**Problem A**

**Problem Description**

“今年暑假不AC？”  
“是的。”  
“那你干什么呢？”  
“看世界杯呀，笨蛋！”  
“@#$%^&\*%...”  
  
确实如此，世界杯来了，球迷的节日也来了，估计很多ACMer也会抛开电脑，奔向电视了。  
作为球迷，一定想看尽量多的完整的比赛，当然，作为新时代的好青年，你一定还会看一些其它的节目，比如新闻联播（永远不要忘记关心国家大事）、非常6+7、超级女生，以及王小丫的《开心辞典》等等，假设你已经知道了所有你喜欢看的电视节目的转播时间表，你会合理安排吗？（目标是能看尽量多的完整节目）

**Input**

输入数据包含多个测试实例，每个测试实例的第一行只有一个整数n(n<=100)，表示你喜欢看的节目的总数，然后是n行数据，每行包括两个数据Ti\_s,Ti\_e (1<=i<=n)，分别表示第i个节目的开始和结束时间，为了简化问题，每个时间都用一个正整数表示。n=0表示输入结束，不做处理。

**Output**

对于每个测试实例，输出能完整看到的电视节目的个数，每个测试实例的输出占一行。

**Sample Input**

12

1 3

3 4

0 7

3 8

15 19

15 20

10 15

8 18

6 12

5 10

4 14

2 9

0

**Sample Output**

5

**代码：**

#include<stdio.h>

void search(int b[100],int e[100],int a){

int num=1;

int esm;

int i,j,t;

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

scanf("%d %d",&b[i],&e[i]);

}

for(i=0;i<a-1;i++){

for(j=0;j<a-1-i;j++){

if(e[j]>e[j+1]){

t=e[j];

e[j]=e[j+1];

e[j+1]=t;

t=b[j];

b[j]=b[j+1];

b[j+1]=t;

}

}

}

esm=e[0];

for(i=1;i<a;i++){

if(b[i]>=esm){

num++;

esm=e[i];

}

}

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

}

int main(){

int b[100],e[100];

int a;

scanf("%d",&a);

while(a!=0){

search(b,e,a);

scanf("%d",&a);

}

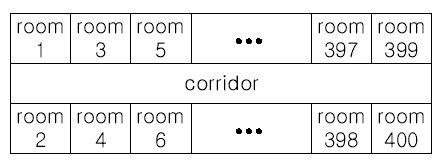
return 0;

}

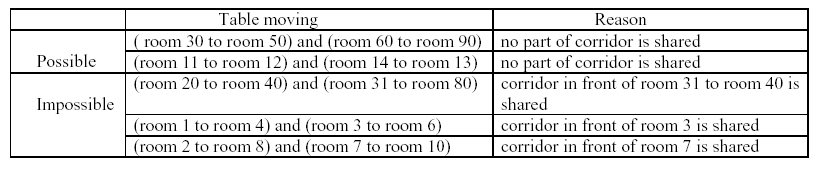
**Problem B**

**Problem Description**

The famous ACM (Advanced Computer Maker) Company has rented a floor of a building whose shape is in the following figure.



The floor has 200 rooms each on the north side and south side along the corridor. Recently the Company made a plan to reform its system. The reform includes moving a lot of tables between rooms. Because the corridor is narrow and all the tables are big, only one table can pass through the corridor. Some plan is needed to make the moving efficient. The manager figured out the following plan: Moving a table from a room to another room can be done within 10 minutes. When moving a table from room i to room j, the part of the corridor between the front of room i and the front of room j is used. So, during each 10 minutes, several moving between two rooms not sharing the same part of the corridor will be done simultaneously. To make it clear the manager illustrated the possible cases and impossible cases of simultaneous moving.



For each room, at most one table will be either moved in or moved out. Now, the manager seeks out a method to minimize the time to move all the tables. Your job is to write a program to solve the manager’s problem.

**Input**

The input consists of T test cases. The number of test cases ) (T is given in the first line of the input. Each test case begins with a line containing an integer N , 1<=N<=200 , that represents the number of tables to move. Each of the following N lines contains two positive integers s and t, representing that a table is to move from room number s to room number t (each room number appears at most once in the N lines). From the N+3-rd line, the remaining test cases are listed in the same manner as above.

**Output**

The output should contain the minimum time in minutes to complete the moving, one per line.

