第一章：算法与数据结构

1.1 程序=算法+数据结构

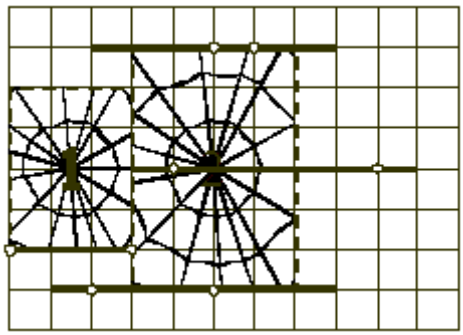
1.2 基本算法

1.2.1 枚举

Pku1116-----Library

Description

Castaway Robinson Crusoe is living alone on a remote island. One day a ship carrying a royal library has wrecked nearby. Usually Robinson brings any useful stuff from the shipwreck to his island, and this time he has brought a big chest with books.
Robinson has decided to build a bookcase for these books to create his own library. He cut a rectangular niche in the rock for that purpose, hammered in wooden pegs, and placed wooden planks on every pair of pegs that have the same height, so that all planks are situated horizontally and suit to act as shelves.
Unfortunately, Robinson has discovered that one especially old and big tome does not fit in his bookcase. He measured the height and width of this tome and has decided to redesign his bookcase in such a way, as to completely fit the tome on one of the shelves, taking into account locations of other shelves and the dimensions of the niche. With each shelf in the bookcase, one of the following operations should be made:



1. Leave the shelf on its original place.
2. Move the shelf to the left or to the right.
3. Shorten the shelf by cutting off a part of the plank and optionally move it to the left or to the right.
4. Move one of the pegs to a different place at the same height and move the shelf to the left or to the right.
5. Shorten the shelf by cutting off a part of the plank, move one of the pegs to a different

place at the same height, and optionally move the shortened shelf to the left or to the right.

6. Remove the shelf from the bookcase along with both supporting pegs.

We say that the shelf is properly supported by its pegs, if exactly two distinct pegs support the shelf and the center of the shelf is between its pegs or coincides with one of the pegs.

The original design of Robinson's library has all the shelves properly supported by their pegs and lengths of all shelves are integer number of inches. The Robinson may only cut an integer number of inches from the planks, because he has no tools for more precise measurements. All remaining shelves after the redesign must be properly supported by their pegs.

You are to find the way to redesign Robinson's library to fit the special old tome without changing original design too much. You have to minimize the number of pegs that are to be removed from their original places during the redesign (operations 4 and 5 remove one peg, and operation 6 removes two pegs). If there are different ways to solve the problem, then you are to find the one that minimizes the total length of planks that are to be cut off (operations 3 and 5 involve cutting something from the planks, and operation 6 counts as if cutting off the whole plank). Width of planks and diameter of pegs shall be considered zero.

The tome may not be rotated. The tome should completely (to all its width) stand on one of the shelves and may only touch other shelves, their pegs or niche's edge.

Input

The first line of the input file contains four integer numbers XN, YN, XT, and YT, separated by spaces. They are, correspondingly, width and height of the niche, and width and height of the old tome in inches (1 <= XN, YN, XT, YT <= 1000).

The second line of the input file contains a single integer number N (1 <= N <= 100) that represents the number of the shelves. Then N lines follow. Each line represents a single shelf along with its two supporting pegs, and contains five integer numbers yi, xi, li, x1i, x2i, separated by spaces, where:

?yi (0 < yi < YN) - the height of the ith shelf above the bottom of the niche in inches.

?xi (0 <= xi < XN) - the distance between the left end of the ith shelf and the left edge of the niche in inches.

?li (0 < li <= XN - xi) - the length of the ith shelf in inches.

?x1i (0 <= x1i <= li/2) - the distance between the left end of the ith shelf and its leftmost supporting peg in inches.

?x2i (li/2 <= x2i <= li; x1i < x2i) - the distance between the left end of the ith shelf and its rightmost supporting peg in inches.

All shelves are situated on different heights and are properly supported by their pegs. The problem is guaranteed to have a solution for the input data.

Output

The output file shall contain two integer numbers separated by a space. The first one is

the minimal number of pegs that are to be removed by Robinson from their original locations to place the tome. The second one is the minimal total length of planks in inches that are to be cut off during the redesign that removes the least number of pegs.

Sample Input

```
11 8 4 6
4
1 1 7 1 4
4 3 7 1 6
7 2 6 3 4
2 0 3 0 3
```

Sample Output

```
1 3
```

```
#include <stdio.h>
#include <iostream>
#include <algorithm>
using namespace std;
struct shelf{int x, y, len, x1, x2;}a[101];
int N, XN, YN, XT, YT;
int optPegs = 20000, optCost = 1000000;
bool cmp (shelf a, shelf b){return a.y < b.y;}
void check (int k, int x, int &minPegs, int &minCost)
{
    int i;
    for (i = k+1; i < N; i++){
        if (a[i].y >= a[k].y + YT) break;
        if (a[i].x + a[i].len <= x || a[i].x >= x + XT);
        else if (a[i].x2 <= x){
            int max;
            if (x - a[i].x1 <= a[i].x1) max = 2 * (x - a[i].x1);
            else max = x;
            if (max < a[i].len) minCost += a[i].len - max;
        }
        else if (a[i].x1 >= x + XT){
            int max;
            if (a[i].x2 - (x + XT) <= XN - a[i].x2) max = 2 * (a[i].x2 - (x + XT));
            else max = XN - (x + XT);
            if (max < a[i].len) minCost += a[i].len - max;
        }
        else if (a[i].x1 <= x && a[i].x2 > x && a[i].x2 < x + XT){
```

```
            if (x == 0) minPegs += 2, minCost += a[i].len;
            else {
                minPegs++;
                if (a[i].len > x) minCost += a[i].len - x;
            }
        }
        else if (a[i].x1 > x && a[i].x1 < x + XT && a[i].x2 >= x + XT){
            if (x + XT == XN)    minPegs += 2, minCost += a[i].len;
            else {
                minPegs++;
                if (a[i].len > XN - (x + XT)) minCost += a[i].len -(XN - (x + XT));
            }
        }
        else if (a[i].x1 <= x && a[i].x2 >= x + XT){
            if (x == 0 && XT == XN) minPegs += 2;
            else minPegs++;
            int max = x > XN - (x + XT) ? x : XN - (x + XT);
            if (a[i].len > max) minCost += a[i].len - max;
        }
        else if (a[i].x1 > x && a[i].x2 < x + XT){
            minPegs += 2;
            minCost += a[i].len;
        }
    }
}
void solve (int k)
{
    int book_l;
    for (book_l = 0; book_l + XT <= XN; book_l++){
        int minPegs = 0, minCost = 0;
        if (book_l + a[k].len < a[k].x1) continue;
        if (a[k].x2 + a[k].len < book_l + XT) break;
        if (book_l + a[k].len < a[k].x1 || a[k].x2 + a[k].len < book_l + XT) continue;
        if (2 * (a[k].x1 - book_l) > a[k].len || 2 * (book_l + XT - a[k].x2) > a[k].len)
minPegs++;
        else if (a[k].x2 - book_l > a[k].len || (book_l + XT) - a[k].x1 > a[k].len) minPegs++;
        check (k, book_l, minPegs, minCost);
        if (minPegs < optPegs || (minPegs == optPegs && minCost < optCost) )
            optPegs = minPegs,    optCost = minCost;
    }
}
int main()
{
    scanf ("%d%d%d%d",&XN, &YN, &XT, &YT);
```

```
    scanf ("%d",&N);
    int i;
    for (i = 0; i < N; i++){
        scanf("%d%d%d%d%d", &a[i].y, &a[i].x, &a[i].len, &a[i].x1, &a[i].x2);
        a[i].x1 += a[i].x;
        a[i].x2 += a[i].x;
    }
    sort (a, a+N, cmp);
    for (i = 0; i < N; i++){
        if (a[i].y + YT > YN) break;
        if (a[i].len >= XT) solve    (i);
    }
    printf ("%d %d", optPegs, optCost);
}
```

1.2.2 贪心

Pku1042--Gone Fishing

Description

John is going on a fishing trip. He has h hours available (1 <= h <= 16), and there are n lakes in the area (2 <= n <= 25) all reachable along a single, one-way road. John starts at lake 1, but he can finish at any lake he wants. He can only travel from one lake to the next one, but he does not have to stop at any lake unless he wishes to. For each i = 1,...,n - 1, the number of 5-minute intervals it takes to travel from lake i to lake i + 1 is denoted ti (0 < ti <=192). For example, t3 = 4 means that it takes 20 minutes to travel from lake 3 to lake 4. To help plan his fishing trip, John has gathered some information about the lakes. For each lake i, the number of fish expected to be caught in the initial 5 minutes, denoted fi(fi >= 0), is known. Each 5 minutes of fishing decreases the number of fish expected to be caught in the next 5-minute interval by a constant rate of di (di >= 0). If the number of fish expected to be caught in an interval is less than or equal to di , there will be no more fish left in the lake in the next interval. To simplify the planning, John assumes that no one else will be fishing at the lakes to affect the number of fish he expects to catch.
Write a program to help John plan his fishing trip to maximize the number of fish expected to be caught. The number of minutes spent at each lake must be a multiple of 5.

Input

You will be given a number of cases in the input. Each case starts with a line containing n. This is followed by a line containing h. Next, there is a line of n integers specifying fi (1 <= i <=n), then a line of n integers di (1 <=i <=n), and finally, a line of n - 1 integers ti (1 <=i <=n - 1). Input is

terminated by a case in which n = 0.

Output

For each test case, print the number of minutes spent at each lake, separated by commas, for the plan achieving the maximum number of fish expected to be caught (you should print the entire plan on one line even if it exceeds 80 characters). This is followed by a line containing the number of fish expected.
If multiple plans exist, choose the one that spends as long as possible at lake 1, even if no fish are expected to be caught in some intervals. If there is still a tie, choose the one that spends as long as possible at lake 2, and so on. Insert a blank line between cases.

Sample Input

```
2
1
10 1
2 5
2
4
4
10 15 20 17
0 3 4 3
1 2 3
4
4
10 15 50 30
0 3 4 3
1 2 3
0
```

Sample Output

```
45, 5
Number of fish expected: 31

240, 0, 0, 0
Number of fish expected: 480

115, 10, 50, 35
Number of fish expected: 724
```

#include <iostream>

```
using namespace std;
int main()
{
 int n,h;
 while(cin>>n)
 {
  if(n==0) break;
  int d[26]={0},t[26]={0},f[26]={0},fs[26];
  cin>>h;
  h*=12;
  int i,j;
  for(i=0;i<n;i++)
  cin>>f[i];
  for(i=0;i<n;i++)
  cin>>d[i];
  for(i=1;i<n;i++)
  cin>>t[i];
  int time,k;
  int ans[26][26]={0};
  memset(ans,0,sizeof(ans));
  //贪心
  for(i=1;i<=n;i++)//在前 i 个湖中每次寻找鱼最多的湖下钓
  {
     memcpy(fs,f,sizeof(f));
     //for(j=0;j<i;j++) fs[j]=f[j];
     int flag=0;
     h=h-t[i-1];
     time=h;
     //每次寻找鱼最多的湖下钓直到时间用完或所有湖中鱼钓完
      while(time>0 && flag!=i)
      {
       k=0;
       for(j=1;j<i;j++)
          if(fs[j]>fs[k]) k=j;
       ans[i][0]+=fs[k];
       ans[i][k+1]++;
       time--;
       if(fs[k]>0)
       {
          fs[k]-=d[k];
          if(fs[k]<0) fs[k]=0;
       }
       flag=0;
       for(j=0;j<i;j++)
```

```
                if(fs[j]<=0) flag++;
        }
        //将剩余时间加到一个湖中用时
        ans[i][1]+=time;
      }
   k=1;
   for(j=2;j<=n;j++)
      if(ans[j][0]>ans[k][0]) k=j;
   for(j=1;j<=n;j++)
   {
    cout<<ans[k][j]*5;
    if(j!=n) cout<<", ";
   }
   cout<<endl;
   cout<<"Number of fish expected: "<<ans[k][0]<<endl;
   cout<<endl;
  }
  return 0;
}
```

Pku1700--Crossing River

Description

A group of N people wishes to go across a river with only one boat, which can at most carry two persons. Therefore some sort of shuttle arrangement must be arranged in order to row the boat back and forth so that all people may cross. Each person has a different rowing speed; the speed of a couple is determined by the speed of the slower one. Your job is to determine a strategy that minimizes the time for these people to get across.

