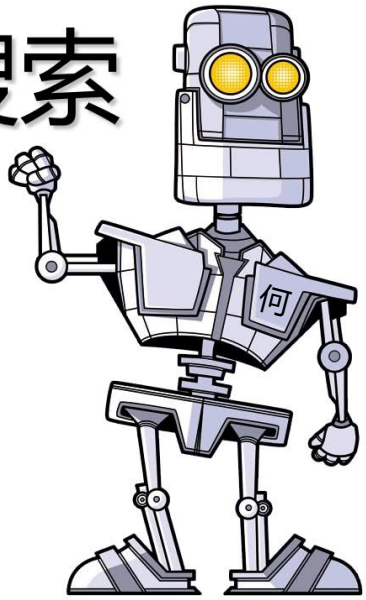
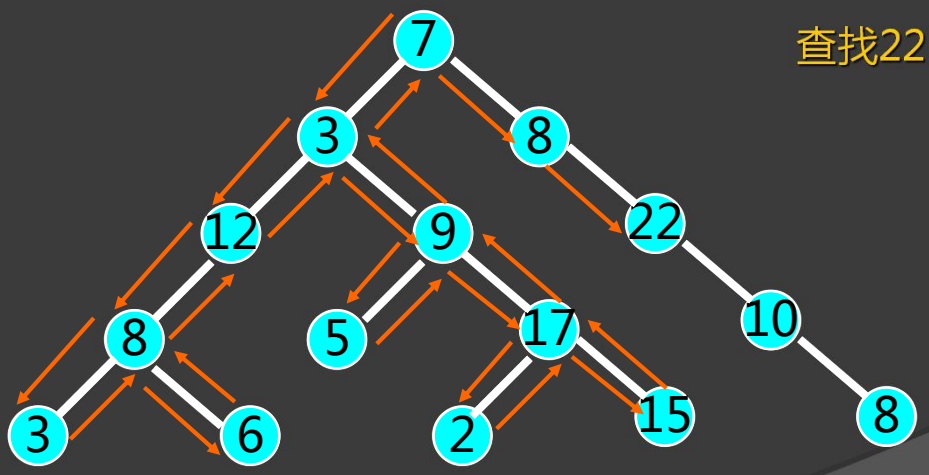
广度优先搜索

Breadth First Search (BFS)



深度优先搜索

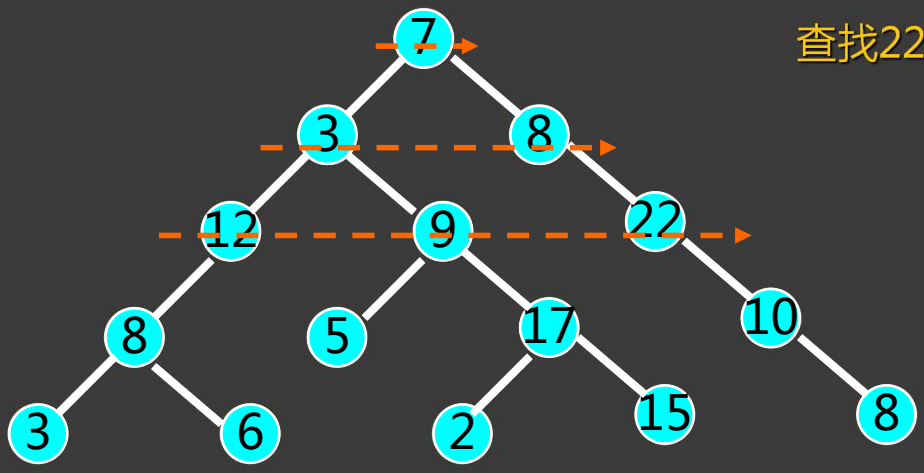
从问题的某一种可能出发，搜索从这种情况出发所能达到的所有可能，当这一条路走到“尽头”而没找到解时，再**倒回**上一个步，从另一个可能出发，继续搜索.



一直走到底，走不通就掉头试下一条路。

广度（宽度）优先搜索

从问题的起点出发，逐层搜索所有的点.