**Sample Input**

3

4

10 20

30 40

50 60

70 80

2

1 3

2 200

3

10 100

20 80

30 50

**Sample Output**

10

20

30

**代码：**

#include<stdio.h>

void search(int b){

int i,t,f,j;

int temp;

int s[200]={0};

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

scanf("%d %d",&t,&f);

if(t>f){

temp=t;

t=f;

f=temp;

}

for(j=(t-1)/2;j<=(f-1)/2;j++){

s[j]++;

}

}

int max=s[0];

for(i=1;i<200;i++){

if(s[i]>max){

max=s[i];

}

}

printf("%d\n",max\*10);

}

int main(){

int a,i,b;

scanf("%d",&a);

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

scanf("%d",&b);

search(b);

}

return 0;

}

**Problem C**

**Problem Description**

There is a pile of n wooden sticks. The length and weight of each stick are known in advance. The sticks are to be processed by a woodworking machine in one by one fashion. It needs some time, called setup time, for the machine to prepare processing a stick. The setup times are associated with cleaning operations and changing tools and shapes in the machine. The setup times of the woodworking machine are given as follows:   
  
(a) The setup time for the first wooden stick is 1 minute.   
(b) Right after processing a stick of length l and weight w , the machine will need no setup time for a stick of length l' and weight w' if l<=l' and w<=w'. Otherwise, it will need 1 minute for setup.   
  
You are to find the minimum setup time to process a given pile of n wooden sticks. For example, if you have five sticks whose pairs of length and weight are (4,9), (5,2), (2,1), (3,5), and (1,4), then the minimum setup time should be 2 minutes since there is a sequence of pairs (1,4), (3,5), (4,9), (2,1), (5,2).

**Input**

The input consists of T test cases. The number of test cases (T) is given in the first line of the input file. Each test case consists of two lines: The first line has an integer n , 1<=n<=5000, that represents the number of wooden sticks in the test case, and the second line contains n 2 positive integers l1, w1, l2, w2, ..., ln, wn, each of magnitude at most 10000 , where li and wi are the length and weight of the i th wooden stick, respectively. The 2n integers are delimited by one or more spaces.

**Output**

The output should contain the minimum setup time in minutes, one per line.

**Sample Input**

3

5

4 9 5 2 2 1 3 5 1 4

3

2 2 1 1 2 2

3

1 3 2 2 3 1

**Sample Output**

2

1

3

**代码：**

#include<iostream>

using namespace std;

class a

{

public:

int x,y,flog;

};

int main()

{

int N,i,j,n,count;

a f[5010],t;

cin>>N;

while(N--)

{

cin>>n;

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

{

cin>>f[i].x>>f[i].y;

f[i].flog=1;

}

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

for(j=i+1; j<n; j++)

if(f[i].x>f[j].x)

{

t=f[i];

f[i]=f[j];

f[j]=t;

}

else if(f[i].x==f[j].x&&f[i].y>f[j].y)

{

t=f[i];

f[i]=f[j];

f[j]=t;

}

count=0;

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

if(f[i].flog)

{

t=f[i];

count++;

for(j=i+1; j<n; j++)

if(f[j].flog)

if(t.y<=f[j].y)

{

t=f[j];

f[j].flog=0;

}

}

cout<<count<<endl;

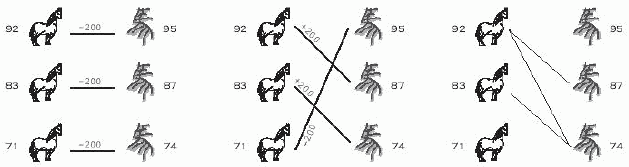
}

return 0;

}

**Problem D**

**Problem Description**

Here is a famous story in Chinese history.  
  
"That was about 2300 years ago. General Tian Ji was a high official in the country Qi. He likes to play horse racing with the king and others."  
  
"Both of Tian and the king have three horses in different classes, namely, regular, plus, and super. The rule is to have three rounds in a match; each of the horses must be used in one round. The winner of a single round takes two hundred silver dollars from the loser."  
  
"Being the most powerful man in the country, the king has so nice horses that in each class his horse is better than Tian's. As a result, each time the king takes six hundred silver dollars from Tian."  
  
