BFS与DFS的讨论：

BFS：这是一种基于队列这种数据结构的搜索方式，它的特点是由每一个状态可以扩展出许多状态，然后再以此扩展，直到找到目标状态或者队列中头尾指针相遇，即队列中所有状态都已处理完毕。  
DFS：基于递归的搜索方式，它的特点是由一个状态拓展一个状态，然后不停拓展，直到找到目标或者无法继续拓展结束一个状态的递归。  
优缺点：

BFS:对于解决最短或最少问题特别有效，而且寻找深度小，但缺点是内存耗费量大（需要开大量的数组单元用来存储状态）。  
DFS：对于解决遍历和求所有问题有效，对于问题搜索深度小的时候处理速度迅速，然而在深度很大的情况下效率不高

# 1181 变形课

**Problem Description**

呃......变形课上Harry碰到了一点小麻烦,因为他并不像Hermione那样能够记住所有的咒语而随意的将一个棒球变成刺猬什么的,但是他发现了变形咒语的一个统一规律:如果咒语是以a开头b结尾的一个单词,那么它的作用就恰好是使A物体变成B物体.   
Harry已经将他所会的所有咒语都列成了一个表,他想让你帮忙计算一下他是否能完成老师的作业,将一个B(ball)变成一个M(Mouse),你知道,如果他自己不能完成的话,他就只好向Hermione请教,并且被迫听一大堆好好学习的道理.

**Input**

测试数据有多组。每组有多行，每行一个单词,仅包括小写字母,是Harry所会的所有咒语.数字0表示一组输入结束.

**Output**

如果Harry可以完成他的作业,就输出"Yes.",否则就输出"No."(不要忽略了句号)

**Sample Input**

so

soon

river

goes

them

got

moon

begin

big

0

**Sample Output**

Yes.

***Hint***

Hint

Harry 可以念这个咒语:"big-got-them".

代码清单：

//广搜

#include<iostream>

#include<queue>

#include<cstring>

using namespace std;

const int maxn=1000+10;

int used[maxn];

struct node//定义好结构体数组

{

char head;

char rear;

}a[maxn];

int main()

{

char ch[maxn];

node b;

queue<struct node> q;

while(cin>>ch)

{

memset(used,0,sizeof(used));

int len;

int i=0;

if(ch[0]!='0')//以0结束要用这样的格式

{

len=strlen(ch);

a[i].head=ch[0];

a[i].rear=ch[len-1];

i++;

}

while(cin>>ch)

{

if(ch[0]!='0')

{

len=strlen(ch);

a[i].head=ch[0];

a[i].rear=ch[len-1];

i++;

}

else

break;

}

for(int j=0;j<i;j++)

{

if(a[j].head=='b')

{

used[j]=1;//标记不用二维的数组，可以与a[j]同步，刚开始用used[a[j].top][a[j].rear]标记，这样标记有错误

q.push(a[j]);

}

}

while(!q.empty())

{

b = q.front();

q.pop();

if(b.rear == 'm')

{

cout<<"Yes."<<endl;

break;//在这里判断完要直接推出循环

}

else

{

for(int j=0;j<i;j++)

{

if(a[j].head == b.rear && !used[j])//没用!used[j]会超时

{

used[j]=1;

q.push(a[j]);

}

}

}

}

if(b.rear != 'm')

cout<<"No."<<endl;

}

return 0;

}

//深搜

#include<iostream>

#include<cstring>

using namespace std;

const int maxn = 1000 + 10;

int used[maxn];

char ch[maxn];

int i;

struct node

{

char head,rear;

}a[maxn];

int flag ;

void dfs(int j)

{

if(a[j].rear == 'm')//输出No和Yes的方法 因为No就是没进入过这条函数 所以flag = 0

{

flag = 1 ;

return ;

}

for(int k=0;k<i;k++)

{

if(a[j].rear == a[k].head && used[k] == 0)

{

used[k] = 1;

dfs(k);

// used[k] = 0;

//这种情况不用把标记还原为0，因为下次如果再找到这个字母，接下去的路也是走不通的

}

}

}

int main()

{

while(cin>>ch )

{

i=0;

if(ch[0] != '0')

{

int len = strlen(ch);

a[i].head = ch[0];

a[i].rear = ch[len-1];

i++;

}

while(cin>>ch)

{

if(ch[0] != '0')

{

int len = strlen(ch);

a[i].head = ch[0];

a[i].rear = ch[len-1];

i++;

}

else

break;

}

flag = 0;

for(int j=0;j<i;j++)

{

if(a[j].head == 'b')

