- 搜索
  - DFS-深度优先搜索
  - BFS-广度优先搜索(个人感觉比dfs好用)
  - 例题

搜索

DFS-深度优先搜索

```
bool check(传入值)
{
   if(满足条件)//通常是 1.vis[i]==false 2.不越界 3.题目给的要求
       return true;
   else return false;
}
void dfs(传入值)
{
    if(到达边界or走不下去了)
    {
        相应操作;
        return;
    }
    尝试每一种可能
    {
        if(满足check条件)
        {
            标记
            dfs(step+1)
            恢复初始状态（回溯用到）//记得恢复完全，如果有ans++这样的要恢复成ans--
        }
    }
}
```

BFS-广度优先搜索(个人感觉比dfs好用)

当搜索范围无限大的时候dfs就用不上啦，这时得用bfs

```
bool check(传入值)
{
   if(满足条件)//通常是 1.vis[i]==false 2.不越界 3.题目给的要求
       return true;
   else return false;
}
```

```
queue<数据类型> q;//bfs用队列实现
void bfs(传入值a)
{
    标记走过;//也可以遍历中在bfs之前标记->vis[i]=true;bfs(i)
    q.push(a);//把a压入队列;
    while(!q.empty)
    {
        b=q.front()//取出队首
        q.pop()//弹出队首
        for(遍历每种可能)
        {
            下一个可能的状态b
            if(check(b))
            {
                标记走过
                更新位置
                b入队
            }
        }
    }
}
```

扩展：记忆化搜索

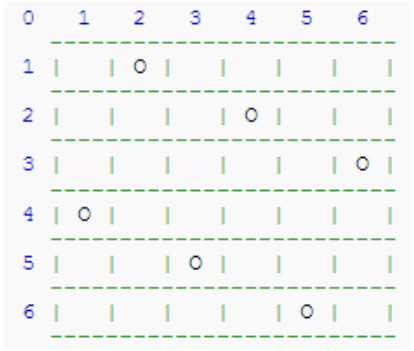
例题

【1】P1219 [USACO1.5]八皇后 Checker Challenge

题目描述

复制Markdown 展开

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



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

行号 $1\ 2\ 3\ 4\ 5\ 6$

列号 $2\ 4\ 6\ 1\ 3\ 5$

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

输入格式

一行一个正整数 n，表示棋盘是 $n \times n$ 大小的。

输出格式

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

输入输出样例

输入 #1 复制

```
6
```

输出 #1 复制

```
2 4 6 1 3 5
3 6 2 5 1 4
4 1 5 2 6 3
4
```

说明/提示

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

题目翻译来自NOCOW。

```
#include<bits/stdc++.h>
using namespace std;
int res_column[13]={0};//对应行的目标列
int occupy_column[13]={0},diagonal_1[26]={0},diagonal_2[26]={0};//被占领的列，主对角线，辅对角线
```

```
int ans=0;//记录答案总数
int n;
void print()
{
    if(ans<=2)
    {
        for(int i=1;i<=n;i++)
            cout<<res_column[i]<<' ';
        cout<<endl;
    }
}
void dfs(int i)//一行一行判断
{
    if(i>n)
    {
        print();
        ans++;
        return;
    }
    for(int j=1;j<=n;j++)
    {
        if((!occupy_column[j])&&(!diagonal_1[i+j])&&(!diagonal_2[i-j+n]))//如果所在列，主对角线，副对角线都没被占领就可以操作
        {
            res_column[i]=j;//标记答案（第i行第j列被占领）
            occupy_column[j]=1;
            diagonal_1[i+j]=1;
            diagonal_2[i-j+n]=1;//占领
            dfs(i+1);
            occupy_column[j]=0;
            diagonal_1[i+j]=0;
            diagonal_2[i-j+n]=0;//回溯
        }
    }
}
int main()
{
    cin>>n;
    dfs(1);
    cout<<ans;
}
```