"Tian Ji was not happy about that, until he met Sun Bin, one of the most famous generals in Chinese history. Using a little trick due to Sun, Tian Ji brought home two hundred silver dollars and such a grace in the next match."  
  
"It was a rather simple trick. Using his regular class horse race against the super class from the king, they will certainly lose that round. But then his plus beat the king's regular, and his super beat the king's plus. What a simple trick. And how do you think of Tian Ji, the high ranked official in China?"  
  
  
  
Were Tian Ji lives in nowadays, he will certainly laugh at himself. Even more, were he sitting in the ACM contest right now, he may discover that the horse racing problem can be simply viewed as finding the maximum matching in a bipartite graph. Draw Tian's horses on one side, and the king's horses on the other. Whenever one of Tian's horses can beat one from the king, we draw an edge between them, meaning we wish to establish this pair. Then, the problem of winning as many rounds as possible is just to find the maximum matching in this graph. If there are ties, the problem becomes more complicated, he needs to assign weights 0, 1, or -1 to all the possible edges, and find a maximum weighted perfect matching...  
  
However, the horse racing problem is a very special case of bipartite matching. The graph is decided by the speed of the horses --- a vertex of higher speed always beat a vertex of lower speed. In this case, the weighted bipartite matching algorithm is a too advanced tool to deal with the problem.  
  
In this problem, you are asked to write a program to solve this special case of matching problem.

**Input**

The input consists of up to 50 test cases. Each case starts with a positive integer n (n <= 1000) on the first line, which is the number of horses on each side. The next n integers on the second line are the speeds of Tian’s horses. Then the next n integers on the third line are the speeds of the king’s horses. The input ends with a line that has a single 0 after the last test case.

**Output**

For each input case, output a line containing a single number, which is the maximum money Tian Ji will get, in silver dollars.

**Sample Input**

3

92 83 71

95 87 74

2

20 20

20 20

2

20 19

22 18

0

**Sample Output**

200

0

0

**代码：**

#include<stdio.h>

#include<iostream>

#include<algorithm>

using namespace std;

bool ran(int a,int b){

return a<b;

}

int main(){

int a,i;

int king[1000];

int tj[1000];

while(true){

scanf("%d",&a);

if(a==0)break;

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

cin>>tj[i];

}

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

cin>>king[i];

}

sort(tj,tj+a,ran);

sort(king,king+a,ran);

int king\_min=0;

int king\_max=a-1;

int tj\_min=0;

int tj\_max=a-1;

int money=0;

while(tj\_min<=tj\_max){

if(tj[tj\_min]>king[king\_min]){

money+=200;

tj\_min++;

king\_min++;

}else if(tj[tj\_max]>king[king\_max]){

money+=200;

tj\_max--;

king\_max--;

}else{

if(tj[tj\_min]<king[king\_max]){

money-=200;

tj\_min++;

king\_max--;

}else{

break;

}

}

}

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

}

}

**Problem E**

**Problem Description**

Ignatius has just come back school from the 30th ACM/ICPC. Now he has a lot of homework to do. Every teacher gives him a deadline of handing in the homework. If Ignatius hands in the homework after the deadline, the teacher will reduce his score of the final test. And now we assume that doing everyone homework always takes one day. So Ignatius wants you to help him to arrange the order of doing homework to minimize the reduced score.

**Input**

The input contains several test cases. The first line of the input is a single integer T that is the number of test cases. T test cases follow.  
Each test case start with a positive integer N(1<=N<=1000) which indicate the number of homework.. Then 2 lines follow. The first line contains N integers that indicate the deadlines of the subjects, and the next line contains N integers that indicate the reduced scores.

**Output**

For each test case, you should output the smallest total reduced score, one line per test case.

**Sample Input**

3

3

3 3 3

10 5 1

3

1 3 1

6 2 3

7

1 4 6 4 2 4 3

3 2 1 7 6 5 4

**Sample Output**

0

3

5

**代码：**

#include<stdio.h>

#include<string.h>

#define max 1005

struct he

{

int time;

int score;

};

bool f[max];

int main()

{

int t,n,i,min,j;

he student[max],temp;

scanf("%d",&t);

while(t--)

{

scanf("%d",&n);

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

scanf("%d",&student[i].time);

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

scanf("%d",&student[i].score);

min=0;

memset(f,0,sizeof(f));

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

for(j=i+1;j<n;j++)

{

if(student[j].score>student[i].score)