由于是逐层搜索，所以它总能保证找出的是离起点最近的点。所以比较适合用于求解最优值的问题。

自然界中的宽搜

课后习题：追牛nkoj1088

农民约翰在清点牛棚里的牛时，发现少了一头牛。他想尽快把牛找回来。约翰在离牛棚N米的地方通过望远镜发现了那头牛，那头牛正在距离牛棚K米的地方吃草(你可以理解为：牛棚、约翰和牛在一条直线上)。

农民约翰可以通过两种方法去追牛：步行和瞬间移动。如果约翰所在的点离牛棚的距离为X，那么:

*步行：用一分钟，约翰可以走到点X-1或点X+1

*瞬间移动：用一分钟，约翰可以移动到点2*X

牛一直在原地吃草，不会移动。约翰追到牛最少需要多少分钟？

(K<N的情况是可能的——何某注)

输入格式：

一行，空格间隔的两个整数N和K(0<=N,K<=100000)

输出格式：

一行，一个整数，即最短时间。

样例输入：

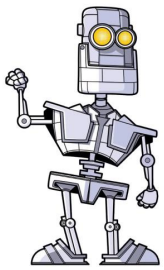
5 17

样例输出：

4

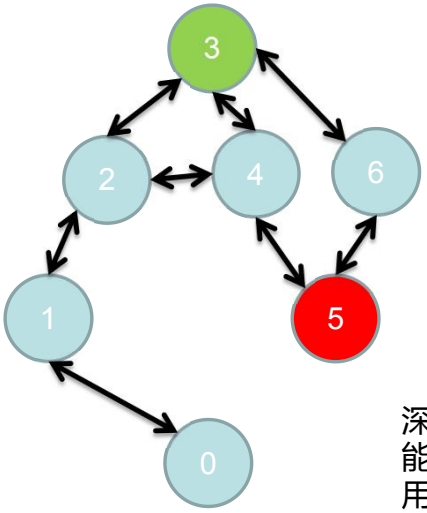
样例说明：

追上牛最快的方式是5->10->9->18->17



假设农夫起始位置位于点3，牛起始位置位于5，即N=3，K=5，最右边是6。如何搜索到一条走到5的路径。

方法一：深搜

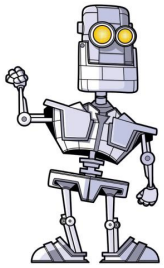


好运气：
3—>4—>5

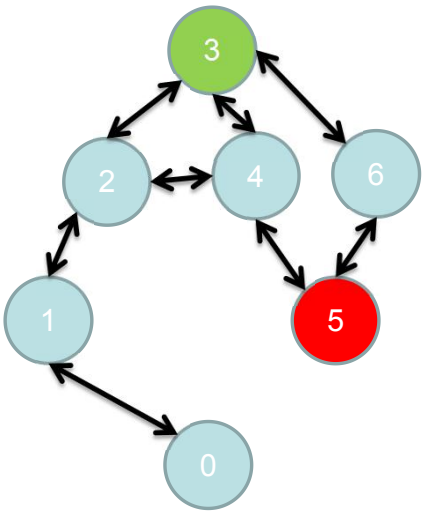
坏运气：
3—>2—>1—>0—>1—>2—>4—>5

求最短？

深搜：需要把所有情况都遍历完，才能确定哪一种方案最短。当然，可以用剪枝策略来减少一些搜索步骤。但仍然很费时间。



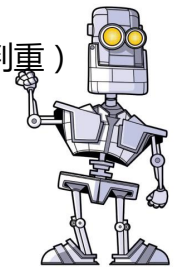
假设农夫起始位置位于点3，牛起始位置位于5，
即N=3，K=5，最右边是6。如何搜索到一条
走到5的路径。