{

memset(used, 0, sizeof(used));

used[j] = 1;

dfs(j);

//used[j] = 0;

}

}

if(!flag)

cout<<"No."<<endl;

else

cout<<"Yes."<<endl;

}

return 0;

}

# 1241 Oil Deposits

**Problem Description**

The GeoSurvComp geologic survey(调查) company is responsible for detecting( 探测) underground oil deposits(  寄存处;   储蓄;  ). GeoSurvComp works with one large rectangular region(地带) of land at a time, and creates a grid(格子) that divides the land into numerous square plots. It then analyzes each plot separately, using sensing equipment to determine whether or not the plot contains oil. A plot containing oil is called a pocket. If two pockets are adjacent, then they are part of the same oil deposit. Oil deposits can be quite large and may contain numerous pockets. Your job is to determine how many different oil deposits are contained in a grid.

**Input**

The input file contains one or more grids. Each grid begins with a line containing m and n, the number of rows and columns in the grid, separated by a single space. If m = 0 it signals the end of the input; otherwise 1 <= m <= 100 and 1 <= n <= 100. Following this are m lines of n characters each (not counting the end-of-line characters). Each character corresponds to one plot, and is either `\*', representing the absence of oil, or `@', representing an oil pocket.

**Output**

For each grid, output the number of distinct oil deposits. Two different pockets are part of the same oil deposit if they are adjacent horizontally, vertically, or diagonally. An oil deposit will not contain more than 100 pockets.

**Sample Input**

1 1

\*

3 5

\*@\*@\*

\*\*@\*\*

\*@\*@\*

1 8

@@\*\*\*\*@\*

5 5

\*\*\*\*@

\*@@\*@

\*@\*\*@

@@@\*@

@@\*\*@

0 0

**Sample Output**

0

1

2

2

代码清单：

//dfs深搜

#include <iostream>

#include <stdio.h>

#include <string.h>

#include <queue>

using namespace std;

const int maxn=105;

char mat[maxn][maxn];

int dir[8][2]={{1,0},{-1,0},{0,1},{0,-1},{1,1},{-1,1},{1,-1},{-1,-1}};

int m,n;

int sum;

void Dfs(int x,int y){

int tx;

int ty;

for(int i=0;i<8;i++){

tx=x+dir[i][0];

ty=y+dir[i][1];

if(1<=tx&&tx<=m&&1<=ty&&ty<=n){

//注意！对于二维数组来说，x行y列，意味着x是纵向坐标，y是横向坐标

if(mat[tx][ty]=='@'){

mat[tx][ty]='\*';

Dfs(tx,ty);

}

}

}

}

int main()

{

while(scanf("%d%d",&m,&n)){

if(m==0&&n==0){

break;

}

//memset(mat,0,sizeof(mat));

sum=0;

for(int i=1;i<=m;i++){

for(int j=1;j<=n;j++){

//scanf("%s",mat[i]+1);

cin>>mat[i][j];

//scanf("%c",&mat[i][j]);

}

}

for(int i=1;i<=m;i++){

for(int j=1;j<=n;j++){

if(mat[i][j]=='@'){

sum++;

mat[i][j]='\*';

Dfs(i,j); //深搜，消除同一连通块的所有油田标记

}

}

}

cout<<sum<<endl;

}

return 0;

}

//BFS广搜

#include <iostream>

#include <stdio.h>

#include <string.h>

#include <queue>

using namespace std;

const int maxn=105;

char mat[maxn][maxn];

int dir[8][2]={{1,0},{-1,0},{0,1},{0,-1},{1,1},{-1,1},{1,-1},{-1,-1}};

int  m,n;

int sum;

struct Node{

    int x;

    int y;

};

void Bfs(int x,int y){

    queue <Node> q;

    Node node;

    node.x=x;

    node.y=y;

    q.push(node);

    while(!q.empty()){

        Node cur=q.front();

        Node next;

        q.pop();

        for(int i=0;i<8;i++){

            next.x=cur.x+dir[i][0];

            next.y=cur.y+dir[i][1];

            if(mat[next.x][next.y]=='@'){

                mat[next.x][next.y]='\*';

                q.push(next);

            }

        }

    }

}

int main()

{

    while(scanf("%d %d",&m,&n)){

        if(m==0&&n==0){

            break;

        }

        memset(mat,0,sizeof(mat));

        sum=0;

        int cur=1;

        for(int i=1;i<=m;i++){

            //for(int j=1;j<=n;j++){

                scanf("%s",mat[i]+1);//因为第二维也都是从1开始到n，所以+1开始

                //cin>>mat[i][j];

                //scanf("%c",&mat[i][j]);

            //}

        }

        for(int i=1;i<=m;i++){

            for(int j=1;j<=n;j++){

                if(mat[i][j]=='@'){

                    sum++;

                    mat[i][j]='\*';

                    Bfs(i,j); //广搜，消除同一连通块的所有油田标记

                }

            }

        }

        cout<<sum<<endl;

    }

    return 0;

}

# 1242 Rescue

**Problem Description**

Angel was caught by the MOLIGPY! He was put in prison by Moligpy. The prison is described as a N \* M (N, M <= 200) matrix. There are WALLs, ROADs, and GUARDs in the prison.  
  