Input

The first line of the input contains a single integer T (1 <= T <= 20), the number of test cases. Then T cases follow. The first line of each case contains N, and the second line contains N integers giving the time for each people to cross the river. Each case is preceded by a blank line. There won't be more than 1000 people and nobody takes more than 100 seconds to cross.

Output

For each test case, print a line containing the total number of seconds required for all the N people to cross the river.

Sample Input

```
1
4
1 2 5 10
```

Sample Output

```
17
```

```
#include<iostream>
#include<algorithm>
using namespace std;
int main()
{
    int a,b,i,j,t,n,*s,sum, cas;
    cin >> cas;
    while (cas--) {
    cin>>n;
    s=new int[n];
    for(j=0;j<n;j++)
        cin>>s[j];
    sort(&s[0],&s[n]);
    sum=0;
    for(j=n-1;j>2;j-=2) {
        a=2*s[0]+s[j]+s[j-1];
        b=2*s[1]+s[0]+s[j];
        if(a>b)
            sum+=b;
        else
            sum+=a;
    }
    if(j==2)
        sum+=s[0]+s[1]+s[2];
    else
        if(j==1)
            sum+=s[1];
    else
        sum+=s[0];
    delete[] s;
    cout<<sum<<endl;
```

```
    }
    return 0;
}
```

1.2.3 递归与分治算法

1.2.4 递推

Pku1090-Chain

Description

Byteland had not always been a democratic country. There were also black pages in its book of history. One lovely day general Bytel − commander of the junta which had power over Byteland −− decided to finish the long−lasting time of war and released imprisoned activists of the opposition. However, he had no intention to let the leader Bytesar free. He decided to chain him to the wall with the bytish chain. It consists of joined rings and the bar fixed to the wall. Although the rings are not joined with the bar, it is hard to take them off.
'General, why have you chained me to the prison walls and did not let rejoice at freedom!' cried Bytesar.
'But Bytesar, you are not chained at all, and I am certain you are able to take off the rings from the bar by yourself.' perfidiously answered general Bytel, and he added 'But deal with that before a clock strikes the cyber hour and do not make a noise at night, otherwise I will be forced to call Civil Cyber Police.'
Help Bytesar! Number the following rings of the chain with integers 1,2,...,n. We may put on and take off these rings according to the following rules:
.only one ring can be put on or taken off from the bar in one move,
.the ring number 1 can be always put on or taken off from the bar,
.if the rings with the numbers 1,...,k−1 (for 1<= k < n) are taken off from the bar and the ring number k is put on, we can put on or take off the ring number k+1.
Write a program which:
.reads from std input the description of the bytish chain,
.computes minimal number of moves necessary to take off all rings of the bytish chain from the bar,
.writes the result to std output.

Input

In the first line of the input there is written one integer n, 1 <= n <= 1000. In the second line there are written n integers o1,o2,...,on (each of them is either 0 or 1) separated by single spaces. If oi=1, then the i−th ring is put on the bar, and if oi=0, then the i−th ring is taken off the bar.

Output

The output should contain exactly one integer equal to the minimal number of moves necessary to take off all the rings of the bytish chain from the bar.

Sample Input

```
4
1 0 1 0
```

Sample Output

```
6
```

给定了一个由 0 和 1 组成的字符串 S (1 <= n<= 1000)

允许进行以下两种操作：

将 S 的第一个字符取反。

若 S[1 .. i – 1] = ‘000…00’，S[i] = ‘1’，那么将 S[i + 1] 取反。（1 <= i <= n – 1）

```
#include<iostream>
#include<cstring>
using namespace std;
void plus(char *,char *,char *);//加法运算
const int MAXLEN=330;//最大长度
int main()
{
    int total;//记录字符串的长度
    char sign[1000];//记录输入的字符串
    char change[MAXLEN+1];//记录 00To01[n]
    char temp[MAXLEN+1];//作为中间替代数组用
    char BeforeResult[2][MAXLEN+1];//每次运算前的结果
    char AfterResult[2][MAXLEN+1];//每次运算后的结果
    int i,j,length;
    cin>>total;
```

```
    change[MAXLEN]=BeforeResult[0][MAXLEN]=AfterResult[0][MAXLEN]
        =BeforeResult[1][MAXLEN]=AfterResult[1][MAXLEN]='\0';
    //尾部为'\0',才为字符串
    while(cin)
    {
        for(i=0;i<MAXLEN;i++)
        BeforeResult[0][i]=BeforeResult[1][i]=change[i]='0';//清空
        change[MAXLEN-1]='1';//仅 1 位数时,00 To 01  需 1 步
        length=-1;
        for(i=0;i<total;i++)
        {
            cin>>sign[i];
            if(sign[i]=='1') length=i+1;//记录字符串中 1 出现的最后位置
        }
        if(length==-1) cout<<0<<endl;//若字符串全为 0,输出 0
        else {

       /* a[1]=1  则  s[1]To00=1 s[1]To01=0
        a[1]=0  则  s[1]To00=0 s[1]To01=1
  所以 sign[1]='0'则  BeforeResult[1][MAXLEN-1]=0 BeforeResult[0][MAXLEN-1]=1
sign[0]='1'  则  BeforeResult[1][MAXLEN-1]=1 BeforeResult[0][MAXLEN-1]=0*/
        if(sign[0]=='0')
            BeforeResult[1][MAXLEN-1]='1';
        else
            BeforeResult[0][MAXLEN-1]='1';

        /*00To01[n]=01To00[n]=2*00To01[n-1]+1
         If a[n]=0
            s[n]To00=s[n-1]To00
            s[n]To01=s[n-1]To01+1+01To00[n-1]
         If a[n]=1
            s[n]To00=s[n-1]To01+1+01To00[n-1]
            s[n]To01=s[n-1]To00
        注意:数组从 0 开始,而字符串从 1 开始
        */
        for(i=1;i<length;i++)
        {
            if(sign[i]=='0')
            {
                plus(BeforeResult[1],change,AfterResult[1]);
                strcpy(BeforeResult[1],AfterResult[1]);

                //s[n]To00=s[n-1]To00,故 BeforeResult[0]不变,故没有运算
            }
```

```
                else
                {     strcpy(temp,BeforeResult[0]);
                 //记录下 BeforeResult[0],以便 s[n]To01=s[n-1]To00 用
                      plus(BeforeResult[1],change,AfterResult[0]);
                      strcpy(BeforeResult[0],AfterResult[0]);
                      strcpy(BeforeResult[1],temp);
                 }
               for(j=0;j<MAXLEN;j++)
                   temp[j]='0';
                plus(change,change,temp);
                strcpy(change,temp);
}
       i=0;
       while(BeforeResult[0][i]=='0')
            i++;//找到第一个不为 0 的位置
       while(i<MAXLEN)
       {     cout<<BeforeResult[0][i];i++;}
       cout<<endl;
          }
       cin>>total;
}
return 0;
}
void plus(char * num1,char * num2,char * result)//实现 1+num1+num2
{
     int i,x,y;
     y=1;
     for(i=MAXLEN-1;i>=0;i--)
     {
          x=num1[i]-'0'+num2[i]-'0'+y;
          y=x/10;
          x=x%10;
          result[i]=x+'0';
     }
}
```

Pku2351--Time Zones

Description

Prior to the late nineteenth century, time keeping was a purely local phenomenon. Each town would set their clocks to noon when the sun reached its zenith each day. A clockmaker or town clock would be the "official" time and the citizens would set their

pocket watches and clocks to the time of the town - enterprising citizens would offer their services as mobile clock setters, carrying a watch with the accurate time to adjust the clocks in customer's homes on a weekly basis. Travel between cities meant having to change one's pocket watch upon arrival.

However, once railroads began to operate and move people rapidly across great distances, time became much more critical. In the early years of the railroads, the schedules were very confusing because each stop was based on a different local time. The standardization of time was essential to efficient operation of railroads.

In 1878, Canadian Sir Sanford Fleming proposed the system of worldwide time zones that we use today. He recommended that the world be divided into twenty-four time zones, each spaced 15 degrees of longitude apart. Since the earth rotates once every 24 hours and there are 360 degrees of longitude, each hour the earth rotates one-twenty-fourth of a circle or 15° of longitude. Sir Fleming's time zones were heralded as a brilliant solution to a chaotic problem worldwide.

United States railroad companies began utilizing Fleming's standard time zones on November 18, 1883. In 1884 an International Prime Meridian Conference was held in Washington D.C. to standardize time and select the Prime Meridian. The conference selected the longitude of Greenwich, England as zero degrees longitude and established the 24 time zones based on the Prime Meridian. Although the time zones had been established, not all countries switched immediately. Though most U.S. states began to adhere to the Pacific, Mountain, Central, and Eastern time zones by 1895, Congress didn't make the use of these time zones mandatory until the Standard Time Act of 1918.

Today, many countries operate on variations of the time zones proposed by Sir Fleming. All of China (which should span five time zones) uses a single time zone - eight hours ahead of Coordinated Universal Time (known by the abbreviation UTC - based on the time zone running through Greenwich at 0° longitude). Russia adheres to its designated time zones although the entire country is on permanent Daylight Saving Time and is an hour ahead of their actual zones. Australia uses three time zones - its central time zone is a half-hour ahead of its designated time zone. Several countries in the Middle East and South Asia also utilize half-hour time zones.

Since time zones are based on segments of longitude and lines of longitude narrow at the poles, scientists working at the North and South Poles simply use UTC time. Otherwise, Antarctica would be divided into 24 very thin time zones!