{ temp=student[j];student[j]=student[i];student[i]=temp; }

else if(student[j].score==student[i].score && student[j].time<student[i].time)

{ temp=student[j];student[j]=student[i];student[i]=temp; }

}

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

{

for(j=student[i].time;j>0;j--)

{

if(f[j]==false)

{

f[j]=true;

break;

}

}

if(j==0)

min+=student[i].score;

}

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

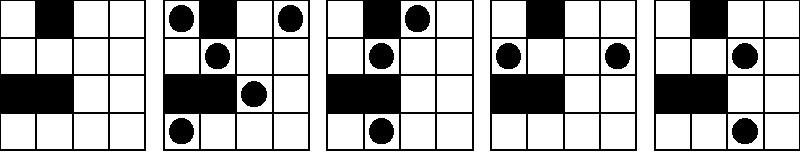
}

return 0;

}

**Problem F**

**Problem Description**

Suppose that we have a square city with straight streets. A map of a city is a square board with n rows and n columns, each representing a street or a piece of wall.   
  
A blockhouse is a small castle that has four openings through which to shoot. The four openings are facing North, East, South, and West, respectively. There will be one machine gun shooting through each opening.   
  
Here we assume that a bullet is so powerful that it can run across any distance and destroy a blockhouse on its way. On the other hand, a wall is so strongly built that can stop the bullets.   
  
The goal is to place as many blockhouses in a city as possible so that no two can destroy each other. A configuration of blockhouses is legal provided that no two blockhouses are on the same horizontal row or vertical column in a map unless there is at least one wall separating them. In this problem we will consider small square cities (at most 4x4) that contain walls through which bullets cannot run through.   
  
The following image shows five pictures of the same board. The first picture is the empty board, the second and third pictures show legal configurations, and the fourth and fifth pictures show illegal configurations. For this board, the maximum number of blockhouses in a legal configuration is 5; the second picture shows one way to do it, but there are several other ways.   
  
  
  
Your task is to write a program that, given a description of a map, calculates the maximum number of blockhouses that can be placed in the city in a legal configuration.

**Input**

The input file contains one or more map descriptions, followed by a line containing the number 0 that signals the end of the file. Each map description begins with a line containing a positive integer n that is the size of the city; n will be at most 4. The next n lines each describe one row of the map, with a '.' indicating an open space and an uppercase 'X' indicating a wall. There are no spaces in the input file.

**Output**

For each test case, output one line containing the maximum number of blockhouses that can be placed in the city in a legal configuration.

**Sample Input**

4

.X..

....

XX..

....

2

XX

.X

3

.X.

X.X

.X.

3

...

.XX

.XX

4

....

....

....

....

0

**Sample Output**

5

1

5

2

4

**代码：**

#include<stdio.h>

int n;

char map[4][4];

int bestn;

int canput(int row,int col)

{

int i;

for(i=row-1;i>=0;i--)

{

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

{

break;

}

if(map[i][col]=='o')

{

return 0;

}

}

for(i=col-1;i>=0;i--)

{

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

{

break;

}

if(map[row][i]=='o')

{

return 0;

}

}

return 1;

}

void backtrack(int k,int current)

{

int x,y;

if(k>=n\*n)

{

if(current>bestn)

{

bestn=current;

}

return;

}

else

{

x=k/n;

y=k%n;

if(map[x][y]=='.'&&canput(x,y))

{

map[x][y]='o';

backtrack(k+1,current+1);

map[x][y]='.';

}

backtrack(k+1,current);

}

}

void initial()

{

int i,j;

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

{

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

{

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

}

}

}

int main()

{

scanf("%d",&n);

while(n)

{

int i,j;

bestn=0;

initial();

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

{

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

{

char ch;

ch=getchar();

if(ch=='\n')

{

j--;

continue;

}

else

{

map[i][j]=ch;

}

}

}

backtrack(0,0);

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

scanf("%d",&n);

}

return 0;

}

**Problem G**

**Problem Description**

An entropy encoder is a data encoding method that achieves lossless data compression by encoding a message with “wasted” or “extra” information removed. In other words, entropy encoding removes information that was not necessary in the first place to accurately encode the message. A high degree of entropy implies a message with a great deal of wasted information; english text encoded in ASCII is an example of a message type that has very high entropy. Already compressed messages, such as JPEG graphics or ZIP archives, have very little entropy and do not benefit from further attempts at entropy encoding.  
  