【2】p1443 马的遍历

P1443 马的遍历

提交答案 加入题单

题目描述

复制Markdown 展开

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

输入格式

输入只有一行四个整数，分别为 n, m, x, y。

输出格式

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

输入输出样例

输入 #1 复制

```
3 3 1 1
```

输出 #1 复制

```
0    3    2
3    -1   1
2    1    4
```

说明/提示

数据规模与约定

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

```cpp
#include<bits/stdc++.h>
using namespace std;
int n,m,x,y;
int direction_x[8]={-1,-2,-2,-1,1,2,2,1};//总所周知马走日
int direction_y[8]={2,1,-1,-2,2,1,-1,-2};
queue<pair<int,int> >q;//bfs用队列实现
int step[400][400];//存步数
bool vis[400][400];//这个点是否走过
int main()
{
    cin>>n>>m>>x>>y;
    memset(vis,false,sizeof(vis));//先全部初始化为没经过
    memset(step,-1,sizeof(step));
    step[x][y]=0;//设置起点为第零步
    vis[x][y]=true;//把起点标记为已经走过的状态
    q.push(make_pair(x,y));//pair真好用
    while(!q.empty())//只要队列非空就说明能扩展到其他的点，就一直做循环
```

```
    {
        int x_first=q.front().first;
        int y_first=q.front().second;//取出队首并出队
        q.pop();
        for(int i=0;i<8;i++)//遍历所有可能状态
        {
            int x_next=x_first+direction_x[i];
            int y_next=y_first+direction_y[i];
            if(x_next<1||x_next>n||y_next<1||y_next>m||vis[x_next]
[y_next])continue;//边界条件不满足就说明走不通，跳过
            vis[x_next][y_next]=true;//把后一个点标记为走过
            q.push(make_pair(x_next,y_next));//把这个点加入队列中
            step[x_next][y_next]=step[x_first][y_first]+1;//相当于是一个递推关系
        }
    }
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)
        {
            cout<<left<<setw(5)<<step[i][j];//控制行距
        }
        cout<<endl;
    }
}
```

【3】P1605 迷宫

P1605 迷宫

提交答案 加入题单

题目描述

复制Markdown 展开

给定一个 $N \times M$ 方格的迷宫，迷宫里有 T 处障碍，障碍处不可通过。

在迷宫中移动有上下左右四种方式，每次只能移动一个方格。数据保证起点上没有障碍。

给定起点坐标和终点坐标，每个方格最多经过一次，问有多少种从起点坐标到终点坐标的方案。

输入格式

第一行为三个正整数 N, M, T，分别表示迷宫的长宽和障碍总数。

第二行为四个正整数 SX, SY, FX, FY，SX, SY 代表起点坐标，FX, FY 代表终点坐标。

接下来 T 行，每行两个正整数，表示障碍点的坐标。

输出格式

输出从起点坐标到终点坐标的方案总数。

输入输出样例

输入 #1 复制

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

输出 #1 复制

```
1
```

说明/提示

对于 100% 的数据，$1 \le N, M \le 5$，$1 \le T \le 10$，$1 \le SX, FX \le n$，$1 \le SY, FY \le m$。

```cpp
#include<bits/stdc++.h>
using namespace std;
int m,n,t;
int sx,sy,fx,fy;
int ans=0;
int mp[10][5]={0};//整体地图，先把每个点都初始化为零
int plus_x[4]={1,-1,0,0},plus_y[4]={0,0,1,-1};//下一步可能的四个状态
bool flag[10][10]={false};//判断某个点是否到达过
bool judge(int x,int y)//判断这个点是否符合要求
{
    if(x<1||x>n||y<1||y>m)return false;//越界
    if(mp[x][y]==1||flag[x][y]==true)return false;//这个状态之前走过或者有障碍
    return true;//否则就可以走
}
void dfs(int x,int y)
{
```

```
    if(x==fx&&y==fy)//到达终点
    {
        ans++;
        return;//return回上一个状态
    }
    flag[x][y]=true;//先标记为走过
    for(int i=0;i<4;i++)//遍历下个状态所有可能情况
    {
       int next_x,next_y;
       next_x=x+plus_x[i];
       next_y=y+plus_y[i];
       if(judge(next_x,next_y))//判断这个情况是否可行
       {
        dfs(next_x,next_y);//对下个状态dfs
        flag[next_x][next_y]=false;//回溯
       }
    }
}
int main()
{
    cin>>m>>n>>t;//长 宽 障碍数
    cin>>sx>>sy>>fx>>fy;//起点坐标和终点坐标
     for(int i=0;i<t;i++)//加入障碍
    {
        int x,y;
        cin>>x>>y;
        mp[x][y]=1;//有障碍的话就置为1
    }
    dfs(sx,sy);//从起点开始dfs
    cout<<ans;
}
```