Time zones have recently been given standard capital-letter abbreviations as follows:

UTC Coordinated Universal Time
GMT Greenwich Mean Time, defined as UTC
BST British Summer Time, defined as UTC+1 hour
IST Irish Summer Time, defined as UTC+1 hour
WET Western Europe Time, defined as UTC
WEST Western Europe Summer Time, defined as UTC+1 hour
CET Central Europe Time, defined as UTC+1
CEST Central Europe Summer Time, defined as UTC+2
EET Eastern Europe Time, defined as UTC+2

EEST Eastern Europe Summer Time, defined as UTC+3
MSK Moscow Time, defined as UTC+3
MSD Moscow Summer Time, defined as UTC+4
AST Atlantic Standard Time, defined as UTC-4 hours
ADT Atlantic Daylight Time, defined as UTC-3 hours
NST Newfoundland Standard Time, defined as UTC-3.5 hours
NDT Newfoundland Daylight Time, defined as UTC-2.5 hours
EST Eastern Standard Time, defined as UTC-5 hours
EDT Eastern Daylight Saving Time, defined as UTC-4 hours
CST Central Standard Time, defined as UTC-6 hours
CDT Central Daylight Saving Time, defined as UTC-5 hours
MST Mountain Standard Time, defined as UTC-7 hours
MDT Mountain Daylight Saving Time, defined as UTC-6 hours
PST Pacific Standard Time, defined as UTC-8 hours
PDT Pacific Daylight Saving Time, defined as UTC-7 hours
HST Hawaiian Standard Time, defined as UTC-10 hours
AKST Alaska Standard Time, defined as UTC-9 hours
AKDT Alaska Standard Daylight Saving Time, defined as UTC-8 hours
AEST Australian Eastern Standard Time, defined as UTC+10 hours
AEDT Australian Eastern Daylight Time, defined as UTC+11 hours
ACST Australian Central Standard Time, defined as UTC+9.5 hours
ACDT Australian Central Daylight Time, defined as UTC+10.5 hours
AWST Australian Western Standard Time, defined as UTC+8 hours
Given the current time in one time zone, you are to compute what time it is in another time zone.

Input

The first line of input contains N, the number of test cases. For each case a line is given with a time, and 2 time zone abbreviations. Time is given in standard a.m./p.m. format with midnight denoted "midnight" and noon denoted "noon" (12:00 a.m. and 12:00 p.m. are oxymorons).

Output

For each case, assuming the given time is the current time in the first time zone, give the current time in the second time zone.

Sample Input

```
4
noon HST CEST
```

```
11:29 a.m. EST GMT
6:01 p.m. CST UTC
12:40 p.m. ADT MSK
```

Sample Output

```
midnight
4:29 p.m.
12:01 a.m.
6:40 p.m.
```

```
#pragma warning (disable:4786)
#include <iostream>
#include <string>
#include <map>
using namespace std;
int Count=0;
map<string,int> Score;
string p[10000];
void getstring(char *st)
{
 int i=0;
 while(1)
 {
  char ch;
  ch=getchar();
  if(ch=='\n')
   break;
  st[i]=ch;
  i++;
 }
}
void Read()
{
 int n;
 cin>>n;
 int i=0;
 Count=n;
 getchar();
 for(i=0;i<n;i++)
 {
  char ch[31];
  memset(ch,0,sizeof(ch));
  getstring(ch);
  p[i]=ch;
```

```
   Score[p[i]]=0;//这样对应起来--p[i]与 score 对应起来
  }
  int i_exam;
  cin>>i_exam;
  for(i=0;i<i_exam;i++)
  {
   int scr=0;//保存 liming  的分数
   for(int j=0;j<Count;j++)
   {
    int sc;
    cin>>sc;
    getchar();
    string str="";
    char ch[31];
    memset(ch,0,sizeof(ch));
    getstring(ch);
    str=ch;
    Score[str]+=sc;
   }
   int Co=1;
   for(int k=0;k<Count;k++)
   {
    if(Score[p[k]]>Score["Li Ming"])
    Co++;
   }
   cout<<Co<<endl;
  }
}
int main()
{
  Read();
  return 1;
}
```

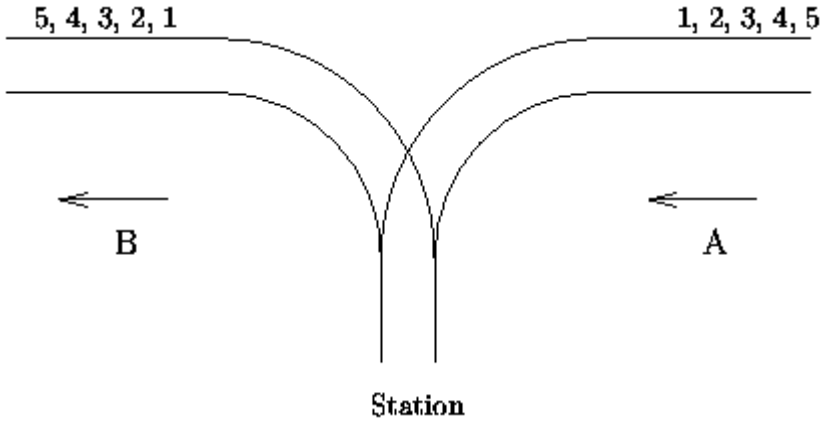
1.3 数据结构（1）------入门

1.3.1 栈和队列

Pku1363--Rails

Description

There is a famous railway station in PopPush City. Country there is incredibly hilly. The station was built in last century. Unfortunately, funds were extremely limited that time. It was possible to establish only a surface track. Moreover, it turned out that the station could be only a dead-end one (see picture) and due to lack of available space it could have only one track.



The local tradition is that every train arriving from the direction A continues in the direction B with coaches reorganized in some way. Assume that the train arriving from the direction A has N <= 1000 coaches numbered in increasing order 1, 2, ..., N. The chief for train reorganizations must know whether it is possible to marshal coaches continuing in the direction B so that their order will be a1, a2, ..., aN. Help him and write a program that decides whether it is possible to get the required order of coaches. You can assume that single coaches can be disconnected from the train before they enter the station and that they can move themselves until they are on the track in the direction B. You can also suppose that at any time there can be located as many coaches as necessary in the station. But once a coach has entered the station it cannot return to the track in the direction A and also once it has left the station in the direction B it cannot return back to the station.

Input

The input consists of blocks of lines. Each block except the last describes one train and possibly more requirements for its reorganization. In the first line of the block there is the integer N described above. In each of the next lines of the block there is a permutation of 1, 2, ..., N. The last line of the block contains just 0.
The last block consists of just one line containing 0.

Output

The output contains the lines corresponding to the lines with permutations in the input. A line of the output contains Yes if it is possible to marshal the coaches in the order required on the corresponding line of the input. Otherwise it contains No. In addition,

there is one empty line after the lines corresponding to one block of the input. There is no line in the output corresponding to the last ``null'' block of the input.

Sample Input

```
5
1 2 3 4 5
5 4 1 2 3
0
6
6 5 4 3 2 1
0
0
```

Sample Output

```
Yes
No
Yes
```

```
#include <stdio.h>
#include <string.h>
#define N 1005
int a[N], b[N], c[N];
int x, y, z;
int main(void)
{
    int n, i;
    while (scanf("%d", &n), n) {
        while (scanf("%d", &c[0]), c[0]) {
            for (i = 1; i < n; i++)
                scanf("%d", &c[i]);
            for (i = 0; i < n; i++)
                a[i] = i + 1;
            memset(b, 0, sizeof(b));
            b[0] = a[0];
            x = 1;
            y = 1;
            for (z = 0; z < n; z++) {
                while (c[z] != b[y-1] && x < n) {
                    b[y] = a[x];
                    y++;
                    x++;
                }
```

```
                    if (c[z] == b[y-1]) {
                        y --;
                    } else if (x >= n) {
                        break;
                    }
                }
                if (z < n)
                    printf("No\n");
                else
                    printf("Yes\n");
            }
            puts("");
        }
        return 0;
}
```

Pku1879-Tempus et mobilius Time and motion

Description

Tempus est mensura motus rerum mobilium.
Time is the measure of movement.--*Auctoritates Aristotelis*

...and movement has long been used to measure time. For example, the ball clock is a simple device which keeps track of the passing minutes by moving ball-bearings. Each minute, a rotating arm removes a ball bearing from the queue at the bottom, raises it to the top of the clock and deposits it on a track leading to indicators displaying minutes, five-minutes and hours. These indicators display the time between 1:00 and 12:59, but without 'a.m.' or 'p.m.' indicators. Thus 2 balls in the minute indicator, 6 balls in the five-minute indicator and 5 balls in the hour indicator displays the time 5:32.

Unfortunately, most commercially available ball clocks do not incorporate a date indication, although this would be simple to do with the addition of further carry and indicator tracks. However, all is not lost! As the balls migrate through the mechanism of the clock, they change their relative ordering in a predictable way. Careful study of these orderings will therefore yield the time elapsed since the clock had some specific ordering. The length of time which can be measured is limited because the orderings of the balls eventually begin to repeat. Your program must compute the time before repetition, which varies according to the total number of balls present.

Operation of the Ball Clock

Every minute, the least recently used ball is removed from the queue of balls at the bottom of the clock, elevated, then deposited on the minute indicator track, which is able to hold four balls. When a fifth ball rolls on to the minute indicator track, its weight causes the track to tilt. The four balls already on the track run back down to join the queue of balls waiting at the bottom in reverse order of their original addition to the minutes track. The fifth ball, which caused the tilt, rolls on down to the five-minute indicator track. This track holds eleven balls. The twelfth ball carried over from the minutes causes the five-minute track to tilt, returning the eleven balls to the queue, again in reverse order of their addition. The twelfth ball rolls down to the hour indicator. The hour indicator also holds eleven balls, but has one extra fixed ball which is always present so that counting the balls in the hour indicator will yield an hour in the range one to twelve. The twelfth ball carried over from the five-minute indicator causes the hour indicator to tilt, returning the eleven free balls to the queue, in reverse order, before the twelfth ball itself also returns to the queue.

Input

The input defines a succession of ball clocks. Each clock operates as described above. The clocks differ only in the number of balls present in the queue at one o'clock when all the clocks start. This number is given for each clock, one per line and does not include the fixed ball on the hours indicator. Valid numbers are in the range 27 to 127. A zero signifies the end of input.

Output

Output For each clock described in the input, your program should report the number of balls given in the input and the number of days (24-hour periods) which elapse before the clock returns to its initial ordering.

Sample Input

```
30
45
```

```
0
```

Sample Output

```
30 balls cycle after 15 days.
45 balls cycle after 378 days.
```

```
#include <iostream>
#include <stack>
#include <queue>
using namespace std;

int gcd(int xx, int yy) {
	if (yy != 0)
		return gcd(yy, xx%yy);
	return xx;
}
int main(void) {
	int n;
	while (cin >> n && n) {
		queue<int>all_ball;
		stack<int>five, mini, hour;
		for (int i = 1; i <= n; i++)
			all_ball.push(i);
		int k = 24*60;
		while (k--) {
			int aa = all_ball.front();
			all_ball.pop();
			if (five.size() == 4) {
				for (int i = 0; i < 4; i++) {
					all_ball.push(five.top());
					five.pop();
				}
				if (mini.size() == 11) {
					for (int i = 0; i < 11; i++) {
						all_ball.push(mini.top());
						mini.pop();
					}
					if (hour.size() == 11) {
						for (int i = 0; i < 11; i++) {
							all_ball.push(hour.top());
							hour.pop();
						}
						all_ball.push(aa);
					} else
```

```
                        hour.push(aa);
                } else
                        mini.push(aa);
            } else
                    five.push(aa);
        }
        int arr[128];
        for (int i = 1; i <= n; i++) {
            arr[i] = all_ball.front();
//          cout << arr[i] << endl;
            all_ball.pop();
        }
        int brr[128];
        for (int i = 1; i <= n; i++)
            brr[i] = 1;
        for (int i = 1; i <= n; i++) {
            int aa = arr[i];
            while (aa != i) {
                brr[i]++;
                aa = arr[aa];
            }
        }
        int sum = 1;
        for (int i = 1; i <= n; i++) {
            sum = sum*brr[i]/gcd(sum, brr[i]);
        }
        cout << n << " balls cycle after " << sum << " days." << endl;
    }
    return 0;
}
```

1.3.2 串

Pku1961-Period

Description

For each prefix of a given string S with N characters (each character has an ASCII code between 97 and 126, inclusive), we want to know whether the prefix is a periodic string. That is, for each i (2 <= i <= N) we want to know the largest K > 1 (if there is one) such that the prefix of S with length i can be written as A^K ,that is A

concatenated K times, for some string A. Of course, we also want to know the period K.

