    //cin >> n;
    n = 8;
    dfs(0);
    int k, a;
    cin >> k;
    for(int i = 0; i < k; i++)
    {
        cin >> a;
        for(int j = 0; j < n; j++) cout << r[a][j];
        cout << endl;
    }
    return 0;
}
```

穷举所有的运算可能，逐个运算检验能否算到 24。

4 个数有 3 个位置，每个位置有 3 种填写的可能，因此一共有 27 种运算符的组合。

```
#include <bits/stdc++.h>//9-1955-1    javacn
using namespace std;

/*
1. 穷举每组数，对应的 27 种组合出来的运算
2. 表达式计算
*/
int n,a[10];
int cnt = 0;// 计数器
string t = "+-*";
bool f;// 标记某组数经过运算能否得到 24
```

```
// 计算 a 数组通过 27 种不同运算符组合的运算能否得到 24
void calc(string opt)
{
	int b[10];
	// 拷贝 a 数组，不能修改 a 数组的值，后续计算还需要使用
	for(int i = 0;i < 4;i++) b[i] = a[i];

	// 先计算乘
	for(int i = 0;i < opt.size();i++)
	{
		if(opt[i] == '-')
		{
			b[i+1] = -b[i+1];
		}
		else if(opt[i] == '*')
		{
			b[i+1] *= b[i];
			b[i] = 0;
		}
	}

	// 计算加
	int sum = 0;
	for(int i = 0;i < 4;i++)
	{
		sum += b[i];
	}

	if(sum == 24) f = true;
}

// 搜索穷举运算符
```

```
// 搜索穷举运算符
void dfs(string opt)
{
	if(opt.size() == 3)
	{
		// 如果通过计算能算到 24
		calc(opt);
		return;
	}
	for(int i = 0;i < t.size();i++)
	{
		dfs(opt+t[i]);
	}
}
int main()
{
	cin>>n;
	while(n--)
	{
		cin>>a[0]>>a[1]>>a[2]>>a[3];
		f = false;// 假设算不到 24
		dfs("");// 穷举运算符
		if(f) cnt++;
	}

	cout<<cnt;
	return 0;
}
```

```cpp
#include<iostream>//9-1955-2    jiangyf70
using namespace std;
int a[4], cnt = 0;
bool f;
void cal(){
    if(a[0] + a[1] + a[2] + a[3] == 24 ) f = 1;
    if(a[0] + a[1] + a[2] - a[3] == 24 ) f = 1;
    if(a[0] + a[1] + a[2] * a[3] == 24 ) f = 1;
    if(a[0] + a[1] - a[2] + a[3] == 24 ) f = 1;
    if(a[0] + a[1] - a[2] - a[3] == 24 ) f = 1;
    if(a[0] + a[1] - a[2] * a[3] == 24 ) f = 1;
    if(a[0] + a[1] * a[2] + a[3] == 24 ) f = 1;
    if(a[0] + a[1] * a[2] - a[3] == 24 ) f = 1;
    if(a[0] + a[1] * a[2] * a[3] == 24 ) f = 1;//

    if(a[0] - a[1] + a[2] + a[3] == 24 ) f = 1;
    if(a[0] - a[1] + a[2] - a[3] == 24 ) f = 1;
    if(a[0] - a[1] + a[2] * a[3] == 24 ) f = 1;
    if(a[0] - a[1] - a[2] + a[3] == 24 ) f = 1;
    if(a[0] - a[1] - a[2] - a[3] == 24 ) f = 1;
    if(a[0] - a[1] - a[2] * a[3] == 24 ) f = 1;
    if(a[0] - a[1] * a[2] + a[3] == 24 ) f = 1;
    if(a[0] - a[1] * a[2] - a[3] == 24 ) f = 1;
    if(a[0] - a[1] * a[2] * a[3] == 24 ) f = 1;//

    if(a[0] * a[1] + a[2] + a[3] == 24 ) f = 1;
    if(a[0] * a[1] + a[2] - a[3] == 24 ) f = 1;
    if(a[0] * a[1] + a[2] * a[3] == 24 ) f = 1;
    if(a[0] * a[1] - a[2] + a[3] == 24 ) f = 1;
    if(a[0] * a[1] - a[2] - a[3] == 24 ) f = 1;
    if(a[0] * a[1] - a[2] * a[3] == 24 ) f = 1;
    if(a[0] * a[1] * a[2] + a[3] == 24 ) f = 1;
    if(a[0] * a[1] * a[2] - a[3] == 24 ) f = 1;
    if(a[0] * a[1] * a[2] * a[3] == 24 ) f = 1;
}
```