【4】p1706 全排列问题

P1706 全排列问题

提交答案 加入题单

题目描述

复制Markdown 展开

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

输入格式

一个整数 n。

输出格式

由 $1 \sim n$ 组成的所有不重复的数字序列，每行一个序列。

每个数字保留 5 个场宽。

输入输出样例

输入 #1 复制

```
3
```

输出 #1 复制

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

说明/提示

$1 \leq n \leq 9$。

```cpp
#include<bits/stdc++.h>
using namespace std;
int vst[10];//是否使用过该数字
int res[10];//储存结果
int n;
void print()//输出n个数全排列结果
{
    for(int i=1;i<=n;i++)
    cout<<"    "<<res[i];//只有四个空格
    cout<<endl;
}
void dfs(int number)//深搜，num表示现在在排第num个
{
    if(number==n+1)//表示前n个数已经排好了
    {
        print();
        return;
```

```
    }
    for(int i=1;i<=n;i++)//尝试范围内的数是否满足条件
    {
        if(vst[i]==0)//没有被使用过
        {
            res[number]=i;//储存排列结果
            vst[i]=1;//标记这个数字已经被使用
            dfs(number+1);//dfs下一个
            vst[i]=0;//回溯（相当于留条退路吗？？）
        }
    }
}
int main()
{
    cin>>n;
    dfs(1);
}
```

【5】P4961 小埋与扫雷

题目描述

小埋会告诉你一盘扫雷，用一个 $n \times m$ 的矩阵表示，1 是雷，0 不是雷，请你告诉她这盘扫雷的 3bv 。

周围八格没有“雷”且自身不是“雷”的方格称为“空格”，周围八格有“雷”且自身不是“雷”的方格称为“数字”，由“空格”组成的八连通块称为一个“空”。3bv = 周围八格没有“空格”的“数字”个数 + “空”的个数。

如果看不懂上面的计算方式，可以看题目背景中给出的教程，或者看下面的样例解释。

注：八连通

输入格式

第一行有两个整数 n 和 m，代表这盘扫雷是一个 $n \times m$ 的矩阵。

后面的 n 行每行有 m 个整数，表示这个矩阵，每个数字为 0 或 1，1 代表是雷，0 代表不是雷。

输出格式

一个整数，代表这盘扫雷的 3bv 。

输入输出样例

输入 #1 复制

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

输出 #1 复制

```
13
```

说明/提示

$1 \leq n, m \leq 1000$

样例解释

```
#include<bits/stdc++.h>
using namespace std;
int mp[1005][1005]={0};//1: 雷, 2: 空格, 3: 数字
```

```
bool vis[1005][1005];
int plus_x[8]={1,1,1,0,0,-1,-1,-1};
int plus_y[8]={1,0,-1,1,-1,1,0,-1};
int n,m,ans;
void find(int x,int y)//判断是数字还是空格
{
    bool boom=false;//是否有雷，有雷的话就不用再扫下去了
    for(int i=0;i<8&&!boom;i++)
    {
        int x_next=x+plus_x[i];
        int y_next=y+plus_y[i];
        if(x_next>=0&&x_next<n&&y_next>=0&&y_next<m)
        {
            if(mp[x_next][y_next]==1)
                boom=true;
        }
    }
    if(!boom)mp[x][y]=2;//是空格
    else mp[x][y]=3;//是数字
}
void dfs(int x,int y)//判断八连通块个数
{
    vis[x][y]=true;
    for(int i=0;i<8;i++)
    {
        int x_next=x+plus_x[i];
        int y_next=y+plus_y[i];
        if(x_next>=0&&x_next<n&&y_next>=0&&y_next<m)
        {
            if(mp[x_next][y_next]==2&&!vis[x_next][y_next])
                dfs(x_next,y_next);//不用回溯
        }
    }
}
bool check(int x,int y)//周围有没有空格
{
    bool st=true;
    for(int i=0;i<8&&st==true;i++)
    {
        int x_next=x+plus_x[i];
        int y_next=y+plus_y[i];
        if(x_next>=0&&x_next<n&&y_next>=0&&y_next<m)
        {
            if(mp[x_next][y_next]==2)
                st=false;
        }
    }
    return st;
}
int main()
{
    cin>>n>>m;
    for(int i=0;i<n;i++)//初始化地图
    {
        for(int j=0;j<m;j++)
        cin>>mp[i][j];
    }
```

```
    for(int i=0;i<n;i++)//判断数字还是空格
    {
        for(int j=0;j<m;j++)
        {
            if(!mp[i][j])//是0才判断
                find(i,j);
        }
    }
    for(int i=0;i<n;i++)//找连通块
    {
        for(int j=0;j<m;j++)
        {
           if(mp[i][j]==2&&!vis[i][j])
           {
                dfs(i,j);
                ans++;
           }
        }
    }
    for(int i=0;i<n;i++)//周围没有空格的数字
    {
        for(int j=0;j<m;j++)
        {
            if(mp[i][j]==3)
            {
                if(check(i,j))
                    ans++;
            }
        }
    }
    cout<<ans;
}
```