English text encoded in ASCII has a high degree of entropy because all characters are encoded using the same number of bits, eight. It is a known fact that the letters E, L, N, R, S and T occur at a considerably higher frequency than do most other letters in english text. If a way could be found to encode just these letters with four bits, then the new encoding would be smaller, would contain all the original information, and would have less entropy. ASCII uses a fixed number of bits for a reason, however: it’s easy, since one is always dealing with a fixed number of bits to represent each possible glyph or character. How would an encoding scheme that used four bits for the above letters be able to distinguish between the four-bit codes and eight-bit codes? This seemingly difficult problem is solved using what is known as a “prefix-free variable-length” encoding.  
  
In such an encoding, any number of bits can be used to represent any glyph, and glyphs not present in the message are simply not encoded. However, in order to be able to recover the information, no bit pattern that encodes a glyph is allowed to be the prefix of any other encoding bit pattern. This allows the encoded bitstream to be read bit by bit, and whenever a set of bits is encountered that represents a glyph, that glyph can be decoded. If the prefix-free constraint was not enforced, then such a decoding would be impossible.  
  
Consider the text “AAAAABCD”. Using ASCII, encoding this would require 64 bits. If, instead, we encode “A” with the bit pattern “00”, “B” with “01”, “C” with “10”, and “D” with “11” then we can encode this text in only 16 bits; the resulting bit pattern would be “0000000000011011”. This is still a fixed-length encoding, however; we’re using two bits per glyph instead of eight. Since the glyph “A” occurs with greater frequency, could we do better by encoding it with fewer bits? In fact we can, but in order to maintain a prefix-free encoding, some of the other bit patterns will become longer than two bits. An optimal encoding is to encode “A” with “0”, “B” with “10”, “C” with “110”, and “D” with “111”. (This is clearly not the only optimal encoding, as it is obvious that the encodings for B, C and D could be interchanged freely for any given encoding without increasing the size of the final encoded message.) Using this encoding, the message encodes in only 13 bits to “0000010110111”, a compression ratio of 4.9 to 1 (that is, each bit in the final encoded message represents as much information as did 4.9 bits in the original encoding). Read through this bit pattern from left to right and you’ll see that the prefix-free encoding makes it simple to decode this into the original text even though the codes have varying bit lengths.  
  
As a second example, consider the text “THE CAT IN THE HAT”. In this text, the letter “T” and the space character both occur with the highest frequency, so they will clearly have the shortest encoding bit patterns in an optimal encoding. The letters “C”, “I’ and “N” only occur once, however, so they will have the longest codes.  
  
There are many possible sets of prefix-free variable-length bit patterns that would yield the optimal encoding, that is, that would allow the text to be encoded in the fewest number of bits. One such optimal encoding is to encode spaces with “00”, “A” with “100”, “C” with “1110”, “E” with “1111”, “H” with “110”, “I” with “1010”, “N” with “1011” and “T” with “01”. The optimal encoding therefore requires only 51 bits compared to the 144 that would be necessary to encode the message with 8-bit ASCII encoding, a compression ratio of 2.8 to 1.

**Input**

The input file will contain a list of text strings, one per line. The text strings will consist only of uppercase alphanumeric characters and underscores (which are used in place of spaces). The end of the input will be signalled by a line containing only the word “END” as the text string. This line should not be processed.

**Output**

For each text string in the input, output the length in bits of the 8-bit ASCII encoding, the length in bits of an optimal prefix-free variable-length encoding, and the compression ratio accurate to one decimal point.

**Sample Input**

AAAAABCD

THE\_CAT\_IN\_THE\_HAT

END

**Sample Output**

64 13 4.9

144 51 2.8

**代码：**

#include<stdio.h>

#include<iostream>

#include<string>

#include<cstring>

using namespace std;

struct{

int weight;

int parent;

int left;

int right;

}node[60];

int main(){

char ch[100000];

int hash[35];

int i,j,k;

int len;

while(cin>>ch,strcmp(ch,"END")!=0){

memset(hash,0,sizeof(hash));

len=strlen(ch);

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

hash[ch[i]-64]++;

}

for(i=1,j=1;i<35;i++){

if(hash[i]!=0){

node[j].weight=hash[i];

node[j].parent=node[j].left=node[j].right=0;

j++;

