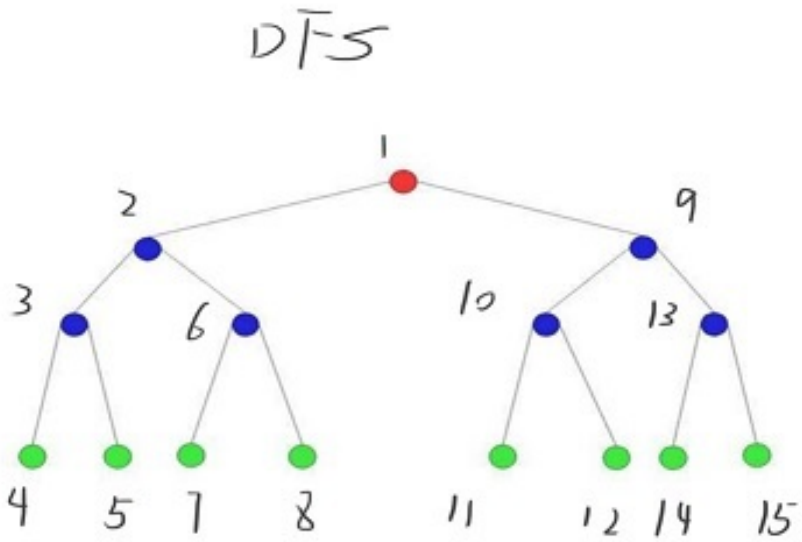
DFS(深度优先搜索)

深度优先搜索是一种用于**遍历**图的算法 。这个算法会尽可能深的搜索树的分支。其过程即：从某一个状态开始，**不断地**转移状态**直到无法转移**为止，然后**回退**到前一步状态，继续转移到其他状态，如此不断重复，直到找到最终解。

深搜的步骤可分为：**1. 递归下去 2. 回溯上来** 直到到达目标。理解 **递归** 和 **回溯** 是理解DFS的关键。

深度优先搜索是图论中的经典算法，利用深度优先搜索算法可以产生目标图的 拓扑排序 表 ，利用拓扑排序表可以方便的解决很多相关的 图论 问题，如无权最长路径问题等等。



向下递归

递归指函数中再次调用该函数自身的行为，而这样的函数也叫做递归函数。

注意递归函数需要边界条件结束递归。

比如斐波那契数列0, 1, 1, 2, 3, 5...中的第x个数可使用递归函数。

```
#include<cstdio>
using namespace std;
int n;
long long fibonacci(int x){if(x==0)return 0;if(x<3)return 1;return fibonacci(x-
1)+fibonacci(x-2);}
int main(){
    scanf("%d",&n);
    printf("%lld",fibonacci(n));
    return 0;}
```

向上回溯

回溯和枚举的思想相近,但不同在于枚举是将所有的情况都列举出来以后再一一筛选。

而回溯法在列举过程如果发现当前情况根本不可能存在答案(例如：到头了没路走了),就停止后续的所有工作,返回上一步进行新的尝试。

回溯是一种算法思想，它是通过用递归这一算法结构来实现的。

模板

因为dfs题型多，解法需根据题目灵活应变，在此只给出矩阵类问题的最基础的模板

```
#include<bits/stdc++.h>
using namespace std;
int sx,sy,zx,zy,dx[4]={0,0,1,-1},dy[4]={1,-1,0,0},dt[105][105],ans;
void dfs(int x,int y){
    if(x==zx&&y==zy){ans++;return;}
    for(int i=0;i<4;i++){
        int xx=x+dx[i],yy=y+dy[i];
        if(判断条件)continue;
            dfs(xx,yy);
    }
}
int main(){
    dfs(sx,sy);
    printf("%d",ans);
    return 0;
}
```

关于存图

- 存矩阵($n \times m$)：**邻接矩阵**

```
dt[105][105];
for(int i=1;i<=n;i++)
    for(int j=1;j<=m;j++)
        scanf("");
```