【6】7-2 山

7-2 山 分数 300

全屏浏览题目 切换布局 作者 Drizzle 单位 山东科技大学

Drizzle 前往山地统计大山的数目，现在收到这片区域的地图，地图中用`0（平地）`和`1（山峰）`绘制而成，请你帮忙计算其中的大山数目
山总是被平地**四面**包围着，每一座山只能在水平或垂直方向上连接相邻的山峰而形成。一座山峰四面被平地包围，这个山峰也算一个大山
另外，你可以假设地图的四面都被平地包围着。

要求:

输入：第一行输入M,N分别表示地图的行列，接下来M行每行输入N个数字表示地图
输出：输出一个整数表示大山的数目

示例:

输入:

```
4 5
1 1 0 0 0
1 1 0 0 0
0 0 1 0 0
0 0 0 1 1
```

输出:

```
3
```

范围:

对于 5% 的数据：M，N ≤ 10
对于 100% 的数据：M，N ≤ 2000

```cpp
#include<bits/stdc++.h>
using namespace std;
int m,n,ans=0;
int mp[2005][2005]={0};
bool vis[2005][2005];
int x_plus[4]={1,0,0,-1};
int y_plus[4]={0,1,-1,0};
queue<pair<int,int> >q;
int main()
{
    memset(vis,false,sizeof(vis));
    cin>>m>>n;
    for(int i=0;i<m;i++)
    {
        for(int j=0;j<n;j++)
            cin>>mp[i][j];//初始化地图
    }

    for(int i=0;i<m;i++)//遍历+bfs
    {
        for(int j=0;j<n;j++)
        {
            if(mp[i][j]==1&&vis[i][j]==false)
            {
                vis[i][j]=true;
                q.push(make_pair(i,j));
                while(!q.empty())
                {
                    int x=q.front().first;
                    int y=q.front().second;
                    q.pop();
```

```
                    for(int k=0;k<4;k++)
                    {
                        int x_next=x+x_plus[k];
                        int y_next=y+y_plus[k];
                        if(x_next>=0&&x_next<m&&y_next>=0&&y_next<n&&mp[x_next]
[y_next]==1&&vis[x_next][y_next]==false)
                        {
                            vis[x_next][y_next]=true;
                            q.push(make_pair(x_next,y_next));
                        }
                    }

                }
                ans++;//注意放的位置
            }
        }
    }

    cout<<ans;
}
```

【7】7-3 跳跃

7-3 跳跃 分数 300

全屏浏览题目 切换布局 作者 Drizzle 单位 山东科技大学

Drizzle 被困到一条充满数字的方块路中，假设这条路由一个非负的整数数组`m`组成，Drizzle 最开始的位置在下标`start`处，当他位于下标`i`位置时可以向前或者向后跳跃`m[i]`步数，已知元素值为`0`处的位置是出口，且只能通过出口出去，不可能数组越界，请你通过编程计算出Drizzle能否逃出这里。

要求:

输入：第一行输入数组`m`的长度`n` 第二行输入数组元素，空格分割开 第三行输入起始下标`start`

示例:

输入:

```
7
4 2 3 0 3 1 2
5
```

输出:

```
True
```

范围:

- 1 <= m.length <= 5 * 10^4
- 0 <= m[i] < m.length
- 0 <= start < m.length

代码长度限制	16 KB
时间限制	400 ms
内存限制	64 MB

```
#include<bits/stdc++.h>
using namespace std;
const int N=5*1e4;
int n,start,flag=0;
int m[N];
```

```
bool vis[N];
void dfs(int x)
{
    if(m[x]==0)flag=1;
    vis[x]=true;
    for(int i=1;i>=-1;i-=2)
    {
        int next=x+i*m[x];
        if(vis[next]==false&&next>=0&&next<n)
        {
            dfs(next);
            vis[next]=false;
        }
    }
}
int main()
{
    cin>>n;
    for(int i=0;i<n;i++)
        cin>>m[i];
    cin>>start;
    dfs(start);
    if(flag)cout<<"True";
    else cout<<"False";
}
```

【8】P1141 01迷宫

P1141 01迷宫

提交答案 加入题单

题目描述

复制Markdown 展开

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

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

输入格式

第1行为两个正整数n, m。

下面n行，每行n个字符，字符只可能是0或者1，字符之间没有空格。

接下来m行，每行2个用空格分隔的正整数i, j，对应了迷宫中第i行第j列的一个格子，询问从这一格开始能移动到多少格。

输出格式

m行，对于每个询问输出相应答案。

输入输出样例

输入 #1 复制

```
2 2
01
10
1 1
2 2
```

输出 #1 复制

```
4
4
```

说明/提示

所有格子互相可达。

对于20%的数据，$n \leq 10$;

对于40%的数据，$n \leq 50$;

对于50%的数据，$m \leq 5$;

对于60%的数据，$n \leq 100, m \leq 100$;

对于100%的数据，$n \leq 1000, m \leq 100000$。

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

```
int k=0;//保存当前是在第几个连通图中
int mp[1005][1005];//地图
int ans[1000005];//记录答案的数组
int vis[1005][1005];//记录
int x_plus[4]={0,0,1,-1},y_plus[4]={1,-1,0,0};
struct coordinate
{
        int x;
        int y;
};
queue<coordinate> q;
coordinate cdt1,cdt2;
void bfs(int x,int y)
{
        cdt1.x=x,cdt1.y=y;
        int sum=1;//从这个点出发能到达其他点的数目
        q.push(cdt1);
        while(!q.empty())
        {
                cdt2=q.front();
                q.pop();
                for(int i=0;i<4;i++)
                {
                        int x_next=cdt2.x+x_plus[i];
                        int y_next=cdt2.y+y_plus[i];
                        if(x_next>=1&&x_next<=n&&y_next>=1&&y_next<=n&&vis[x_next]
[y_next]==-1&&(mp[x_next][y_next]==1-mp[cdt2.x][cdt2.y]))
                        {
                                sum++;
                                vis[x_next][y_next]=k;//所有被标记为k的点具有相同答案
                                cdt1.x=x_next;
                                cdt1.y=y_next;//更新位置
                                q.push(cdt1);
                        }
                }
        }
        ans[k]=sum;//将标记为k的点能到达其他点的个数存入ans数组（保存当前连通图能到多少
格）
}
int main()
{
        cin>>n>>m;
        char str[1005];
        for(int i=0;i<n;i++)//初始化地图
        {
                cin>>str;
                for(int j=0;j<n;j++)
                        mp[i+1][j+1]=str[j]-'0';
        }
        memset(vis,-1,sizeof(vis));
        for(int i=1;i<=n;i++)
        {
                for(int j=1;j<=n;j++)
                {
                        if(vis[i][j]==-1)//如果这个点没到过就对他bfs
                        {
                                vis[i][j]=k;
```

```
                                bfs(i,j);
                                k++;
                        }
                }
        }
        int x,y;
        for(int i=0;i<m;i++)
        {
                cin>>x>>y;
                cout<<ans[vis[x][y]]<<endl;//
        }
}
```