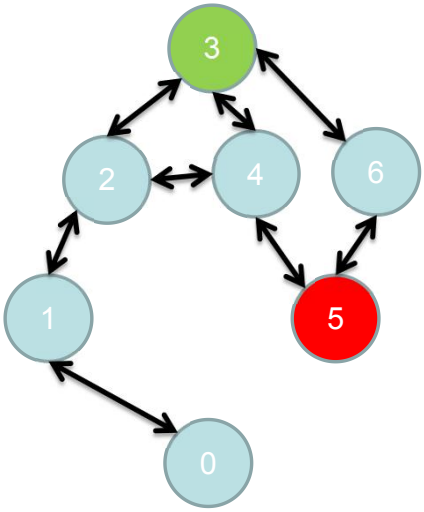
方法二：广搜（宽搜）

按步数给节点分层：
起点（走0步）：3
第1层（走1步）：2，4，6
第2层（走2步）：1，5
第3层（走3步）：0

由低层到高层逐步搜索：
3—>2—>4—>6—>1—>5
问题解决。
注意：走过的点不重复走。（判重）

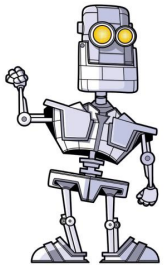


假设农夫起始位置位于点3，牛起始位置位于5，即N=3，K=5，最右边是6。如何搜索到一条走到5的路径。



广搜的优势：
可确保找到一定是最优解（最短路径），用时短。

广搜的劣势：
因为是逐步扩展的过程，扩展出来的节点较多，并且都需要保存这些节点，因此需要的储空间较大。耗空间。



广度优先搜索算法如下：（用QUEUE）

((1))把初始节点S0放入QUEUE中；

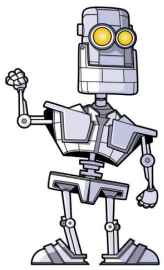
(2) 如果QUEUE为空，则问题无解，失败退出；

(3) 把QUEUE的第一个节点取出；

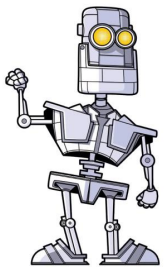
(4) 考察取出的节点是否为目标节点。若是，则得到问题的解，成功退出；

(5) 若不是目标节点，则判断该节点是否，若不可扩展，则转第(2)步；

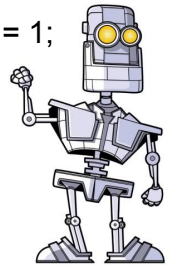
(6) 若可扩展，判断扩展节点是否已经访问过了，若不是，则放入QUEUE的尾部，然后转第(2)步。



```
#include <iostream>
#include <cstring>
#include <queue>
using namespace std;
const int MAXN = 100000;
int N,K,visited[MAXN+10]; //判重标记,visited[i] = true表示i已经扩展过
struct Node{
        int x;  //位置
        int steps; //到达x所需的步数};
queue<Node> q;  //队列
int main()   {
        cin >> N >> K;
        memset(visited,0,sizeof(visited));
        Node s0;
        s0.x=N;
        s0.steps=0;
        q.push(s0);
        visited[N] = 1;
        while(!q.empty()) {
                Node s = q.front();
                if( s.x == K ) {//找到目标
                        cout << s.steps <<endl;
                        return 0;
                }
```



```
else {
                                        Node temp;
                                        if( s.x -1 >= 0 && !visited[s.x-1] ) {
                                                temp.x=s.x-1;
                                                temp.steps=s.steps+1;
                                                q.push(temp);
                                                visited[temp.x] = 1;
                                        }
                                        if( s.x + 1 <= MAXN && !visited[s.x+1] ) {
                                                temp.x=s.x+1;
                                                temp.steps=s.steps+1;
                                                q.push(temp);
                                                visited[temp.x] = 1;
                                        }
                                        if( s.x * 2 <= MAXN &&!visited[s.x*2]  ) {
                                                temp.x=s.x*2;
                                                temp.steps=s.steps+1;
                                                q.push(temp);
                                                visited[temp.x]  = 1;
                                        }
                                        q.pop();
                                }
                        }
                        return 0;
}
```



例：好人何老板nkoj1087

八十高龄的邓大爷在大街上摔倒了，因为众所周知的原因围观的路人都不敢去救助。恰好何老板下班路过，一向助人为乐的他赶紧抱起邓大爷往医院跑。但好心的何老板面临着一个问题，城市里面有很多医院，到底哪家医院最近呢？

城市地图用一个由数字0,1,2,3构成的n*m矩阵表示(n,m<=1000)。数字0表示可以行走的道路或空地。数字1表示邓大爷摔倒的位置。数字2表示不可通过的建筑物或障碍物。数字3表示医院。