Angel's friends want to save Angel. Their task is: approach Angel. We assume that "approach Angel" is to get to the position where Angel stays. When there's a guard in the grid, we must kill him (or her?) to move into the grid. We assume that we moving up, down, right, left takes us 1 unit time, and killing a guard takes 1 unit time, too. And we are strong enough to kill all the guards.  
  
You have to calculate the minimal time to approach Angel. (We can move only UP, DOWN, LEFT and RIGHT, to the neighbor grid within bound, of course.)

**Input**

First line contains two integers stand for N and M.  
  
Then N lines follows, every line has M characters. "." stands for road, "a" stands for Angel, and "r" stands for each of Angel's friend.   
  
Process to the end of the file.

**Output**

For each test case, your program should output a single integer, standing for the minimal time needed. If such a number does no exist, you should output a line containing "Poor ANGEL has to stay in the prison all his life."

**Sample Input**

7 8

#.#####.

#.a#..r.

#..#x...

..#..#.#

#...##..

.#......

........

**Sample Output**

13

代码清单：

//DFS

#include <cstdio>

#include <cstdlib>

#include <cstring>

#include <cmath>

#include <climits>

const int MAX = 202;

char map[MAX][MAX];

int visit[MAX][MAX];

int n,m,ax,ay,minx;

void dfs(int x,int y,int len){

    if(x<0 || y<0 || x>=n || y>=m)return;

    if(len>=minx)return;

    if(map[x][y]=='#')return;

    if(visit[x][y]==1)return;

    if(map[x][y]=='r'){

        if(len<minx)minx=len;

        return;

    }

    if(map[x][y]=='x'){

        ++len;

    }

    visit[x][y]=1;

    dfs(x+1,y,len+1);

    dfs(x-1,y,len+1);

    dfs(x,y+1,len+1);

    dfs(x,y-1,len+1);

    visit[x][y]=0;

}

int main(){

    //freopen("in.txt","r",stdin);

    int i,j,len;

    while(scanf("%d %d%\*c",&n,&m)!=EOF){

        for(i=0;i<n;++i){

            for(j=0;j<m;++j){

                map[i][j]=getchar();

                if(map[i][j]=='a'){

                    ax = i;

                    ay = j;

                }

            }

            getchar();

        }

        len = 0;

        minx = INT\_MAX;

        dfs(ax,ay,len);

        if(minx!=INT\_MAX){

            printf("%d\n",minx);

        }else{

            printf("Poor ANGEL has to stay in the prison all his life.\n");

        }

    }

    return 0;

}

//BFS和优先队列

#include<stdio.h>

#include<string.h>

#include<algorithm>

#include<queue>

#include<iostream>

#define N 300

using namespace std;

typedef struct node

{

int x,y;

int time;

friend bool operator<(node n1,node n2)

{

return n2.time<n1.time;

}

}node;

int map[N][N];

int dir[4][2]={{1,0},{-1,0},{0,1},{0,-1}};

int n,m;

int a1,b1,a2,b2;

int judge(int x,int y)

{

if(x<0||x>=m||y<0||y>=n||map[y][x]==-1)

return 1;

return 0;

}

int bfs()

{

priority\_queue<node>Q;

while(!Q.empty())

Q.pop();

node fir,nex;

fir.x=a1;

fir.y=a2;

fir.time=0;

map[a2][a1]=-1;

Q.push(fir);

while(!Q.empty())

{

fir=Q.top();

Q.pop();

if(fir.x==b1&&fir.y==b2)

{

return fir.time;

}

for(int i=0;i<4;i++)

{

nex.x=fir.x+dir[i][0];

nex.y=fir.y+dir[i][1];

if(judge(nex.x,nex.y))

continue;

nex.time=fir.time+map[nex.y][nex.x]+1;

map[nex.y][nex.x]=-1;

Q.push(nex);

}

}

return -1;

}

int main()

{

char c[N][N];

while(~scanf("%d%d",&n,&m))

{

for(int i=0;i<n;i++)

scanf("%s",c[i]);

for(int i=0;i<n;i++)

for(int j=0;j<m;j++)

{

if(c[i][j]=='.')