Input

The input consists of several test cases. Each test case consists of two lines. The first one contains N (2 <= N <= 1 000 000) – the size of the string S.The second line contains the string S. The input file ends with a line, having the
number zero on it.

Output

For each test case, output "Test case #" and the consecutive test case number on a single line; then, for each prefix with length i that has a period K > 1, output the prefix size i and the period K separated by a single space; the prefix sizes must be in increasing order. Print a blank line after each test case.

Sample Input

```
3
aaa
12
aabaabaabaab
0
```

Sample Output

```
Test case #1
2 2
3 3

Test case #2
2 2
6 2
9 3
12 4
```

```
#include <stdio.h>
#include <string.h>
#define N 1000005
char arr[N];
int next[N], len;
```

```
void get_next(void) {
	int i, j;
	i = 1;
	next[1] = 0;
	j = 0;
	while (i <= len) {
		if (j == 0 || arr[i] == arr[j]) {
			++i;
			++j;
			next[i] = j;
		} else {
			j = next[j];
		}
	}
	return ;
}
int main(void) {
	int ncas, k, count;

	ncas = 1;
	while (scanf ("%d", &len), len) {
		printf ("Test case #%d\n", ncas++);
		scanf ("%s", &arr[1]);
		get_next();
/*		for (k = 1; k <= len; k++)
			printf ("%d ", next[k]);
		puts("");*/
		for (k = 2; k <= len; k++) {
			if (k%((k+1)-next[k+1]) == 0) {
				count = k/((k+1)-next[k+1]);
				if (count != 1)
					printf("%d %d\n", k, count);
			}
		}
		puts("");
	}
	return 0;
}
```

Pku2406--Power Strings

Description

Given two strings a and b we define a*b to be their concatenation. For example, if a = "abc" and b = "def" then a*b = "abcdef". If we think of concatenation as multiplication, exponentiation by a non-negative integer is defined in the normal way: a^0 = "" (the empty string) and a^(n+1) = a*(a^n).

Input

Each test case is a line of input representing s, a string of printable characters. The length of s will be at least 1 and will not exceed 1 million characters. A line containing a period follows the last test case.

Output

For each s you should print the largest n such that s = a^n for some string a.

Sample Input

```
abcd
aaaa
abababab
.
```

Sample Output

```
1
4
3
```

```
#include <stdio.h>
#include <string.h>
#define N 1000005
char arr[N];
int next[N], len;
void get_next(void) {
    int i, j;
    i = 1;
    next[1] = 0;
    j = 0;
    while (i <= len) {
        if (j == 0 || arr[i] == arr[j]) {
            ++i;
            ++j;
```

```
                next[i] = j;
            } else {
                j = next[j];
            }
        }
        return ;
}
int main(void) {
        int k, count;
        while (scanf ("%s", &arr[1]), arr[1] != '.') {
            len = strlen(&arr[1]);
            get_next();
            if (len % ((len + 1) - next[len + 1]) == 0)
                count = len/(len + 1 - next[len + 1]);
            else
                count = 1;
            printf ("%d\n", count);
        }
        return 0;
}
```

Pku2752--Seek the Name, Seek the Fame

Description

The little cat is so famous, that many couples tramp over hill and dale to Byteland, and asked the little cat to give names to their newly-born babies. They seek the name, and at the same time seek the fame. In order to escape from such boring job, the innovative little cat works out an easy but fantastic algorithm:

Step1. Connect the father's name and the mother's name, to a new string S.
Step2. Find a proper prefix-suffix string of S (which is not only the prefix, but also the suffix of S).

Example: Father='ala', Mother='la', we have S = 'ala'+'la' = 'alala'. Potential prefix-suffix strings of S are {'a', 'ala', 'alala'}. Given the string S, could you help the little cat to write a program to calculate the length of possible prefix-suffix strings of S? (He might thank you by giving your baby a name:)

Input

The input contains a number of test cases. Each test case occupies a single line that contains the string S described above.
Restrictions: Only lowercase letters may appear in the input. 1 <= Length of S <= 400000.

Output

For each test case, output a single line with integer numbers in increasing order, denoting the possible length of the new baby's name.

Sample Input

```
ababcababababcabab
aaaaa
```

Sample Output

```
2 4 9 18
1 2 3 4 5
```

```
#include <stdio.h>
#include <string.h>
#define N 400005
char T[N];
int len, result[N], next[N];
void get_next() {
	int i, j;
	i = 1;
	next[1] = 0;
	j = 0;
	while (i <= len) {
		if (j == 0 || T[i] == T[j]) {
			i++;
			j++;
			next[i] = j;
		} else {
			j = next[j];
		}
	}
	return ;
}
int main(void) {
	int j, index, i;
```

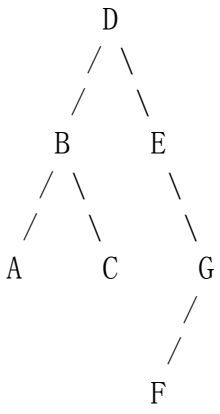
```
    while (scanf ("%s", &T[1]) != EOF) {
        len = strlen(&T[1]);
        get_next();
        j = len+1;
        index = 0;
        while (j != 1) {
            result[++index] = j;
            j = next[j];
        }
        for (i = index; i > 0; i--) {
            printf ("%d ", result[i]-1);
        }
        puts("");
    }
    return 0;
}
```

1.3.3 树和二叉树

Pku2255--Tree Recovery

Description

Little Valentine liked playing with binary trees very much. Her favorite game was constructing randomly looking binary trees with capital letters in the nodes.
This is an example of one of her creations:



To record her trees for future generations, she wrote down two strings for each tree: a preorder traversal (root, left subtree, right subtree) and an inorder traversal (left subtree, root, right subtree). For the tree drawn above the preorder traversal is DBACEGF and the inorder traversal is ABCDEFG.
She thought that such a pair of strings would give enough information to reconstruct the tree later (but she never tried it).

Now, years later, looking again at the strings, she realized that reconstructing the trees was indeed possible, but only because she never had used the same letter twice in the same tree.
However, doing the reconstruction by hand, soon turned out to be tedious.
So now she asks you to write a program that does the job for her!

Input

The input will contain one or more test cases.
Each test case consists of one line containing two strings preord and inord, representing the preorder traversal and inorder traversal of a binary tree. Both strings consist of unique capital letters. (Thus they are not longer than 26 characters.)
Input is terminated by end of file.

Output

For each test case, recover Valentine's binary tree and print one line containing the tree's postorder traversal (left subtree, right subtree, root).

Sample Input

```
DBACEGF ABCDEFG
BCAD CBAD
```

Sample Output

```
ACBFGED
CDAB
```

```
#include <iostream>
#include <string>
using namespace std;
void opt(string a, string b)
{
    int len = a.length();
    if (len == 1) {
        cout << a[0];
        return;
    }
    int t = a.find(b[0]);
    string aa(a, 0, t);
    string bb(a, t+1, len-t-1);
```

```
    string cc(b, 1, t);
    string dd(b, t+1, len-t-1);
    if (aa.length() > 0)
        opt(aa, cc);
    if (bb.length() > 0)
    opt(bb, dd);
    cout << b[0];
}
int main(void)
{
    string arr, brr;
    cin >> arr;
    cin >> brr;
    opt(arr, brr);
    cout << endl;
    return 0;
}
```

Pku1470--Closest Common Ancestors

Description

Write a program that takes as input a rooted tree and a list of pairs of vertices. For each pair (u,v) the program determines the closest common ancestor of u and v in the tree. The closest common ancestor of two nodes u and v is the node w that is an ancestor of both u and v and has the greatest depth in the tree. A node can be its own ancestor (for example in Figure 1 the ancestors of node 2 are 2 and 5)

Input

The data set, which is read from a the std input, starts with the tree description, in the form:

nr_of_vertices
vertex:(nr_of_successors) successor1 successor2 ... successorn
...
where vertices are represented as integers from 1 to n (n <= 900). The tree description is followed by a list of pairs of vertices, in the form:
nr_of_pairs
(u v) (x y) ...
The input file contents several data sets (at least one).
Note that white-spaces (tabs, spaces and line breaks) can be used freely in the input.

Output

For each common ancestor the program prints the ancestor and the number of pair for which it is an ancestor. The results are printed on the standard output on separate lines, in to the ascending order of the vertices, in the format: ancestor:times
For example, for the following tree:

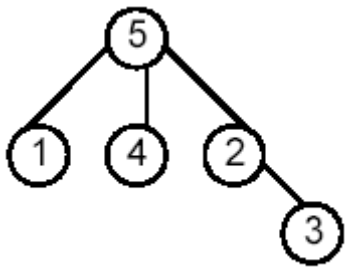


Figure 1

Sample Input

```
5
5:(3) 1 4 2
1:(0)
4:(0)
2:(1) 3
3:(0)
6
(1 5) (1 4) (4 2)
     (2 3)
(1 3) (4 3)
```

Sample Output

```
2:1
5:5
Hint:lca->rmq,very good!!!!!!
#include<iostream>
#include<cmath>
#include<vector>
#include<algorithm>
using namespace std;
const int MAX=2010;
int a[MAX],f[MAX][30],size;
vector<vector<int> >adj;
int mark[MAX],top=1,ccount[MAX]={0},level[MAX];
```