何老板只能延上下左右四个方向移动，每走一步的距离是1。问到最近的医院需要走多少步？（地图中至少有一个可到达的医院）

输入格式：

第一行，两个空格间隔的整数n和m

接下来是一个n*m的矩阵，用空格做间隔

输出格式：

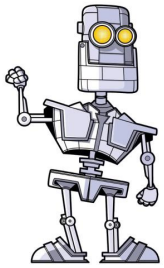
一个整数，表示最小的步数。

样例输入

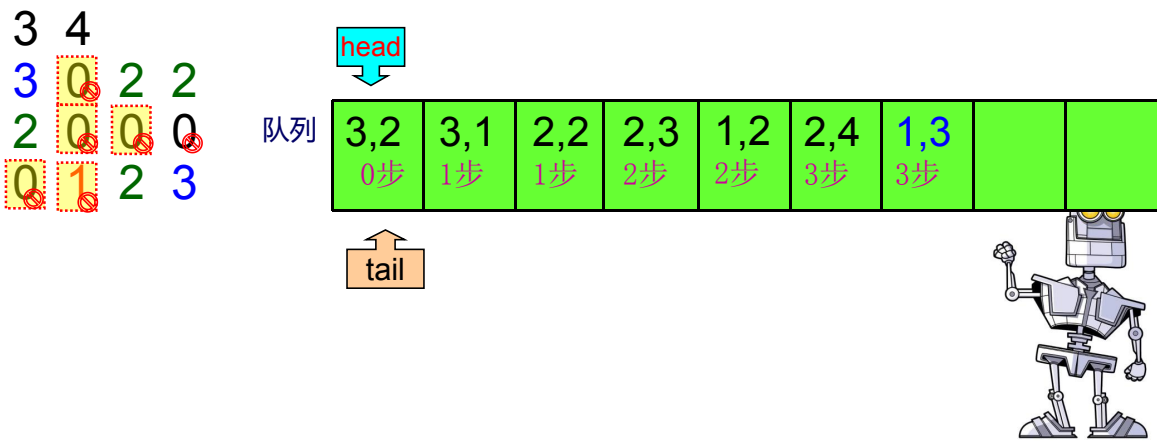
```
5 8
3 0 0 0 0 2 0 3
2 0 0 2 3 0 2 0
0 2 0 2 0 3 0 2
0 1 0 2 0 0 0 0
0 0 0 0 0 0 0 3
```

样例输出

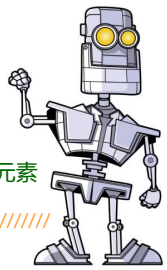
```
6
```



通过宽搜求解：
1.读入数据，找出起点坐标start.x,start.y
2.将起点加入队列。
3.讨论队首元素的上下左右四个点，如果有值为3的点(目标点)，输出结果，程序结束。否则，如果值为0(可通过的点)，则将其入队，记录该点到起点的距离，**并将该点标记为不可通过**，队尾指针后移。
4.队首指针后移，执行第3步。

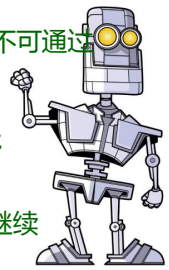


```
struct node{int x,y,step;};                          //x,y标记每个点的坐标，step记录该格子里起点的距离
int dx[4]={-1,1,0,0};   int dy[4]={0,0,-1,1};
queue<node>q;
int main()        //读入数据
{
      node head,start,tmp;
      scanf("%d%d",&n,&m);
      for(i=1;i<=n;i++)
          for(j=1;j<=m;j++)
          {    scanf("%d",&a[i][j]);
               if(a[i][j]==1){ start.x=i;  start.y=j;  start.step=0; }
          }
    ////////////////////////////////////////////////////////宽搜 ////////////////////////////////////////////////////////////////////////////
    q.push(start);
    a[start.x][start.y]=2;              //将起点标记为不可通过,避免死循环
    while(q.size())                     //反复讨论，直到找出目标。若q.size()==0表示队列为空
     {
          head=q.front();
          for(i=0;i<4;i++)                                    //讨论队首元素上下左右四个点。
          {
               tx=head.x+dx[i];       ty=head.y+dy[i];        //dx[ ]和dy[ ]为增量数组
               if((tx>0)&&(tx<=n)&&(ty>0)&&(ty<=m)&&(a[tx][ty]!=2)) //边界判断和可行判断
               {
                    if(a[tx][ty]==3)                //一旦找到目标，立即输出结果，程序结束。
                    {
                         printf("%d\n",head.step+1);
                         return 0;
                    }
                    tmp.x=tx;    tmp.y=ty;                    //tmp记录新到达的点的信息
                    tmp.step=head.step+1;                     //记录新到的点离起点的距离
                    q.push(tmp);                             //将新到达的点tmp加入队列
                    a[tx][ty]=2;                              //标记为不可通过，避免死循环
               }
          }
          q.pop();                    //讨论完当前队首元素后，删除当前队首，循环继续讨论新的队首元素
     }
   ////////////////////////////////////////////////////////宽搜////////////////////////////////////////////////////////////////////////////
}
```