{

map[i][j]=0;

continue;

}

if(c[i][j]=='#')

{

map[i][j]=-1;

continue;

}

if(c[i][j]=='x')

{

map[i][j]=1;

continue;

}

if(c[i][j]=='r')

{

map[i][j]=0;

a1=j,a2=i;

continue;

}

if(c[i][j]=='a')

{

map[i][j]=0;

b1=j,b2=i;

}

}

int step=bfs();

if(step!=-1)

printf("%d\n",step);

else

printf("Poor ANGEL has to stay in the prison all his life.\n");

}

return 0;

}

# 1312Red and Black

**Problem Description**

There is a rectangular room, covered with square tiles. Each tile is colored either red or black. A man is standing on a black tile. From a tile, he can move to one of four adjacent tiles. But he can't move on red tiles, he can move only on black tiles.  
  
Write a program to count the number of black tiles which he can reach by repeating the moves described above.

**Input**

The input consists of multiple data sets. A data set starts with a line containing two positive integers W and H; W and H are the numbers of tiles in the x- and y- directions, respectively. W and H are not more than 20.  
  
There are H more lines in the data set, each of which includes W characters. Each character represents the color of a tile as follows.  
  
'.' - a black tile   
'#' - a red tile   
'@' - a man on a black tile(appears exactly once in a data set)

**Output**

For each data set, your program should output a line which contains the number of tiles he can reach from the initial tile (including itself).

**Sample Input**

6 9

....#.

.....#

......

......

......

......

......

#@...#

.#..#.

11 9

.#.........

.#.#######.

.#.#.....#.

.#.#.###.#.

.#.#..@#.#.

.#.#####.#.

.#.......#.

.#########.

...........

11 6

..#..#..#..

..#..#..#..

..#..#..###

..#..#..#@.

..#..#..#..

..#..#..#..

7 7

..#.#..

..#.#..

###.###

...@...

###.###

..#.#..

..#.#..

0 0

**Sample Output**

45

59

6

13

//DFS

#include <stdio.h>

#include <string.h>

int n,m,cnt;

char map[30][30];

int to[4][2] = {{1,0},{0,1},{-1,0},{0,-1}};

void dfs(int i,int j)

{

cnt++;

map[i][j] = '#';

for(int k = 0; k<4; k++)

{

int x = i+to[k][0];

int y = j+to[k][1];

if(x<n && y<m && x>=0 && y>=0 && map[x][y] == '.')

dfs(x,y);

}

return;

}

int main()

{

int i,j,fi,fj;

while(scanf("%d%d",&m,&n),m||n)

{

getchar();

for(i = 0; i<n; i++)

{

for(j = 0; j<m; j++)

{

scanf("%c",&map[i][j]);

if(map[i][j] == '@')

{

fi = i;

fj = j;

}

}

getchar();

}

cnt = 0;

//map[i][j] = '#';

dfs(fi,fj);

printf("%d\n",cnt);

}

return 0;

}

//BFS

#include<cstdio>

#include<queue>

using namespace std;

char a[25][25];

int x,y,sum;

int dir[4][2]={1,0,0,1,-1,0,0,-1};

struct node

{

    int x,y;

}q[500];

bool YES(int x0,int y0)//xx,yy

{

   //x--列,y--行

    if(x0<y&&x0>=0&&y0>=0&&y0<x)

        return true;

    else

        return false;

}

void BFS(int x0,int y0)

{

    queue<node>q;

    node start,endd;

    start.x=x0;

    start.y=y0;

    q.push(start);

    while(!q.empty())

    {

        start=q.front();

        q.pop();

        for(int i=0;i<4;i++)

        {

            endd.x=start.x+dir[i][0];

            endd.y=start.y+dir[i][1];

            if(YES(endd.x,endd.y)&&a[endd.x][endd.y]=='.')

            {

                a[endd.x][endd.y]='#';

                sum++;

                q.push(endd);

            }

        }

    }

}

int main()

{

    int i,j,di,dj;

    while (scanf("%d%d",&x,&y)!=EOF)

    {

        getchar();

        if (x==0&&y==0) break;

        for (i=0; i<y; i++)

        {

            for (j=0; j<x; j++)

            {

                scanf("%c",&a[i][j]);

                if(a[i][j] == '@')

                {

                    di = i;

                    dj = j;

                }

            }

            getchar();

        }

        sum=1;

        BFS(di,dj);

        printf("%d\n",sum);

    }

    return 0;

}