**jnuProblem A**

**Problem Description**

You are given a number of case-sensitive strings of alphabetic characters, find the largest string X, such that either X, or its inverse can be found as a substring of any of the given strings.

**Input**

The first line of the input file contains a single integer t (1 <= t <= 10), the number of test cases, followed by the input data for each test case. The first line of each test case contains a single integer n (1 <= n <= 100), the number of given strings, followed by n lines, each representing one string of minimum length 1 and maximum length 100. There is no extra white space before and after a string.

**Output**

There should be one line per test case containing the length of the largest string found.

**Sample Input**

2

3

ABCD

BCDFF

BRCD

2

rose

orchid

**Sample Output**

2

2

**代码：**

#include <stdio.h>

#include <string.h>

int main()

{

char string[105][105],str[105],pos[105],inv[105];

int i,j,t,n,min\_len,index;

int flag;

scanf("%d",&t);

while(t--)

{

scanf("%d",&n);

min\_len = 105;

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

{

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

if(strlen(string[i])<min\_len)

{

min\_len = strlen(string[i]);

index = i;

}

}

int len;

len = min\_len;

strcpy(str,string[index]);

while(len>0)

{

flag = 0;

for(i = 0; i <= min\_len-len;i++)

{

flag = 1;

strncpy(pos,str+i,len);

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

inv[j] = pos[len-j-1];

pos[len] = inv[len] ='\0';

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

{

if(strstr(string[j],pos)==NULL&&strstr(string[j],inv)==NULL)

{

flag = 0;

break;

}

}

if(flag)break;

}

if(flag)break;

len--;

}

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

}

return 0;

}

**Problem B**

**Problem Description**

A message from humans to extraterrestrial intelligence was sent through the Arecibo radio telescope in Puerto Rico on the afternoon of Saturday November 16, 1974. The message consisted of 1679 bits and was meant to be translated to a rectangular picture with 23 \* 73 pixels. Since both 23 and 73 are prime numbers, 23 \* 73 is the unique possible size of the translated rectangular picture each edge of which is longer than 1 pixel. Of course, there was no guarantee that the receivers would try to translate the message to a rectangular picture. Even if they would, they might put the pixels into the rectangle incorrectly. The senders of the Arecibo message were optimistic.   
We are planning a similar project. Your task in the project is to find the most suitable width and height of the translated rectangular picture. The term "most suitable" is defined as follows. An integer m greater than 4 is given. A positive fraction a / b less than or equal to 1 is also given. The area of the picture should not be greater than m. Both of the width and the height of the translated picture should be prime numbers. The ratio of the width to the height should not be less than a / b nor greater than 1. You should maximize the area of the picture under these constraints.  
  
In other words, you will receive an integer m and a fraction a / b. It holds that m > 4 and 0 < a / b < 1. You should find the pair of prime numbers p, q such that pq <= m and a / b <= p / q <= 1, and furthermore, the product pq takes the maximum value among such pairs of two prime numbers. You should report p and q as the "most suitable" width and height of the translated picture.

**Input**

The input is a sequence of at most 2000 triplets of positive integers, delimited by a space character in between. Each line contains a single triplet. The sequence is followed by a triplet of zeros, 0 0 0, which indicated the end of the input and should not be treated as data to be processed.  
  
The integers of each input triplet are the integer m, the numerator a, and the denominator b described above, in this order. You may assume 4 < m <= 100000 and 1 <= a <= b <= 1000.

**Output**

The output is a sequence of pairs of positive integers. The i-th output pair corresponds to the i-th input triplet. The integers of each output pair are the width p and the height q described above, in this order.  
  
Each output line contains a single pair. A space character is put between the integers as a delimiter. No other characters should appear in the output.

**Sample Input**

5 1 2

99999 999 999

1680 5 16

1970 1 1

2002 4 11

0 0 0

**Sample Output**

2 2

313 313

23 73

43 43

37 53

**代码：**

#include <fstream>

#include <iostream>

#include <cmath>

#include <cstring>

using namespace std;

bool isPrime[10000]={0};

int prime[10000];

int num=0;

void init(){

memset( isPrime, 0, sizeof(isPrime) );

int i, j;

double d=sqrt(10000\*1.0);

for( i=2; i<=d; i++)

for( j=2; j\*i<10000; j++)

isPrime[i\*j]=1;

for( i=2; i<10000; i++ )

if(!isPrime[i]){

prime[num] = i;

num++;

}

};

int main()

{

int n, i, j;

int m, a, b, p, q;

init();

while ( cin >> m >> a >> b && m!=0 ){

float aspect = a\*1.0/b;

int product, final = 0;

for( i=num-1; i>=0; i-- ){

for( j=num-1; j>=i; j--){

product = prime[i]\*prime[j];

if( product<=m && prime[i]\*1.0/prime[j]>=aspect ){

if( product>final ){

final = product;

p = prime[i];

q = prime[j];

}

break;

}

}

}

cout << p << " " << q << endl;

}

return 0;

}

**Problem C**

**Problem Description**

The doggie found a bone in an ancient maze, which fascinated him a lot. However, when he picked it up, the maze began to shake, and the doggie could feel the ground sinking. He realized that the bone was a trap, and he tried desperately to get out of this maze.  
  