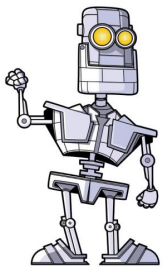
手工队列版本的代码

```
#define maxn 2001
using namespace std;
struct node{int x,y,step;};          //x,y标记每个点的坐标，step记录该格子里起点的距离
node q[maxn*maxn+1],start;    //start记录起点的坐标
int a[maxn][maxn],n,m,i,j,head,tail,tx,ty;
int dx[4]={-1,1,0,0};   int dy[4]={0,0,-1,1};
int main()                                  //读入数据
{      scanf("%d%d",&n,&m);
       for(i=1;i<=n;i++)
           for(j=1;j<=m;j++)
           {     scanf("%d",&a[i][j]);
                 if(a[i][j]==1){ start.x=i;  start.y=j; }
           }
     head=1;    tail=2;              //初始化队列，将起点入队
     q[head].x=start.x;  q[head].y=start.y;   q[head].step=0;
     a[start.x][start.y]=2;
     while(true)                    //反复讨论，直到找出目标位置。题目已告知至少有一个可到达的目标
     {     for(i=0;i<4;i++)         //讨论队首元素上下左右四个点。
           {
                 tx=q[head].x+dx[i];
                 ty=q[head].y+dy[i];
                 if((tx>0)&&(tx<=n)&&(ty>0)&&(ty<=m)&&(a[tx][ty]!=2))       //边界判断和可行判断
                 {       if(a[tx][ty]==3)                          //一旦找到目标，立即输出结果，程序结束。
                         {       printf("%d\n",q[head].step+1);
                                 return 0;
                         }                                         //将新的点入队，并将其标记为不可通过
                         q[tail].x=tx;    q[tail].y=ty;
                         q[tail].step=q[head].step+1;              //记录节点离起点的距离
                         a[tx][ty]=2;                             //标记为不可通过，避免死循环
                         tail++;                    //队列每加入一个元素后，记得将队尾指针后移
                 }
           }
           head++;  //讨论完当前队首元素后，队首指针后移，指向新的队首元素，循环继续
      }return 0;
}
```



宽搜的特点：**省时间、耗空间**

运用宽搜需注意的几点：
1.正确估算宽搜队列的长度,如果搜索的结点太多，那不应选择宽搜。
2.别忘了更新tail++和head++
3.注意判重，避免死循环



细胞分裂nkoj1086

在培养皿中有很多细胞，相邻的细胞都是由同一个细胞分裂出来的，称为细胞团。问：这个培养皿最初有多少个细胞。

一矩形阵表示培养皿，里面有数字0到9,数字1到9代表细胞,相邻的细胞都是由同一个细胞分裂出来的（一个细胞只能在上下和左右方向上分裂），求给定矩形阵中细胞最初个数。

输入格式：

第一行两个整数n,m(<=500)

接下来是一个n行m列的矩阵（**无间隔符号**）

输出格式：一个整数，表示结果

样例输入：

```
4 10
0 0 0 0 0 0 0 0 6 7
1 0 0 0 0 0 0 5 0 0
2 0 0 0 0 0 0 6 7 0
0 0 0 0 0 0 0 0 8 0
```

样例输出：

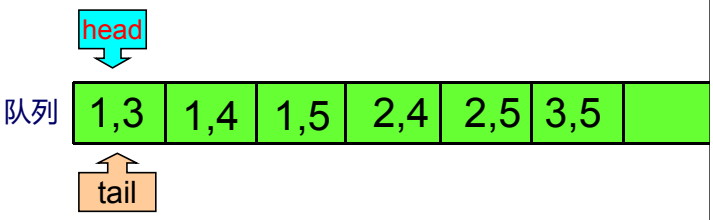
```
4
```

1.将矩阵存入二维字符数组a

2.从左到右，从上到下依次讨论每一个细胞a[x][y]。如果a[x][y]=='0'，跳过。否则表示找到了一个细胞团，统计最初细胞个数的计数器ans++。接着讨论哪些细胞属于该细胞团，并把他们全部改为'0'

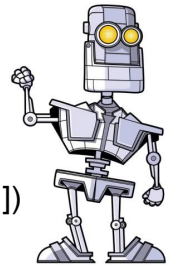
```
for(x=1;i<=n;x++)
    for(y=1;y<=m;y++)if(a[x][y]!='0').....
```

怎样判断哪些细胞属于同一个细胞团？

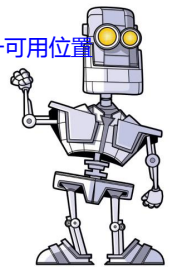


读入字符矩阵的方法：

```
char a[501][501];
.....
scanf("%d%d",&n,&m);
for(i=1;i<=n;i++)scanf("%s",&a[i][1])
```