- 存其他(树，图)：**链式前向星，邻接表**

链式前向星：

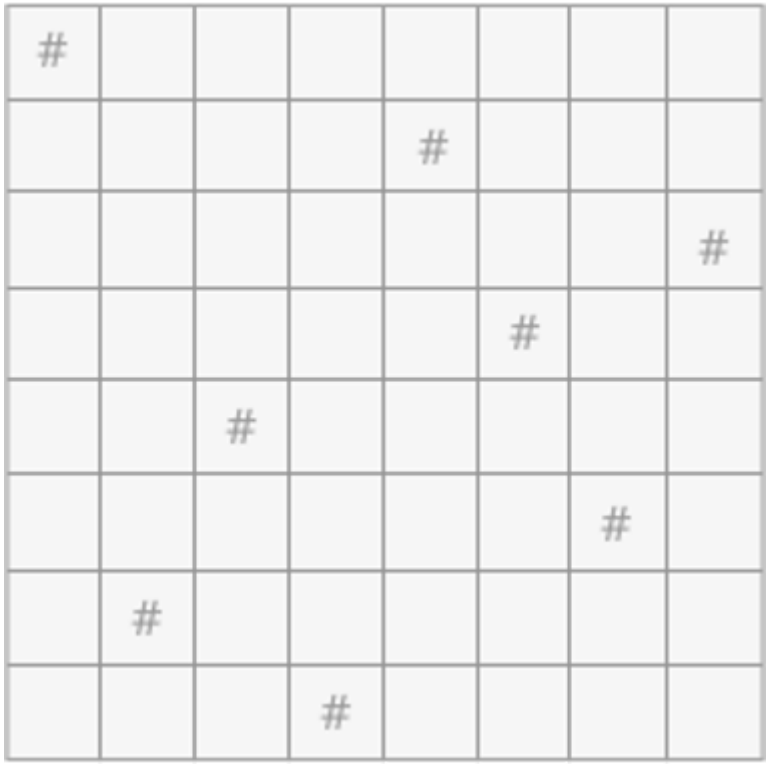
```
int h[100010],cnt;
struct edge{
    int v,w,next;
}e[100010];
void add(int u,int v,int w){//加边;
    e[++cnt].v=v;
    e[cnt].w=w;
    e[cnt].next=h[u];
    h[u]=cnt;
}
```

```
void dfs(int x){
    for(int i=h[x];i;i=e[i].next){
        //要干的事;
    }
}
int main(){
    scanf("");
    for()add(,,);
    dfs(sx);
    return 0;
}
```

例题：

八皇后问题

有八个皇后,如何在 8×8 的棋盘中放置八个皇后,使得任意两个皇后都不在同一条横线、纵线或者斜线上，打印各个方案。



思路

/1.从棋盘的第一行开始，从第一个位置开始，依次判断当前位置是否能够放置皇后，判断的依据为：同该行之前的所有行中皇后的所在位置进行比较，如果在同一列，或者在同一条斜线上（斜线有两条，为正方形的两个对角线），都不符合要求，继续检验后序的位置。

/2.如果该行所有位置都不符合要求，则回溯到前一行，改变皇后的位置，继续搜索。

/3.如果搜索到最后一行，所有皇后摆放完毕，则直接打印出 8*8 的棋盘。并将棋盘恢复原样，避免影响下一次摆放。

代码

```
#include<cstdio>
using namespace std;
int ans=0,a[100];//a[z]=i:第z个皇后放在第i行
int e[100];//e[i]=0 第i行没放皇后
int r[100];//r[k]=0 第k条副对角线(/)没放皇后
int j[100];//j[k]=0 第k条主对角线(\)没放皇后
void fang(int z){
    if(z>8){
        ans++;
        for(int i=1;i<=8;i++)
        printf("%d ",a[i]);
        printf("\n");
        return;
    }
    else{
    for(int i=1;i<=8;i++){
        if((!e[i])&&(!r[i+z])&&(!j[z-i+n])){
            a[z]=i;
            e[i]=1;
            r[z+i]=1;
            j[z-i+n]=1;
        fang(z+1);
        e[i]=0;
        r[i+z]=0;
        j[z-i+n]=0;}
    }
}
}
int main(){
    fang(1);
    return 0;
}
```