The maze was a rectangle with sizes N by M. There was a door in the maze. At the beginning, the door was closed and it would open at the T-th second for a short period of time (less than 1 second). Therefore the doggie had to arrive at the door on exactly the T-th second. In every second, he could move one block to one of the upper, lower, left and right neighboring blocks. Once he entered a block, the ground of this block would start to sink and disappear in the next second. He could not stay at one block for more than one second, nor could he move into a visited block. Can the poor doggie survive? Please help him.

**Input**

The input consists of multiple test cases. The first line of each test case contains three integers N, M, and T (1 < N, M < 7; 0 < T < 50), which denote the sizes of the maze and the time at which the door will open, respectively. The next N lines give the maze layout, with each line containing M characters. A character is one of the following:  
  
'X': a block of wall, which the doggie cannot enter;   
'S': the start point of the doggie;   
'D': the Door; or  
'.': an empty block.  
  
The input is terminated with three 0's. This test case is not to be processed.

**Output**

For each test case, print in one line "YES" if the doggie can survive, or "NO" otherwise.

**Sample Input**

4 4 5

S.X.

..X.

..XD

....

3 4 5

S.X.

..X.

...D

0 0 0

**Sample Output**

NO

YES

**代码：**

#include <iostream>

#include <fstream>

#include <cmath>

#include <cstring>

#include <cstdio>

#include <algorithm>

using namespace std;

int dx[] = {-1,0,1,0};

int dy[] = {0,1,0,-1};

int N,M,T;

char map[10][10];

int Wall;

int di,dj;

int si,sj;

bool is;

void dfs(int si,int sj,int t)

{

if(si<1||sj<1||si>N||sj>M||t>T||is)

return;

if(si==di && sj==dj && t==T)

{

is=true;

return;

}

if ((abs(di-si)+abs(dj-sj))%2!=(T-t)%2) return ;

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

{

if(map[si+dx[i]][sj+dy[i]]!='X')

{

map[si+dx[i]][sj+dy[i]]='X';

dfs(si+dx[i],sj+dy[i],t+1);

map[si+dx[i]][sj+dy[i]]='.';

}

}

return;

}

int main()

{

while(~scanf("%d %d %d",&N,&M,&T) && N!=0)

{

Wall=0;

for(int i=1;i<=N;i++)

{

for(int j=1;j<=M;j++)

{

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

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

{

si=i;sj=j;

}

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

Wall++;

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

{

di=i;dj=j;

}

}

}

is=false;

if(N\*M-Wall<T)

is=false;

else

{

map[si][sj]='X';

dfs(si,sj,0);

}

if(is)

printf("YES\n");

else

printf("NO\n");

}

return 0;

}

**Problem D**

**Problem Description**

You're in space.  
You want to get home.  
There are asteroids.  
You don't want to hit them.

**Input**

Input to this problem will consist of a (non-empty) series of up to 100 data sets. Each data set will be formatted according to the following description, and there will be no blank lines separating data sets.  
  
A single data set has 5 components:  
  
Start line - A single line, "START N", where 1 <= N <= 10.  
  
Slice list - A series of N slices. Each slice is an N x N matrix representing a horizontal slice through the asteroid field. Each position in the matrix will be one of two values:  
  
'O' - (the letter "oh") Empty space  
  
'X' - (upper-case) Asteroid present  
  
Starting Position - A single line, "A B C", denoting the <A,B,C> coordinates of your craft's starting position. The coordinate values will be integers separated by individual spaces.  
  
Target Position - A single line, "D E F", denoting the <D,E,F> coordinates of your target's position. The coordinate values will be integers separated by individual spaces.  
  
End line - A single line, "END"  
  
The origin of the coordinate system is <0,0,0>. Therefore, each component of each coordinate vector will be an integer between 0 and N-1, inclusive.  
  
The first coordinate in a set indicates the column. Left column = 0.  
  
The second coordinate in a set indicates the row. Top row = 0.  
  
The third coordinate in a set indicates the slice. First slice = 0.  
  
Both the Starting Position and the Target Position will be in empty space.

**Output**

For each data set, there will be exactly one output set, and there will be no blank lines separating output sets.  
  
A single output set consists of a single line. If a route exists, the line will be in the format "X Y", where X is the same as N from the corresponding input data set and Y is the least number of moves necessary to get your ship from the starting position to the target position. If there is no route from the starting position to the target position, the line will be "NO ROUTE" instead.  
  