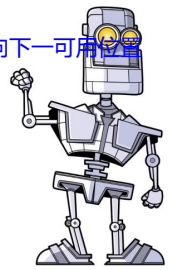


```
struct node{
      int x,y;  };
queue<node>q;
node tmp;
int dx[4]={-1,1,0,0};                          //dx,dy为增量数组，用于便捷讨论上下左右四个点
int dy[4]={0,0,-1,1};
int main()
{   scanf("%d%d",&n,&m);
    for(i=1;i<=n;i++)scanf("%s",&a[i][1]);   //读入二维字符数组
    for(i=1;i<=n;i++)
      for(j=1;j<=m;j++)
         if(a[i][j]!='0')                          //找到一个不为‘0’的点，把它存入队列队首
         {
             ans++;                                //结果计数器加1
             tmp.x=i;   tmp.y=j;  q.push(tmp);  //把点(i,j)存入队首
             a[i][j]='0';
             while(q.size())                       //当队列不为空，继续讨论
             {
                 tx=q.front().x; ty=q.front().y;     //取出队首元素，存于(tx,ty)中，讨论与它相关联的点。
                 q.pop();
                 for(k=0;k<4;k++)                    //讨论(tx,ty)上下左右四个点是否为'0'
                 {
                     xx=tx+dx[k];                    //(xx,yy)用来记录跟(tx,ty)相关联的上下左右的点
                     yy=ty+dy[k];
                     if((xx>0)&&(xx<=n)&&(yy>0)&&(yy<=m)&&(a[xx][yy]!='0'))
                     {    tmp.x=xx; tmp.y=yy; //如果点(xx,yy)不为‘0’,则加入队列，tail指向下一可用位置
                          a[xx][yy]='0';
                          q.push(tmp);
                     }
                 }
              }
         }
    printf("%d\n",ans);  return 0;
}
```



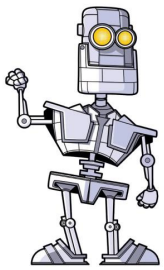
手工队列版本的代码

```
struct node{
      int x,y;  };
node q[250001];
int n,m,i,j,k,head,tail,tx,ty,xx,yy,ans=0;
char a[501][501];
int dx[4]={-1,1,0,0};                    //dx,dy为增量数组，用于便捷讨论上下左右四个点
int dy[4]={0,0,-1,1};
int main()
{   scanf("%d%d",&n,&m);
    for(i=1;i<=n;i++)scanf("%s",&a[i][1]);  //读入二维字符数组
    for(i=1;i<=n;i++)
      for(j=1;j<=m;j++)
        if(a[i][j]!='0')                    //找到一个不为‘0’的点，把它存入队列队首
        {   ans++;                          //结果计数器加1
            head=1;  tail=2;                //tail指向下一个可用的队列位置
            q[head].x=i;   q[head].y=j;     //把点(i,j)存入队首
            a[i][j]='0';
            while(head!=tail)               //当队列不为空，继续讨论
            {   tx=q[head].x; ty=q[head].y;  //取出队首元素，存于(tx,ty)中，讨论与它相关联的点。
                head++;
                for(k=0;k<4;k++)            //讨论(tx,ty)上下左右四个点是否为'0'
                {
                    xx=tx+dx[k];            //(xx,yy)用来记录跟(tx,ty)相关联的上下左右的点
                    yy=ty+dy[k];
                    if((xx>0)&&(xx<=n)&&(yy>0)&&(yy<=m)&&(a[xx][yy]!='0'))
                    {   q[tail].x=xx;  q[tail].y=yy; //如果点(xx,yy)不为‘0’,则加入队列，tail指向下一可用位置
                        a[xx][yy]='0';
                        tail++;
                    }
                }
             }
        }
    printf("%d\n",ans);  return 0;
}
```



总结

广度(宽度)优先搜索的步骤：
1.清空一个队列(head=tail=1),从某个点出发开始搜索，把该点装入队列首部；
2.取出队首元素进行讨论；
3.把与队首元素相关的点装入队列(若找到目标点，则结束搜索),队尾指针后移tail++；
4.队首指针后移(head++)，若队列为空(head==tail)，搜索结束，未找到目标。否则重复第2步；



课后习题：南渝宽搜，密码：bfsbfs