迷宫

题目背景

展开

给定一个N*M方格的迷宫，迷宫里有T处障碍，障碍处不可通过。给定起点坐标和终点坐标，问: 每个方格最多经过1次，有多少种从起点坐标到终点坐标的方案。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。

题目描述

无

输入格式

第一行N、M和T，N为行，M为列，T为障碍总数。第二行起点坐标SX,SY，终点坐标FX,FY。接下来T行，每行为障碍点的坐标。

输出格式

给定起点坐标和终点坐标，问每个方格最多经过1次，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

【数据规模】

1≤N,M≤5

代码

```
#include<iostream>
#include<cstdlib>
#include<cstdio>
#include<cmath>
using namespace std;
int mp[6][6];//地图；
bool temp[6][6];//走过的标记；
int dx[4]={0,0,1,-1};//打表；
int dy[4]={-1,1,0,0};//打表；
int total,fx,fy,sx,sy,T,n,m,l,r;//total计数器，fx，fy是终点坐标，sx，sy是起点坐标，T是障碍总数，n，m是地图的长和宽，l，r是障碍的横坐标和纵坐标；
void walk(int x,int y)//定义walk；
{
    if(x==fx&&y==fy)//fx表示结束x坐标，fy表示结束y坐标；
```

```
    {
        total++;//总数增加；
        return;//返回，继续搜索；
    }
    else
    {
        for(int i=0;i<=3;i++)//0--3是左，右，下，上四个方向；
        {
            if(temp[x+dx[i]][y+dy[i]]==0&&mp[x+dx[i]][y+dy[i]]==1)//判断没有走过和
没有障碍；
            {
                temp[x][y]=1;//走过的地方打上标记；
                walk(x+dx[i],y+dy[i]);
                temp[x][y]=0;//还原状态；
            }
        }
    }
}
int main()
{
    cin>>n>>m>>T;//n，m长度宽度，T障碍个数
    for(int ix=1;ix<=n;ix++)
        for(int iy=1;iy<=m;iy++)
            mp[ix][iy]=1;//把地图刷成1；
    cin>>sx>>sy;//起始x，y
    cin>>fx>>fy;//结束x，y
    for(int u=1;u<=T;u++)
    {
        cin>>l>>r;//l，r是障碍坐标；
        mp[l][r]=0;
    }
    walk(sx,sy);
    cout<<total;//输出总数；
    return 0;
}
```

DFS非递归

DFS的搜索顺序是先进后出与栈相同，因此可以用栈实现非递归版本的dfs。

写法是先定义一个栈,然后找到与最先出发点的所有邻接点,将他们入栈,同时标记这些点已被访问过。后面先是栈顶元素出栈重复上述步骤直至栈空。

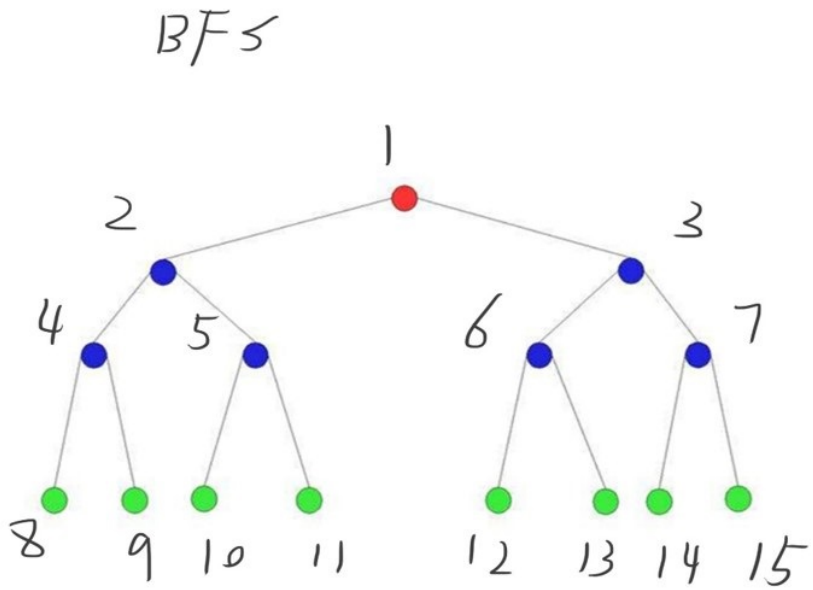
那么队列呢?