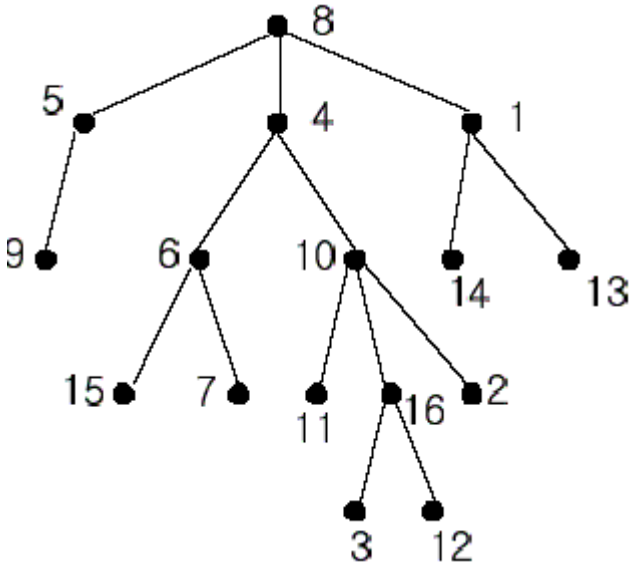
```
int mmin(int a,int b){return level[a]<level[b]?a:b;}
void init()
{
    int i,j,k,s,t,r;
    int m=floor(log(size+0.0)/log(2.0));
    for(i=1;i<=size;++i)
        f[i][0]=a[i];
    for(j=1;j<=m;++j)
    {
        k=1<<(j-1);
        for(i=1;i+(1<<j)<=size;++i)
            f[i][j]=mmin(f[i][j-1],f[i+k][j-1]);
    }
}
int query(int l,int r)
{
    if(l>r)swap(l,r);
    int k=(int)(log(r-l+1.0)/log(2.0));
    return mmin(f[l][k],f[r-(1<<k)+1][k]);
}
void dfs(int k)
{
    int i;
    mark[k]=top;
    a[top++]=k;
    for(i=0;i<adj[k].size();++i)
    {
        level[adj[k][i]]=level[k]+1;
        dfs(adj[k][i]);
        a[top++]=k;
    }
    return;
}
int main()
{
    int i,j,k,s,t,r;
    char c;
    int n,m;
    while(scanf("%d",&n)!=EOF)
    {
        adj.clear();
        adj.resize(n+1);
        memset(mark,0,sizeof(mark));
        memset(ccount,0,sizeof(ccount));
```

```
        for(i=0;i<n;++i)
        {
            scanf("%d",&k);
            while((c=getchar())!='(');
            scanf("%d",&s);
            while((c=getchar())!=')');
            for(j=0;j<s;++j)
            {
                scanf("%d",&t);
                adj[k].push_back(t);
                mark[t]=k;
            }
        }
        for(i=1;i<=n;++i)
            if(mark[i]==0)break;
        level[i]=1;
        top=1;
        dfs(i);
        size=top-1;
        init();
        scanf("%d",&m);
        while(m--)
        {
            while((c=getchar())!='(');
            scanf("%d%d",&i,&j);
            ++ccount[query(mark[i],mark[j])];
            while((c=getchar())!=')');
        }
        for(i=1;i<=n;++i)
        {
            if(ccount[i])
                printf("%d:%d\n",i,ccount[i]);
        }
    }
    return 0;
}
```

Pku1330--Nearest Common Ancestors

Description

A rooted tree is a well-known data structure in computer science and engineering. An example is shown below:



In the figure, each node is labeled with an integer from {1, 2,...,16}. Node 8 is the root of the tree. Node x is an ancestor of node y if node x is in the path between the root and node y. For example, node 4 is an ancestor of node 16. Node 10 is also an ancestor of node 16. As a matter of fact, nodes 8, 4, 10, and 16 are the ancestors of node 16. Remember that a node is an ancestor of itself. Nodes 8, 4, 6, and 7 are the ancestors of node 7. A node x is called a common ancestor of two different nodes y and z if node x is an ancestor of node y and an ancestor of node z. Thus, nodes 8 and 4 are the common ancestors of nodes 16 and 7. A node x is called the nearest common ancestor of nodes y and z if x is a common ancestor of y and z and nearest to y and z among their common ancestors. Hence, the nearest common ancestor of nodes 16 and 7 is node 4. Node 4 is nearer to nodes 16 and 7 than node 8 is.

For other examples, the nearest common ancestor of nodes 2 and 3 is node 10, the nearest common ancestor of nodes 6 and 13 is node 8, and the nearest common ancestor of nodes 4 and 12 is node 4. In the last example, if y is an ancestor of z, then the nearest common ancestor of y and z is y.

Write a program that finds the nearest common ancestor of two distinct nodes in a tree.

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 starts with a line containing an integer N , the number of nodes in a tree, 2<=N<=10,000. The nodes are labeled with integers 1, 2,..., N. Each of the next N -1 lines contains a pair of integers that represent an edge --the first integer is the parent node of the second integer. Note that a tree with N nodes has exactly N - 1 edges. The last line of each test case contains two distinct integers whose nearest common ancestor is to be computed.

Output

Print exactly one line for each test case. The line should contain the integer that is the nearest common ancestor.

Sample Input

```
2
16
1 14
8 5
10 16
5 9
4 6
8 4
4 10
1 13
6 15
10 11
6 7
10 2
16 3
8 1
16 12
16 7
5
2 3
3 4
3 1
1 5
3 5
```

Sample Output

```
4
3
```

Hint:tarjan 算法，very very good!

```
#include<iostream>
#include<vector>
using namespace std;
#define maxn 10001
```

```
int pnt[maxn],ancestor[maxn];
bool root[maxn],color[maxn];
vector<int> child[maxn];
int first,second,ret;
int find(int x)
{
    if(pnt[x]!=x)pnt[x]=find(pnt[x]);
    return pnt[x];
}
void unionset(int x,int y)
{
    x=find(x);y=find(y);
    pnt[y]=x;
}
void Lcancestor(int parent)
{
    pnt[parent]=parent;
    ancestor[find(parent)]=parent;
    for(int i=0;i<child[parent].size();i++)
    {
        Lcancestor(child[parent][i]);
        unionset(parent,child[parent][i]);
        ancestor[find(child[parent][i])]=parent;
    }
    color[parent]=true;
    if(parent==first&&color[second])
    {
        ret=ancestor[find(second)];
        return;
    }
    if(parent==second&&color[first])
    {
        ret=ancestor[find(first)];
        return;
    }
}
```

Pku3264--Balanced Lineup

Description

For the daily milking, Farmer John's *N* cows ($1 \le N \le 50{,}000$) always line up in the same order. One day Farmer John decides to organize a game of Ultimate Frisbee

with some of the cows. To keep things simple, he will take a contiguous range of cows from the milking lineup to play the game. However, for all the cows to have fun they should not differ too much in height.

Farmer John has made a list of Q ($1 \le Q \le 200{,}000$) potential groups of cows and their heights ($1 \le height \le 1{,}000{,}000$). For each group, he wants your help to determine the difference in height between the shortest and the tallest cow in the group.

Input

Line 1: Two space-separated integers, N and Q.
Lines 2..N+1: Line i+1 contains a single integer that is the height of cow i
Lines N+2..N+Q+1: Two integers A and B ($1 \le A \le B \le N$), representing the range of cows from A to B inclusive.

Output

Lines 1..Q: Each line contains a single integer that is a response to a reply and indicates the difference in height between the tallest and shortest cow in the range.

Sample Input

```
6 3
1
7
3
4
2
5
1 5
4 6
2 2
```

Sample Output

```
6
3
0
#include <iostream>
#include <cstdio>
```

```
using namespace std;
const int N = 500005;
int arr[N];
struct node {
	int l, r;
	int Min, Max;
};
node T[N];
void bulid(int k, int left, int right) {
	T[k].l = left;
	T[k].r = right;
	if (left == right) {
		T[k].Min = arr[left];
		T[k].Max = arr[right];
	} else {
		bulid(2*k, left, (left+right)/2);
		bulid(2*k+1, (left+right)/2+1, right);
		T[k].Min = min(T[2*k].Min, T[2*k+1].Min);
		T[k].Max = max(T[2*k].Max, T[2*k+1].Max);
	}
	return;
}
void quary(int k, int left, int right, int &Min, int &Max)
{
	int lc = 2*k, min1, min2, max1, max2;
	if (left == T[k].l && right == T[k].r) {
		Min = T[k].Min;
		Max = T[k].Max;
	} else if (T[lc+1].l <= left) {
		quary(lc+1, left, right, Min, Max);
	} else if (T[lc].r >= right) {
		quary(lc, left, right, Min, Max);
	} else {
		quary(lc, left, T[lc].r, min1, max1);
		quary(lc+1, T[lc+1].l, right, min2, max2);
		Min = min(min1, min2);
		Max = max(max1, max2);
	}
	return ;
}
int main(void) {
	int n, q, t, a, b, Min, Max;
	while(scanf("%d%d", &n, &q)!=EOF) {
	for (int i = 1; i <= n; i++)
```

```
            cin >> arr[i];
        bulid(1, 1, n);
        for (int i = 1; i <= q; i++) {
            scanf("%d%d", &a, &b);
            quary(1, a, b, Min, Max);
            cout << Max-Min << endl;
        }
        }
        return 0;
}
```

Hint->rmq

```
#include<iostream>
#include<cmath>
#include<algorithm>
using namespace std;
const int maxsize = 50005;
int a[maxsize], d[maxsize][18], e[maxsize][18];
int n, q;
void work()
{
    int i, j, m;
    for (i = 1; i <= n; i++)
    {
        d[i][0] = a[i];
        e[i][0] = a[i];
    }
    m = int(log(n * 1.0) / log(2.0));
    for (j = 1; j <= m; j++)
        for (i = 1; i <= n; i++)
        {
            d[i][j] = d[i][j - 1];
            e[i][j] = e[i][j - 1];
            if (i + (1 << (j - 1)) <= n)
            {
                d[i][j] = min(d[i][j], d[i + (1 << (j - 1))][j - 1]);
                e[i][j] = max(e[i][j], e[i + (1 << (j - 1))][j - 1]);
            }

        }
}
int RMQ(int l, int r, int flag)
{
    int m;
```

```
    m = int(log((r - l + 1) * 1.0) / log(2.0));
    if (flag == 0)
        return min(d[l][m], d[r - (1 << m )+ 1][m]);
    else return max(e[l][m], e[r - (1 << m) + 1][m]);
}
void init()
{
    int i, r, l, mini, maxi;
    scanf("%d%d", &n, &q);
    for (i = 1; i <= n; i++)
        scanf("%d", &a[i]);
    work();
    while (q--)
    {
        scanf("%d%d", &l, &r);
        mini = RMQ(l, r, 0);
        maxi = RMQ(l, r, 1);
        printf("%d\n", abs(mini - maxi));
    }
}

int main()
{
    init();
    return 0;
}
```

1.3.4 图及其基本算法

1.3.5 排序与检索算法

Pku1064--Cable master

Description

Inhabitants of the Wonderland have decided to hold a regional programming contest. The Judging Committee has volunteered and has promised to organize the most honest contest ever. It was decided to connect computers for the contestants using a "star" topology - i.e. connect them all to a single central hub. To organize a truly honest contest, the Head of the Judging Committee has decreed to place all contestants evenly around the hub on an equal distance from it.

To buy network cables, the Judging Committee has contacted a local network solutions provider with a request to sell for them a specified number of cables with equal lengths. The Judging Committee wants the cables to be as long as possible to sit contestants as far from each other as possible.
The Cable Master of the company was assigned to the task. He knows the length of each cable in the stock up to a centimeter,and he can cut them with a centimeter precision being told the length of the pieces he must cut. However, this time, the length is not known and the Cable Master is completely puzzled.
You are to help the Cable Master, by writing a program that will determine the maximal possible length of a cable piece that can be cut from the cables in the stock, to get the specified number of pieces.

Input

The first line of the input file contains two integer numb ers N and K, separated by a space. N (1 = N = 10000) is the number of cables in the stock, and K (1 = K = 10000) is the number of requested pieces. The first line is followed by N lines with one number per line, that specify the length of each cable in the stock in meters. All cables are at least 1 meter and at most 100 kilometers in length. All lengths in the input file are written with a centimeter precision, with exactly two digits after a decimal point.

Output

Write to the output file the maximal length (in meters) of the pieces that Cable Master may cut from the cables in the stock to get the requested number of pieces. The number must be written with a centimeter precision, with exactly two digits after a decimal point.
If it is not possible to cut the requested number of pieces each one being at least one centimeter long, then the output file must contain the single number "0.00" (without quotes).