A move can only be in one of the six basic directions: up, down, left, right, forward, back. Phrased more precisely, a move will either increment or decrement a single component of your current position vector by 1.

**Sample Input**

START 1

O

0 0 0

0 0 0

END

START 3

XXX

XXX

XXX

OOO

OOO

OOO

XXX

XXX

XXX

0 0 1

2 2 1

END

START 5

OOOOO

OOOOO

OOOOO

OOOOO

OOOOO

OOOOO

OOOOO

OOOOO

OOOOO

OOOOO

XXXXX

XXXXX

XXXXX

XXXXX

XXXXX

OOOOO

OOOOO

OOOOO

OOOOO

OOOOO

OOOOO

OOOOO

OOOOO

OOOOO

OOOOO

0 0 0

4 4 4

END

**Sample Output**

1 0

3 4

NO ROUTE

**代码：**

#include <string.h>

#include <stdio.h>

#include <queue>

using namespace std;

char map[20][20][20];

int vis[20][20][20];

int n;

int sx,sy,sz;

int ex,ey,ez;

int tx[] = {1,-1,0,0,0,0};

int ty[] = {0,0,1,-1,0,0};

int tz[] = {0,0,0,0,1,-1};

struct node

{

int x,y,z,step;

};

int check(int x,int y,int z)

{

if(x<0 || y<0 || z<0 || x>=n || y>=n || z>=n || vis[x][y][z] )

return 0;

return 1;

}

int bfs(int x,int y,int z)

{

int i;

queue<node> Q;

node a,next;

a.x = x;

a.y = y;

a.z = z;

a.step = 0;

vis[x][y][z] = 1;

Q.push(a);

while(!Q.empty())

{

a = Q.front();

Q.pop();

if(a.x == ex && a.y == ey && a.z == ez)

return a.step;

for(i = 0;i<6;i++)

{

next = a;

next.x+=tx[i];

next.y+=ty[i];

next.z+=tz[i];

if(check(next.x,next.y,next.z))

{

next.step++;

vis[next.x][next.y][next.z] = 1;

Q.push(next);

}

}

}

return -1;

}

int main()

{

char s[10];

int i,j,k;

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

{

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

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

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

scanf("%d%d%d%d%d%d",&sx,&sy,&sz,&ex,&ey,&ez);

scanf("%s",s);

int ans = bfs(sx,sy,sz);

if(ans>=0)

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

else

printf("NO ROUTE\n");

}

return 0;

}

**Problem E**

**Problem Description**

The GeoSurvComp geologic survey company is responsible for detecting underground oil deposits. GeoSurvComp works with one large rectangular region of land at a time, and creates a grid 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：**

#include<cstdio>

#include<cstring>

#include<iostream>

#include<algorithm>

#include<cmath>

#include<queue>

#include<cstdlib>

#include<string>

using namespace std;

typedef long long ll;

const double pi=acos(-1.0);

char s[120][120];

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

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

void dfs(int i,int j){

if(i<0 || j<0)

return;

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

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

for(int k=0;k<8;k++)

dfs(i+a[k],j+b[k]);

}

}

int main(){

int m,n;

while(scanf("%d%d",&m,&n)!=EOF && (m||n)){

int i,j;

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

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

int ans=0;

for(i=0;i<m;i++){

for(j=0;j<n;j++){

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

ans++;

dfs(i,j);

}

}

}

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

}

return 0;

}

**Problem F**

**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：**

#include<cstdio>

#include<cstring>

#include<queue>

using namespace std;

int n,m;

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

char str[210][210];

struct point{

int x,y,time;

friend bool operator<(point a,point b)

{

return a.time>b.time;

}

};

int bfs(point start)

{

priority\_queue<point> q;

q.push(start);

point now,next;

int i;

while(!q.empty())

{

now=q.top();

q.pop();

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

{

next.x=now.x+d[i][0];

next.y=now.y+d[i][1];

if(str[next.x][next.y]=='r')

{

next.time=now.time+1;

while(!q.empty())

q.pop();

return next.time;

}

if(next.x>=0&&next.y>=0&&next.x<n&&next.y<m&&str[next.x][next.y]!='#')

{

if(str[next.x][next.y]=='x')

{

str[next.x][next.y]='#';

next.time=now.time+2;

q.push(next);

}

else

{

str[next.x][next.y]='#';

next.time=now.time+1;

q.push(next);

}

}

}

}

return -1;

}

int main()

{

int i,j,t;

point start;

while(scanf("%d%d",&n,&m)!=-1)

{

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

{

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

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

{

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

{

start.x=i;

start.y=j;

start.time=0;

}

}

}

t=bfs(start);

if(t==-1)

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

else

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

}

return 0;

}