```
int main()
{
    int n;
    cin >> n;
    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < 4; j++) cin >> a[j];
        f = 0;
        cal();
        if(f) cnt++;
    }
    cout << cnt;
    return 0;
}
```

思路：穷举所有的运算可能，由于本题不允许改变运算数顺序，因此本题只能穷举运算符的可能 。

有 n 个数，有 n-1 个空格可以插入运算符，每个空格可以插入 3 种不同的运算符，因此结果有：3^9 种不同的可能 。

```
#include<bits/stdc++.h>//10-1956    javacn
using namespace std;

/*
思路：穷举所有的运算可能
由于本题不允许改变运算数顺序，因此本题只能穷举运算符的可能
有 n 个数，有 n-1 个空格可以插入运算符，每个空格可以插入 3 种不同的运算符
因此结果有：3^9 种不同的可能
*/
int n;
int a[20],b[20];
string t ="+-*";
int ans = 0;
```

```
// 通过长度为 n-1 的运算符字符串 s 以及 n 个整数进行计算
// 判断结果是否是 24
bool calc(string s)
{
	// 由于 a 数组还要用，要把 a 数组留下来
	for(int i = 0;i < n;i++)
	{
		b[i] = a[i];
	}

	// 先计算乘法
	for(int i = 0;i < s.size();i++)
	{
		// 如果上一个运算符是 -，那么将 a[i] 视为 -a[i]
		// 这样最终就没有减法，只有加法
		if(s[i] == '-')
		{
			b[i+1] = -b[i+1];
		}
		else if(s[i] == '*')
		{
			b[i+1] *= b[i];
			b[i] = 0;
		}
	}

	// 再计算加法
	int r = 0;
	for(int i = 0;i < n;i++)
	{
		r += b[i];//		cout<<b[i]<<" ";
	}
	if(r == 24) return true;
	else return false;
}
```

```
// 穷举不同的运算符的排列组合
void dfs(string s)
{
	if(s.size() == n - 1)
	{
		if(calc(s)) ans++;
		return;
	}

	//3 种不同的选择
	for(int i = 0;i < 3;i++)
	{
		dfs(s+t[i]);
	}
}

int main()
{
	cin>>n;
	for(int i = 0;i < n;i++)
	{
		cin>>a[i];
	}

	dfs("");
	cout<<ans;
    return 0;
}
```

本题求解：满足曼哈顿距离 <=2 的情况下的连通块的数量

距离 x,y 点，满足曼哈顿距离 <=2 的点，有 12 个

```
#include<bits/stdc++.h>//11-1966 -1    javacn
using namespace std;

/*
本题求解：满足曼哈顿距离 <=2 的情况下的连通块的数量
距离 x,y 点，满足曼哈顿距离 <=2 的点，有 12 个
*/
int n,m;
char a[110][110];
int c = 0;// 统计有多少个连通块

// 存储相对 xy 点曼哈顿距离 <=2 的相对位置的差值
int fx[13] = {0,-2,-1,-1,-1,0,0,0,0,1,1,1,2};
int fy[13] = {0,0,-1,0,1,-2,-1,1,2,-1,0,1,0};

// 将 xy 相邻 12 个方向的能走到的 #，深搜标记为 -
void dfs(int x,int y)
{
	a[x][y] = '-';
	int tx,ty;
	for(int i = 1;i <= 12;i++)
	{
		tx = x + fx[i];
		ty = y + fy[i];
		// 出边界不可能是 #
		if(a[tx][ty] =='#')
		{
			dfs(tx,ty);
		}
	}
}
```

```
int main()
{
	cin>>n>>m;
	for(int i = 1;i <= n;i++)
	{
		for(int j = 1;j <= m;j++)
		{
			cin>>a[i][j];
		}
	}