}

}

j--;

for(i=j+1;i<2\*j;i++){

node[i].parent=node[i].left=node[i].right=0;

}

for(i=j+1;i<j\*2;i++){

int m1,m2;

m1=m2=999999;

int x1,x2;

for(k=1;k<i;k++){

if(node[k].weight<m1&&node[k].parent==0){

m2=m1;

x2=x1;

m1=node[k].weight;

x1=k;

}else if(node[k].weight<m2&&node[k].parent==0){

m2=node[k].weight;

x2=k;

}

}

node[i].weight=m1+m2;

node[i].right=x1;

node[i].left=x2;

node[x1].parent=i;

node[x2].parent=i;

}

int sum=0;

int cnt;

for(i=1;i<j+1;i++){

k=i;

cnt=0;

while(node[k].parent!=0){

k=node[k].parent;

cnt++;

}

sum+=cnt\*node[i].weight;

}

if(j==1)sum=len;

float ratio;

ratio=float(len\*8)/sum;

printf("%d %d %.1f\n",len\*8,sum,ratio);

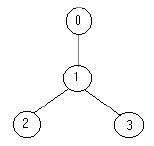
}

return 0;

}

**Problem H**

**Problem Description**

Bob enjoys playing computer games, especially strategic games, but sometimes he cannot find the solution fast enough and then he is very sad. Now he has the following problem. He must defend a medieval city, the roads of which form a tree. He has to put the minimum number of soldiers on the nodes so that they can observe all the edges. Can you help him?  
  
Your program should find the minimum number of soldiers that Bob has to put for a given tree.  
  
The input file contains several data sets in text format. Each data set represents a tree with the following description:  
  
the number of nodes  
the description of each node in the following format  
node\_identifier:(number\_of\_roads) node\_identifier1 node\_identifier2 ... node\_identifier  
or  
node\_identifier:(0)  
  
The node identifiers are integer numbers between 0 and n-1, for n nodes (0 < n <= 1500). Every edge appears only once in the input data.  
  
For example for the tree:   
  
  
  
the solution is one soldier ( at the node 1).  
  
The output should be printed on the standard output. For each given input data set, print one integer number in a single line that gives the result (the minimum number of soldiers). An example is given in the following table:

**Sample Input**

4

0:(1) 1

1:(2) 2 3

2:(0)

3:(0)

5

3:(3) 1 4 2

1:(1) 0

2:(0)

0:(0)

4:(0)

**Sample Output**

1

2

**代码：**

#include <stdio.h>

#include <string.h>

#include <algorithm>

using namespace std;

int dp[1505][2];

int vis[1505],head[1505];

int len,root;

struct node

{

int now,next;

} tree[3005];

void add(int x,int y)

{

tree[len].now = y;

tree[len].next = head[x];

head[x] = len++;

tree[len].now = x;

tree[len].next = head[y];

head[y] = len++;

}

void dfs(int root)

{

int i,k;

vis[root] = 1;

dp[root][1] = 1;

dp[root][0] = 0;

for(i = head[root]; i!=-1; i = tree[i].next)

{

k = tree[i].now;

if(!vis[k])

{

dfs(k);

dp[root][0] += dp[k][1];

dp[root][1] += min(dp[k][1],dp[k][0]);

}

}

}

int main()

{

int t,x,y,n,i,j;

while(~scanf("%d",&t))

{

len = 0;

root = -1;

memset(dp,0,sizeof(dp));

memset(vis,0,sizeof(vis));

memset(head,-1,sizeof(head));

for(j = 1; j<=t; j++)

{

scanf("%d:(%d)",&x,&n);

if(root==-1)

root = x;

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

{

scanf("%d",&y);

add(x,y);

}

}

dfs(root);

printf("%d\n",min(dp[root][0],dp[root][1]));

for(i=0;i<9;i++)

printf("%d\t",head[i]);

for(i=0;i<5;i++){

printf("\n");

for(j=0;j<2;j++){

printf("%d\t",dp[i][j]);

}

}

}

return 0;