Sample Input

```
4 11
8.02
7.43
4.57
5.39
```

Sample Output

2.00

```
#include <iostream>
#include <cstdio>

using namespace std;

long long arr[10005], maxx;
int n, k;

void input(void) {
	double f;
	maxx = 0;
	cin >> n >> k;
	for (int i = 0; i < n; i++) {
		cin >> f;
		arr[i] = (long long) (f*100);
		if (maxx < arr[i])
			maxx = arr[i];
	}
	return ;
}

void opt(void) {
	long long cnt, ans = 0;
	long long l = 1, r = maxx, mid;
	while (l <= r) {
		mid = (l+r)/2;
		cnt = 0;
		for (int i = 0; i < n; i++) {
			cnt += (long long) (arr[i]/mid);
		}
		if (cnt >= k && mid > ans)
			ans = mid;
		if (cnt >= k)
			l = mid+1;
		else
			r = mid-1;
	}
	printf("%.2lf\n", (double)(ans*0.01));
}
int main(void) {
	input();
	opt();
	return 0;
```

}

Pku1723--**SOLDIERS**

Description

N soldiers of the land Gridland are randomly scattered around the country.
A position in Gridland is given by a pair (x,y) of integer coordinates. Soldiers can move - in one move, one soldier can go one unit up, down, left or right (hence, he can change either his x or his y coordinate by 1 or -1).
The soldiers want to get into a horizontal line next to each other (so that their final positions are (x,y), (x+1,y), ..., (x+N-1,y), for some x and y). Integers x and y, as well as the final order of soldiers along the horizontal line is arbitrary.
The goal is to minimise the total number of moves of all the soldiers that takes them into such configuration.
Two or more soldiers must never occupy the same position at the same time.

Input

The first line of the input contains the integer N, 1 <= N <= 10000, the number of soldiers.
The following N lines of the input contain initial positions of the soldiers : for each i, 1 <= i <= N, the $(i+1)^{st}$ line of the input file contains a pair of integers x[i] and y[i] separated by a single blank character, representing the coordinates of the ith soldier, -10000 <= x[i],y[i] <= 10000.

Output

The first and the only line of the output should contain the minimum total number of moves that takes the soldiers into a horizontal line next to each other.

Sample Input

```
5
1 2
2 2
1 3
3 -2
3 3
```

Sample Output

8

```
#include<algorithm>
#include<cstdio>
#include<cmath>
using namespace std;
const int maxsize = 10005;
struct node{
	int x;
	int y;
};
node a[maxsize];
int n;
int cmp1(const node &b, const node &c)
{
	return b.x < c.x;
}
int cmp2(const node &b, const node &d)
{
	return b.y < d.y;
}
int main()
{
	int i, t;
	long long sum = 0;

	scanf("%d", &n);
	for (i = 0; i < n; i++)
		scanf("%d%d", &a[i].x, &a[i].y);
	sort(a, a + n, cmp2);
	for (i = 0; i < n; i++)
		sum += abs(a[i].y - a[n / 2].y);
	sort(a, a + n, cmp1);
	for (i = 0; i < n; i++)
	    a[i].x -= i;
		sort(a, a + n, cmp1);
	for (i = 0; i < n; i++)
		sum += abs(a[i].x - a[i / 2].x);
	printf("%lld\n", sum);
	return 0;
}
```

Pku1433--**Exchanges**

Description

Given n integer registers r1, r2, ..., rn we define a Compare-Exchange Instruction CE(a, b), where a, b are register indices (1 <= a < b <= n):
CE(a, b)::
if content(ra) > content(rb) then
exchange the contents of registers ra and rb;
A Compare-Exchange program (shortly CE-program) is any nite sequence of Compare-Exchange instructions. A CE-program is called a Minimum-Finding program if after its execution the register r1 always contains the smallest value among all values in the registers. Such program is called reliable if it remains a Minimum-Finding program after removing any single Compare-Exchange instruction. Given a CE-program P , what is the smallest number of instructions that should be added at the end of program P in order to get a reliable Minimum-Finding program?
Example
Consider the following CE-program for 3 registers:
CE(1, 2); CE(2, 3); CE(1, 2).
In order to make this program a reliable Minimum-Finding program it is sufficient to add only two instructions, CE(1, 3) and CE(1, 2).
Task
Write a program which for each data set:
reads the description of a CE-program,
computes the smallest number of CE-instructions that should be added to make this program a reliable Minimum-Finding program,
writes the result.

Input

The first line of the input contains exactly one positive integer d equal to the number of data sets, 1 <= d <=10. The data sets follow.
Each data set consists of exactly two consecutive lines.
The first of those lines contains exactly two integers n and m separated by a single space, 2 <= n <= 10 000, 0 <= m <= 25 000. Integer n is the number of registers and integer m is the number of program instructions.
The second of those lines contains exactly 2m integers separated by single spaces - the program
itself. Integers aj, bj on positions 2j -1 and 2j, 1 < j < m, 1 <= aj < bj <= n, are parameters of the j-th instruction in the program.

Output

The output should consist of exactly d lines, one line for each data set.
Line i, 1 <= i <= d, should contain only one integer - the smallest number of instructions that should be added at the end of the i-th input program in order to make this program a reliable Minimum-Finding program.

Sample Input

```
1
3 3
1 2 2 3 1 2
```

Sample Output

```
2
```

1.4 数据结构(2)-----拓宽与应用举例

1.4.1 并查集

Pku1703--Find them, Catch them

Description

The police office in Tadu City decides to say ends to the chaos, as launch actions to root up the TWO gangs in the city, Gang Dragon and Gang Snake. However, the police first needs to identify which gang a criminal belongs to. The present question is, given two criminals; do they belong to a same clan? You must give your judgment based on incomplete information. (Since the gangsters are always acting secretly.)
Assume N (N <= 10^5) criminals are currently in Tadu City, numbered from 1 to N. And of course, at least one of them belongs to Gang Dragon, and the same for Gang Snake. You will be given M (M <= 10^5) messages in sequence, which are in the following two kinds:
1. D [a] [b]
where [a] and [b] are the numbers of two criminals, and they belong to different gangs.

2. A [a] [b]

where [a] and [b] are the numbers of two criminals. This requires you to decide whether a and b belong to a same gang.

Input

The first line of the input contains a single integer T (1 <= T <= 20), the number of test cases. Then T cases follow. Each test case begins with a line with two integers N and M, followed by M lines each containing one message as described above.

Output

For each message "A [a] [b]" in each case, your program should give the judgment based on the information got before. The answers might be one of "In the same gang.", "In different gangs." and "Not sure yet."

Sample Input

```
1
5 5
A 1 2
D 1 2
A 1 2
D 2 4
A 1 4
```

Sample Output

```
Not sure yet.
In different gangs.
In the same gang.
```

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>

#define M      100001
struct Crime{
    int parent,weight;
};
Crime crime[M];          // 所用数据结构
```

```
int n,m;
int main( ){
    int k,one,two;
    char order;
// freopen("data.txt","r",stdin);
    int kk,X;
    scanf("%d",&X);
    for(kk=0;kk<X;kk++){
        memset(crime,0,sizeof(crime));        //初始化清零
        scanf("%d%d",&n,&m);
        for(k=0;k<m;k++){
            while(1){            // 清除空格等额外字符，有点多虑
                order=getchar();
                if(order=='D'||order=='A') break;
            }
            scanf("%d%d",&one,&two);
            if(order=='D'){     // 执行命令 D one two
                int temp1=one,temp2=two,t1=0,t2=0;
                while(crime[temp1].parent!=0) {
                    t1+=crime[temp1].weight;temp1=crime[temp1].parent;
                }// 查找 one 所在树的头节点，并计算到头节点的权之和
                while(crime[temp2].parent!=0) {
                    t2+=crime[temp2].weight;temp2=crime[temp2].parent;
                }      // 查找 two 所在树的头节点，并计算到头节点的权之和
                if(temp1!=temp2){
                    if(t1<t2){     // 若 t1<t2 ,把 one 所在树挂到 two 上
                        crime[temp1].parent=temp2;
                        crime[temp1].weight=(t1-t2+1)%2;
                    }
                    else {           // 反之，将 two 所在树挂到 one 上。
                        crime[temp2].parent=temp1;
                        crime[temp2].weight=(t1-t2+1)%2;
                    }
                }
            }
            else{       // 执行命令 A one two
                int temp1=one,temp2=two,t1=0,t2=0;
                while(crime[temp1].parent!=0) {
                    t1+=crime[temp1].weight;temp1=crime[temp1].parent;
                }     // 查找 one 所在树的头节点，并计算到头节点的权之和
                while(crime[temp2].parent!=0) {
                    t2+=crime[temp2].weight;temp2=crime[temp2].parent;
                }     // 查找 two 所在树的头节点，并计算到头节点的权之和
                if(temp1!=temp2)          // 不在同一棵树
```

```
                printf("Not sure yet.\n");
            else if((t2-t1)%2==0)        // 权相差为偶数
                printf("In the same gang.\n");
            else                                // 权相差为一个奇数
                printf("In different gangs.\n");
        }
    }
}
    return 0;
}
```

Pku1182--食物链

Description

动物王国中有三类动物 A,B,C，这三类动物的食物链构成了有趣的环形。A 吃 B，B 吃 C，C 吃 A。
现有 N 个动物，以 1－N 编号。每个动物都是 A,B,C 中的一种，但是我们并不知道它到底是哪一种。
有人用两种说法对这 N 个动物所构成的食物链关系进行描述：
第一种说法是"1 X Y"，表示 X 和 Y 是同类。
第二种说法是"2 X Y"，表示 X 吃 Y。
此人对 N 个动物，用上述两种说法，一句接一句地说出 K 句话，这 K 句话有的是真的，有的是假的。当一句话满足下列三条之一时，这句话就是假话，否则就是真话。
1） 当前的话与前面的某些真的话冲突，就是假话；
2） 当前的话中 X 或 Y 比 N 大，就是假话；
3） 当前的话表示 X 吃 X，就是假话。
你的任务是根据给定的 N（1 <= N <= 50,000）和 K 句话（0 <= K <= 100,000），输出假话的总数。

Input

第一行是两个整数 N 和 K，以一个空格分隔。
以下 K 行每行是三个正整数 D，X，Y，两数之间用一个空格隔开，其中 D 表示说法的种类。
若 D=1，则表示 X 和 Y 是同类。
若 D=2，则表示 X 吃 Y。

Output

只有一个整数，表示假话的数目。

Sample Input

```
100 7
1 101 1
2 1 2
2 2 3
2 3 3
1 1 3
2 3 1
1 5 5
```

Sample Output

```
3
```

```
#include <iostream>
#include <cstdio>
using namespace std;
const int N = 50005;
int r[N], pre[N];
int n, k;
int ans = 0;
void init(void) {
	for (int i = 1; i <= n; i++) {
		r[i] = 0;
		pre[i] = i;
	}
	return;
}
int Find(int xx) {
	if (pre[xx] == xx) {
		return xx;
	}
	int t = pre[xx];
	pre[xx] = Find(pre[xx]);
	r[xx] = (r[xx] + r[t]) %3;
	return pre[xx];
}
void Union(int xx, int yy, int dd) {
	int a = Find(xx), b = Find(yy);
	pre[a] = b;
	r[a] = (r[yy] - r[xx] + 3 + dd) %3;
```