BFS(广度优先搜索)

广搜目的是系统地展开并检查图中的所有节点，以找寻结果。

广搜在面临一个路口时，将所有的岔路口都标记存储起来，然后选择其中一个进入，然后将它的分路情况记录下来，然后再返回来进入另外一个岔路，并重复这样的操作。一般用**队列**结构实现。

一般可以用它做什么呢？一个最直观经典的例子就是走迷宫，我们从起点开始，找出到终点的**最短路程**，很多最短路径算法(例如Dijkstra)就是基于广度优先的思想成立的。



回顾队列

队列是一种特殊的线性表，特殊之处在于它只允许在表的前端进行删除操作，而在表的后端进行插入操作，即“先进先出”。

使用时需要头文件$\#include<queue>$

定义一个队列：$queue<int>q;$

入队：$q.push(a)$

出队：$q.pop()$

获得队首元素：$a=q.front()$

判断队列是否为空：$q.empty()==0/q.empty()!=0$

模板

```
#include<cstdio>
#include<queue>
using namespace std;
queue<int>qx;
queue<int>qy;
int wx[5]={1,-1,0,0};
int wy[5]={0,0,-1,-1};
int main(){
    int dx,dy;
    map[x][y]=1;
    qx.push(x);
    qy.push(y);
    while(!qx.empty()){
        int a=qx.front();
        int b=qy.front();
        for(int i=0;i<4;i++){
            dx=a+wx[i];
            dy=b+wy[i];
            if(判断条件)continue;
            map[dx][dy]=1
            qx.push(dx);
            qy.push(dy);
        }
        qx.pop();
        qy.pop();
    }
}
```

例题

马的遍历

题目描述

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（左对齐，宽 5 格，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 $1 \leq x \leq n \leq 400$，$1 \leq y \leq m \leq 400$。

代码

```
#include<cstdio>
#include<queue>
using namespace std;
queue<int>qx;
queue<int>qy;
int qda,qdb,n,m,s;
int dt[501][501];
bool w[501][501];
int wx[9]={1,-1,1,-1,2,-2,2,-2};
int wy[9]={2,2,-2,-2,1,1,-1,-1};
void bfs(int x,int y){
    int dx,dy;
    qx.push(x);
    qy.push(y);
    while(!qx.empty()){
        int a=qx.front();
        int b=qy.front();
        s=dt[a][b];
        for(int i=0;i<8;i++){
            dx=a+wx[i];
            dy=b+wy[i];
```

```
            if(dx<=0||dx>n||dy<=0||dy>m)continue;
            if(dt[dx][dy]>=s+1||dt[dx][dy]==-1){
                dt[dx][dy]=s+1;
                if(w[dx][dy]==0){
                    qx.push(dx);
                    qy.push(dy);
                    w[dx][dy]=1;
                  }
            }
        }
        qx.pop();
        qy.pop();
    }
}
int main(){
    scanf("%d%d%d%d",&n,&m,&qda,&qdb);
    for(int i=1;i<=n;i++)
        for(int j=1;j<=m;j++)
            dt[i][j]=-1;
    dt[qda][qdb]=0;
    bfs(qda,qdb);
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++)
            printf("%-5d",dt[i][j]);
        printf("\n");
    }
    return 0;
}
```