	// 遍历每个点，如果是 #，深搜将 12 方向相邻的 # 都标记为 -
	for(int i = 1;i <= n;i++)
	{
		for(int j = 1;j <= m;j++)
		{
			if(a[i][j] == '#')
			{
				c++;
				dfs(i,j);// 从 ij 开始搜索标记
			}
		}
	}

	cout<<c;
	return 0;
}
```

```cpp
#include<bits/stdc++.h>//11-1966-2  jiangyf70
using namespace std;
int n, m;
char a[110][110];
int r[110110][3];
int fx[] ={0, 0, 0, 0, 1, 2, -1, -2, 1, 1, -1, -1};
int fy[] ={1, 2, -1, -2, 0, 0, 0, 0, 1, -1, 1, -1};

void bfs(int x, int y)
{
    memset(r, 0, sizeof(r));
    int head = 0, tail = 0;
    r[head][0] = x;
    r[head][1] = y;
    a[x][y] = '-';
    while(head <= tail)
    {
        for(int i = 0; i < 12; i++)
        {
            int dx = r[head][0] + fx[i];
            int dy = r[head][1] + fy[i];
            if(a[dx][dy] == '#')
            {
                tail++;
                r[tail][0] = dx;
                r[tail][1] = dy;
                a[dx][dy] = '-';
            }
        }
        head++;
    }
}
```

```
int main()
{
    cin >> n >> m;
    for(int i = 0; i < n; i++) cin >> a[i];
    int cnt = 0;
    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < m; j++)
        {
            if(a[i][j] == '#')
            {
                cnt++;
                bfs(i, j);
            }
        }
    }
    cout << cnt;
    return 0;
}
```

注意特殊情况：

8 4

3 1 1 1

1 1 1 0

1 1 1 1

0 1 1 2

1 4 0 1

1 4 0 1

1 0 1 1

1 0 1 1

输出：8

数字 0： 障碍物。

数字 1： 空地， 小 H 可以自由行走。

数字 2： 小 H 出发点， 也是一片空地。

数字 3： 小 H 的家。

数字 4： 有金币在上面的空地。

只要下一个深搜的点能走，且走到那个点血量会变多（默认每个点的值血量都是 0）或者步数更少（因为有可能先走出一个血量更多的走法，后面再走发现血量没有更多，但是在可以走到终点的情况下有更少的步数也能走到，比如：不去捡金币也能走到终点），那么可以走。

```
#include<bits/stdc++.h>//12-1914     javacn
using namespace std;
int a[10][10],mi=INT_MAX;
int n,m,sx,sy,ex,ey;
int b[20][20];// 到每个点的最少血量
int s[20][20];// 到每个点的最少步数
// 方向数组
int fx[5] = {0,0,1,0,-1};
int fy[5] = {0,1,0,-1,0};
// 递归
```

```
// 递归
void dfs(int x,int y,int step,int blood)
{
	if(a[x][y]==4)
	{
		blood = 6;
	}
	b[x][y] = blood;
	s[x][y] = step;
	// 如果到达终点停止递归
	if(a[x][y]==3)
	{
		mi = min(mi,step);
		return;
	}
	int tx,ty,i;
	for(i=1;i<=4;i++)
	{
		tx = x + fx[i];
		ty = y + fy[i];
		// 如果下一个点在迷宫内，且到下一个点至少还有 1 个血，且不是障碍，且到下一个点的血更多或者步数更少
		if(tx>=1&&tx<=n&&tx>=1&&ty<=m && blood-1>=1 && a[tx][ty]!=0 && (b[x][y]-1 > b[tx][ty] || step + 1 < s[tx][ty]))
		{
			dfs(tx,ty,step+1,blood-1);
		}
	}
}
```

```
int main()
{
	cin>>n>>m;
	int i,j;
	for(i=1;i<=n;i++)
	{
		for(j=1;j<=m;j++)
		{
			cin>>a[i][j];
			s[i][j] = INT_MAX;
			//出发地
			if(a[i][j]==2)
			{
				sx = i,sy = j;
			}
			else if(a[i][j] ==3)
			{
				ex = i,ey = j;//目的地
			}
		}
	}
	//起始点
	dfs(sx,sy,0,6);
	if(mi == INT_MAX)		cout<<-1;
	else cout<<mi;
	return 0;
}
```