```
        return;
}
int main(void) {
        int d, x, y;
        cin >> n >> k;
        init();
        for (int i = 1; i <= k; i++) {
                scanf("%d%d%d", &d, &x, &y);
                if (x > n || y > n) {
                        ans++;
                        continue;
                }
                if (d == 1) {
                        if (Find(x) == Find(y)) {
                                if (r[x] != r[y])
                                        ans++;
                        } else
                                Union(x, y, 0);
                } else {
                        if (Find(x) == Find(y)) {
                                if (r[x] != (r[y]+1)%3)
                                        ans++;
                        } else
                                Union(x, y, 1);
                }
        }
        cout << ans << endl;
        return 0;
}
```

Pku1988--Cube Stacking

Description

Farmer John and Betsy are playing a game with N (1 <= N <= 30,000)identical cubes labeled 1 through N. They start with N stacks, each containing a single cube. Farmer John asks Betsy to perform P (1<= P <= 100,000) operation. There are two types of operations:
moves and counts.
* In a move operation, Farmer John asks Bessie to move the stack containing cube X on top of the stack containing cube Y.
* In a count operation, Farmer John asks Bessie to count the number of cubes on the stack with cube X that are under the cube X and report that value.

Write a program that can verify the results of the game.

Input

* Line 1: A single integer, P
* Lines 2..P+1: Each of these lines describes a legal operation. Line 2 describes the first operation, etc. Each line begins with a 'M' for a move operation or a 'C' for a count operation. For move operations, the line also contains two integers: X and Y.For count operations, the line also contains a single integer: X.

Note that the value for N does not appear in the input file. No move operation will request a move a stack onto itself.

Output

Print the output from each of the count operations in the same order as the input file.

Sample Input

```
6
M 1 6
C 1
M 2 4
M 2 6
C 3
C 4
```

Sample Output

```
1
0
2
```

```
#include <stdio.h>
#define size 30005
int pre[size], num[size], behind[size];
int n;
void make_set(int x) {
	pre[x] = -1;
	num[x] = 1;
	behind[x] = 0;
	return ;
```

```
}
int find_set(int x) {
	int r, j, q;
	r = x;
	while (pre[r] != -1)
		r = pre[r];
	while (x != r) {
		q = pre[x];
		pre[x] = r;
		j = q;
		do {
			behind[x] += behind[j];
			j = pre[j];
		} while (j != -1);
		x = q;
	}
	return r;
}
void union_set(int a, int b) {
	int t1, t2;
	t1 = find_set(a);
	t2 = find_set(b);
	pre[t2] = t1;
	behind[t2] = num[t1];
	num[t1] += num[t2];
	return ;
}
int main(void) {
	int n, i, x, y, t;
	char c;
	scanf ("%d", &n);
	for (i = 1; i < size; i++)
		make_set(i);
	while (n--) {
		getchar();
		scanf ("%c", &c);
		if (c == 'M') {
			scanf ("%d%d", &x, &y);
			union_set(x, y);
		} else if (c == 'C'){
			scanf ("%d", &x);
			t = find_set(x);
			printf ("%d\n", num[t]-behind[x]-1);
		}
```

```
    }
}
```

Pku1733--**Parity game**

Description

Now and then you play the following game with your friend. Your friend writes down a sequence consisting of zeroes and ones. You choose a continuous subsequence (for example the subsequence from the third to the fifth digit inclusively) and ask him, whether this subsequence contains even or odd number of ones. Your friend answers your question and you can ask him about another subsequence and so on. Your task is to guess the entire sequence of numbers.

You suspect some of your friend's answers may not be correct and you want to convict him of falsehood. Thus you have decided to write a program to help you in this matter. The program will receive a series of your questions together with the answers you have received from your friend. The aim of this program is to find the first answer which is provably wrong, i.e. that there exists a sequence satisfying answers to all the previous questions, but no such sequence satisfies this answer.

Input

The first line of input contains one number, which is the length of the sequence of zeroes and ones. This length is less or equal to 1000000000. In the second line, there is one positive integer which is the number of questions asked and answers to them. The number of questions and answers is less or equal to 5000. The remaining lines specify questions and answers. Each line contains one question and the answer to this question: two integers (the position of the first and last digit in the chosen subsequence) and one word which is either `even' or `odd' (the answer, i.e. the parity of the number of ones in the chosen subsequence, where `even' means an even number of ones and `odd' means an odd number).

Output

There is only one line in output containing one integer X. Number X says that there exists a sequence of zeroes and ones satisfying first X parity conditions, but there exists none satisfying X+1 conditions. If there exists a sequence of zeroes and ones satisfying all the given conditions, then number X should be the number of all the questions asked.

Sample Input

```
10
5
1 2 even
3 4 odd
5 6 even
1 6 even
7 10 odd
```

Sample Output

```
3
```

```
#include <iostream>
#include <map>
using namespace std;
const int MAX = 10005;
int n, p;
int pre[MAX];
int parity[MAX];              //i 到目前集合的根的奇偶情况
map<int, int> numIndex;              //用于离散化
int Find (int x){
    if ( pre[x] == -1 )
        return x;
    int f;
    f = Find(pre[x]);
    parity[x] = (parity[x] + parity[pre[x]]) % 2;
    //此时 pre[x]已指向最终的集合的根
    pre[x] = f;
    return f;
}
bool Query (int x, int y, int odd){
    int r1, r2;
    r1 = Find(x);
    r2 = Find(y);
    if ( r1 == r2 ) {
        if ( (parity[x] + parity[y]) % 2 == odd )
            return true;
        else
            return false;
    }
    else                    //只是将 r1 接到 r2 下面，这边还可以优化
    {
```

```
            pre[r1] = r2;
            parity[r1] = (parity[x] + parity[y] + odd) % 2;
            return true;
        }
}
void Solve (){
        int i, x, y, index, idx1, idx2, odd;
        char s[10];
        scanf("%d%d", &n, &p);
        index = 0;
        for (i=0; i<p; i++){
            scanf("%d%d%s", &x, &y, &s);
            x --;
            if ( numIndex.find(x) == numIndex.end() )
                numIndex[x] = index ++;
            idx1 = numIndex[x];
            if ( numIndex.find(y) == numIndex.end() )
                numIndex[y] = index ++;
            idx2 = numIndex[y];
            if ( strcmp(s, "odd") == 0 )
                odd = 1;
            else
                odd = 0;
            if ( Query(idx1, idx2, odd) == false )      {
                break;
            }
        }
        printf("%d\n", i);
}
void Init (){
        memset(pre, -1, sizeof(pre));
}
int main (){
        Init();
        Solve();
        return 0;
}
```

Pku2492--A Bug's Life

Description

Background
Professor Hopper is researching the sexual behavior of a rare species of bugs. He assumes that they feature two different genders and that they only interact with bugs of the opposite gender. In his experiment, individual bugs and their interactions were easy to identify, because numbers were printed on their backs.
Problem
Given a list of bug interactions, decide whether the experiment supports his assumption of two genders with no homosexual bugs or if it contains some bug interactions that falsify it.

Input

The first line of the input contains the number of scenarios. Each scenario starts with one line giving the number of bugs (at least one, and up to 2000) and the number of interactions (up to 1000000) separated by a single space. In the following lines, each interaction is given in the form of two distinct bug numbers separated by a single space. Bugs are numbered consecutively starting from one.

Output

The output for every scenario is a line containing "Scenario #i:", where i is the number of the scenario starting at 1, followed by one line saying either "No suspicious bugs found!" if the experiment is consistent with his assumption about the bugs' sexual behavior, or "Suspicious bugs found!" if Professor Hopper's assumption is definitely wrong.

Sample Input

```
2
3 3
1 2
2 3
1 3
4 2
1 2
3 4
```

Sample Output

```
Scenario #1:
Suspicious bugs found!
```

```
Scenario #2:
No suspicious bugs found!
```

```
#include <iostream>
#include <cstdio>
using namespace std;
int ncas, arr[2005], brr[2005];
int m, n;
void init(int x) {
	arr[x] = x;
	brr[x] = -1;
	return;
}
int find(int x) {
	int y;
	if (arr[x] == x)
		return x;
	else {
		y = find(arr[x]);
		arr[x] = y;
	}
	return y;
}
void Union(int xx, int yy) {
	if (brr[yy] == -1) {
		brr[yy] = xx;
	} else {
		arr[brr[yy]] = xx;
		brr[yy] = xx;
	}
	if (brr[xx] == -1) {
		brr[xx] = yy;
	} else {
		arr[brr[xx]] = yy;
		brr[xx] = yy;
	}
	return ;
}
void opt(void) {
	int aa, bb, flag;
	int tmp1, tmp2;
	scanf("%d%d", &m, &n);
	for (int i = 1; i <= m; i++)
		init(i);
```

```
        flag = 1;
        for (int i = 1; i <= n; i++) {
            if (!flag) {
                scanf("%d%d", &aa, &bb);
                continue;
            }
            scanf("%d%d", &aa, &bb);
            tmp1 = find(aa);
            tmp2 = find(bb);
            if (tmp1 == tmp2) {
                flag = 0;
                continue;
            } else {
                Union(tmp1, tmp2);
            }
        }
        if (flag)
            cout << "No suspicious bugs found!\n" << endl;
        else
            cout << "Suspicious bugs found!\n" << endl;
        return ;
}
int main(void) {
        cin >> ncas;
        for (int t = 1; t <= ncas; t++) {
            printf("Scenario #%d:\n", t);
            opt();
        }
        return 0;
}
```

Pku2236--**Wireless Network**

Description

An earthquake takes place in Southeast Asia. The ACM (Asia Cooperated Medical team) have set up a wireless network with the lap computers, but an unexpected aftershock attacked, all computers in the network were all broken. The computers are repaired one by one, and the network gradually began to work again. Because of the hardware restricts, each computer can only directly communicate with the computers that are not farther than d meters from it. But every computer can be regarded as the intermediary of the communication between two other computers, that is to say computer A and computer B can communicate if computer A and computer B can

communicate directly or there is a computer C that can communicate with both A and B.
In the process of repairing the network, workers can take two kinds of operations at every moment, repairing a computer, or testing if two computers can communicate. Your job is to answer all the testing operations.

Input

The first line contains two integers N and d (1 <= N <= 1001, 0 <= d <= 20000). Here N is the number of computers, which are numbered from 1 to N, and D is the maximum distance two computers can communicate directly. In the next N lines, each contains two integers xi, yi (0 <= xi, yi <= 10000), which is the coordinate of N computers. From the (N+1)-th line to the end of input, there are operations, which are carried out one by one. Each line contains an operation in one of following two formats:
1. "O p" (1 <= p <= N), which means repairing computer p.
2. "S p q" (1 <= p, q <= N), which means testing whether computer p and q can communicate.
The input will not exceed 300000 lines.

Output

For each Testing operation, print "SUCCESS" if the two computers can communicate, or "FAIL" if not.

Sample Input

```
4 1
0 1
0 2
0 3
0 4
O 1
O 2
O 4
S 1 4
O 3
S 1 4
```

Sample Output

FAIL
SUCCESS