}

**Problem I**

**Problem Description**



In the year 8888, the Earth is ruled by the PPF Empire . As the population growing , PPF needs to find more land for the newborns . Finally , PPF decides to attack Kscinow who ruling the Mars . Here the problem comes! How can the soldiers reach the Mars ? PPF convokes his soldiers and asks for their suggestions . “Rush … ” one soldier answers. “Shut up ! Do I have to remind you that there isn’t any road to the Mars from here!” PPF replies. “Fly !” another answers. PPF smiles :“Clever guy ! Although we haven’t got wings , I can buy some magic broomsticks from HARRY POTTER to help you .” Now , it’s time to learn to fly on a broomstick ! we assume that one soldier has one level number indicating his degree. The soldier who has a higher level could teach the lower , that is to say the former’s level > the latter’s . But the lower can’t teach the higher. One soldier can have only one teacher at most , certainly , having no teacher is also legal. Similarly one soldier can have only one student at most while having no student is also possible. Teacher can teach his student on the same broomstick .Certainly , all the soldier must have practiced on the broomstick before they fly to the Mars! Magic broomstick is expensive !So , can you help PPF to calculate the minimum number of the broomstick needed .  
For example :   
There are 5 soldiers (A B C D E)with level numbers : 2 4 5 6 4;  
One method :  
C could teach B; B could teach A; So , A B C are eligible to study on the same broomstick.  
D could teach E;So D E are eligible to study on the same broomstick;  
Using this method , we need 2 broomsticks.  
Another method:  
D could teach A; So A D are eligible to study on the same broomstick.  
C could teach B; So B C are eligible to study on the same broomstick.  
E with no teacher or student are eligible to study on one broomstick.  
Using the method ,we need 3 broomsticks.  
……  
  
After checking up all possible method, we found that 2 is the minimum number of broomsticks needed.

**Input**

Input file contains multiple test cases.   
In a test case,the first line contains a single positive number N indicating the number of soldiers.(0<=N<=3000)  
Next N lines :There is only one nonnegative integer on each line , indicating the level number for each soldier.( less than 30 digits);

**Output**

For each case, output the minimum number of broomsticks on a single line.

**Sample Input**

4

10

20

30

04

5

2

3

4

3

4

**Sample Output**

1

2

**代码：**

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

int main()

{

int n,i,max,t;

int a[3005];

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

{

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

scanf("%d",&a[i]);

sort(a,a+n);

max=1;

t=1;

for(i=1;i<n;++i)

{

if(a[i]!=a[i-1])

t=1;

else

++t;

if(t>max)

max=t;

}

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

}

return 0;

}

**Problem J**

**Problem Description**

FatMouse prepared M pounds of cat food, ready to trade with the cats guarding the warehouse containing his favorite food, JavaBean.  
The warehouse has N rooms. The i-th room contains J[i] pounds of JavaBeans and requires F[i] pounds of cat food. FatMouse does not have to trade for all the JavaBeans in the room, instead, he may get J[i]\* a% pounds of JavaBeans if he pays F[i]\* a% pounds of cat food. Here a is a real number. Now he is assigning this homework to you: tell him the maximum amount of JavaBeans he can obtain.

**Input**

The input consists of multiple test cases. Each test case begins with a line containing two non-negative integers M and N. Then N lines follow, each contains two non-negative integers J[i] and F[i] respectively. The last test case is followed by two -1's. All integers are not greater than 1000.

**Output**

For each test case, print in a single line a real number accurate up to 3 decimal places, which is the maximum amount of JavaBeans that FatMouse can obtain.

**Sample Input**

5 3

7 2

4 3

5 2

20 3

25 18

24 15

15 10

-1 -1

**Sample Output**

13.333

31.500

**代码：**

#include<stdio.h>

#include<stdlib.h>

#include<algorithm>

using namespace std;

#define Max 1000

struct room{

double j,f;

double rate;

}Room[Max];

bool cmp(room a,room b)

{

return a.rate>b.rate;

}

int main()

{

int N;

double M,sumJ;

int i;

while(scanf("%lf %d",&M,&N))

{

if(M==-1&&N==-1)

break;

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

{

scanf("%lf%lf",&Room[i].j,&Room[i].f);

Room[i].rate=(double)Room[i].j/Room[i].f;

}

sort(Room,Room+N,cmp);

sumJ=0;

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

{

if(M>=Room[i].f)

{

sumJ+=Room[i].j;

M-=Room[i].f;

}

else

{

sumJ+=(Room[i].rate)\*M;

break;

}

}

printf("%.3lf\n",sumJ);

}

return 0;

}