```
#include <stdio.h>
#define MAX 1006
#define EC(a) ((a)*(a))
int rank[MAX];
int pre[MAX];
int n, d;
struct node {
	int x;
	int y;
	int f;
};
struct node dik[MAX];

void make_set(int a) {
	int x, y;
	scanf ("%d%d", &x, &y);
	dik[a].x = x;
	dik[a].y = y;
	dik[a].f = 0;
	rank[a] = 1;
	pre[a] = a;
	return ;
}
int find_set(int a) {
	if (a != pre[a])
		pre[a] = find_set(pre[a]);
	return pre[a];
}

void union_set(int a, int b) {
	int t1, t2;
	t1 = find_set(a);
	t2 = find_set(b);
	if (rank[t1] > rank[t2])
		pre[t2] = t1;
	else
		pre[t1] = t2;
	if (rank[t1] == rank[t2])
		rank[t2]++;
	return ;
}
void opt(int a) {
```

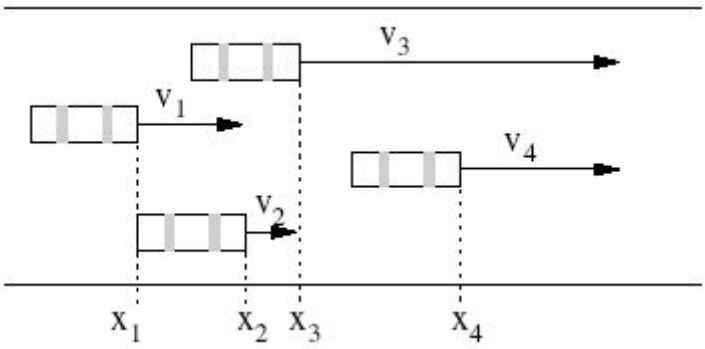
```
    int i, t1, t2;
    t1 = find_set(a);
    for (i = 1; i <= n; i++) {
        if (dik[i].f && i != a) {
            if ((EC(dik[i].x - dik[a].x)+
            EC(dik[i].y - dik[a].y)) <= d) {
                t2 = find_set(i);
                if (t1 != t2)
                    union_set(t1, t2);
            }
        }
    }
    return ;
}
int main(void) {
    int i, a, b;
    char arr[100];
    scanf ("%d%d", &n, &d);
    d *= d;
    for (i = 1; i <= n; i++) {
        make_set(i);
    }
    while (scanf ("%s", arr) != EOF) {
        if (arr[0] == 'O') {
            scanf ("%d", &a);
            dik[a].f = 1;
            opt(a);
        } else {
            scanf ("%d%d", &a, &b);
            if (find_set(a) == find_set(b))
                printf ("SUCCESS\n");
            else
                puts("FAIL");
        }
    }
    return 0;
}
```

1.4.2 堆及其变种

Pku2274--**The Race**

Description

During the Annual Interstellar Competition for Tuned Spaceships, N spaceships will be competing. Each spaceship i is tuned in such a way that it can accelerate in zero time to its maximum speed Vi and remain cruising at that speed. Due to past achievements, each spaceship starts at a starting position Xi, specifying how many kilometers the spaceship is away from the starting line.
The race course is infinitely long. Because of the high speeds of the spaceships, the race course goes straight all the time. On that straight course, spaceships can pass one another very easily, without interfering with each other.
Many people in the audience have not realized yet that the outcome of the race can be determined in advance. It is your task to show this to them, by telling them how many times spaceships will pass one another, and by predicting the first 10 000 times that spaceships pass in chronological order.
You may assume that each spaceship starts at a different position. Furthermore, there will never be more than two spaceships at the same position of the course at any time.



Input

The first line of the input specifies the number of spaceshipsN (0 < N <= 250 000) that are competing. Each of the next N lines describe the properties of one spaceship. The i+1th line describes the ith ship with two integers Xi and Vi, representing the starting position and the velocity of the ith spaceship (0 <= Xi <= 1 000 000, 0 < Vi < 100). The spaceships are ordered according to the starting position, i.e. X1 < X2 < . . . < XN. The starting position is the number of kilometers past the starting line where the spaceship starts, and the velocity is given in kilometers per second.

Output

The first line of the output should contain the number of times that spaceships pass one another during the race modulo 1 000 000. By publishing the number of passes only modulo 1 000 000, you can at the same time prove your knowledge of it and don't spoil the party for the less intelligent people in the audience.
Each of the subsequent lines should represent one passing, in chronological order. If there would be more than 10 000 passings, only output the first 10 000 passings. If there are less than 10 000 passings, output all passings. Each line should consist of two integers i and j, specifying that spaceship i passes spaceship j. If multiple passings occur at the same time, they have to be sorted by their position on the course. This means that passings taking place closer to the starting line must be listed first. The time of a passing is the time when the two spaceships are at the same position.

Sample Input

```
4
0 2
2 1
3 8
6 3
```

Sample Output

```
2
3 4
1 2
```

Pku1197--Depot

Description

A Finnish high technology company has a big rectangular depot. The depot has a worker and a manager. The sides of the depot, in the order around it, are called left, top, right and bottom. The depot area is divided into equal-sized squares by dividing the area into rows and columns. The rows are numbered starting from the top with integers 1,2,... and the columns are numbered starting from the left with integers 1,2,...

The depot has containers, which are used to store invaluable technological devices. The containers have distinct identification numbers. Each container occupies one

square. The depot is so big, that the number of containers ever to arrive is smaller than the number of rows and smaller than the number of columns. The containers are not removed from the depot, but sometimes a new container arrives. The entry to the depot is at the top left corner.
The worker has arranged the containers around the top left corner of the depot in such a way that he will be able to find them by their identification numbers. He uses the following method.
Suppose that the identification number of the next container to be inserted is k (container k, for short). The worker travels the first row starting from the left and looks for the first container with identification number larger than k. If no such container is found, then container k is placed immediately after the rightmost of the containers previously in the row. If such a container l is found, then container l is replaced by container k, and l is inserted to the following row using the same method. If the worker reaches a row having no containers, the container is placed in the leftmost square of that row.
Suppose that containers 3,4,9,2,5,1 have arrived to the depot in this order. Then the placement of the containers at the depot is as follows.
1 4 5
2 9
3
The manager comes to the worker and they have the following dialogue:
Manager: Did container 5 arrive before container 4?
Worker: No, that is impossible.
Manager: Oh, so you can tell the arrival order of the containers by their placement.
Worker: Generally not. For instance, the containers now in the depot could have arrived in the order 3,2,1,4,9,5 or in the order 3,2,1,9,4,5 or in one of 14 other orders.

As the manager does not want to show that the worker seems much smarter, he goes away. You are to help the manager and write a program which, given a container placement, counts all possible orders in which they might have arrived.

Input

Your program is to read from standard input. The first line contains one integer R: the number of rows with containers in them. The following R lines contain information about rows 1,...,R starting from the top as follows. First on each of those lines is an integer M: the number of containers in that row. Following that, there are M integers on the line: the identification numbers of the containers in the row starting from the left. All container identification numbers I satisfy 1 <= I <= 50. Let N be the number of containers in the depot, then 1 <= N <= 13.

Output

Your program is to write to standard output. The output contains an integer: the number of possible orders in which containers might have arrived. An arrival order should be counted only once.

Sample Input

```
3
3 1 4 5
2 2 9
1 3
```

Sample Output

```
16
#include<iostream>
#define  MAXNUM  13
using namespace std;
int B1[MAXNUM+1][MAXNUM+1], B2[MAXNUM+1][MAXNUM+1];
int A[MAXNUM], B[MAXNUM];
int count=0;
int Equal(int B1[][MAXNUM+1], int B2[][MAXNUM+1],int n){
        int i,j;
        for(i=1; i<=n; i++){
                for(j=1; j<=n; j++){
                        if(B1[i][j]!=B2[i][j])
                                return 0;
                }
        }
        return 1;
}
void put(int n, int r,int N){//在第 r 行放置 n;
        int c, tN;//列号; 辅助变量
        for(c=1; c<=N; c++){
                if(n<B2[r][c]){//找到第一个比 n 大的编号;
                        tN=B2[r][c];
                        B2[r][c]=n;
                        put(tN, r+1, N);
                        return;
                }
                else if(B2[r][c]==0){
                        if(c==1){//新增一行
                                B2[r][c]=n;
                                B2[r][0]++;//每行箱子数加 1
```

```
                                B2[0][0]++;//总行数加 1
                        }else{//行末尾加入
                                B2[r][c]=n;
                                B2[r][0]++;
                        }
                        return;
                }
        }
}
int bein(int e, int a[], int n){//判断元素 e 在不在数组 a 中：1 在，0 不在
        int i;
        if(n==0) return 0;
        else{
                for(i=0; i<n; i++){
                        if(e==a[i]) return 1;
                }
                return 0;
        }
}
int All_orders(int co, int n){
        int i, j, k;
        int B3[MAXNUM+1][MAXNUM+1];
        for(j=0; j<=n; j++){
                for(k=0; k<=n; k++)
                        B3[j][k]=B2[j][k];//B3 用来保留 B2 的当前状态
        }
        if(co>=n && Equal(B1,B2,n)/**/)
                count++;
        else{
                for(i=0; i<n; i++){
                        if( !bein(A[i],B,co) ){//如果 A[i]不在数组 B 中
                                put(A[i],1,n);//将 A[i]放入仓库 B2 中，B2 状态
改变
                                int flag=0;
                                for(j=0; j<=B2[0][0]; j++){
                                        if(B2[j][0]>B1[j][0]){
                                                flag=1; break;
                                        }
                                }
                                if(flag==0 ){
                                        B[co]=A[i];//将 A[i]加到数组 B 的最后
                                        All_orders(co+1, n);//递归调用
                                        for(j=0; j<=n; j++)
                                                for(k=0; k<=n; k++)
```

```
                                                B2[j][k]=B3[j][k];
                                        }else{//恢复仓库 B2 的状态
                                                for(j=0; j<=n; j++)
                                                        for(k=0; k<=n; k++)
                                                                B2[j][k]=B3[j][k];
                                        }
                                }//if
                        }//for
                }
                return count;
}
int main(){
                int i,j,R,M, count_A=0;
                cin>>R;  B1[0][0]=R;
                for(i=1; i<=R; i++){
                        cin>>M;  B1[i][0]=M;
                        for(j=1; j<=M; j++){
                                cin>>B1[i][j];
                                A[count_A++]=B1[i][j];
                        }
                }
                cout<<All_orders(0,count_A)<<endl;
                return 0;
}
```

1.4.2 字典树的两种实现方式：哈希表,二叉搜索树

Pku2503--**Babelfish**

Description

You have just moved from Waterloo to a big city. The people here speak an incomprehensible dialect of a foreign language. Fortunately, you have a dictionary to help you understand them.

Input

Input consists of up to 100,000 dictionary entries, followed by a blank line, followed by a message of up to 100,000 words. Each dictionary entry is a line containing an English word, followed by a space and a foreign language word. No foreign word appears more than once in the dictionary. The message is a sequence of words in the

foreign language, one word on each line. Each word in the input is a sequence of at most 10 lowercase letters.

Output

Output is the message translated to English, one word per line. Foreign words not in the dictionary should be translated as "eh".

Sample Input

```
dog ogday
cat atcay
pig igpay
froot ootfray
loops oopslay
atcay
ittenkay
oopslay
```

Sample Output

```
cat
eh
loops
```

```
#include<stdio.h>
#include<string.h>
#include<math.h>
#define MOD 300005
struct abc
{
    bool boo;
    char akey[12];
    char bkey[12];
}hash[300005];

int ELFhash(char *key)
{
    unsigned long h=0;
    while(*key)
    {
        h=(h<<4)+*key++;
        unsigned long g=h&0Xf0000000L;
```

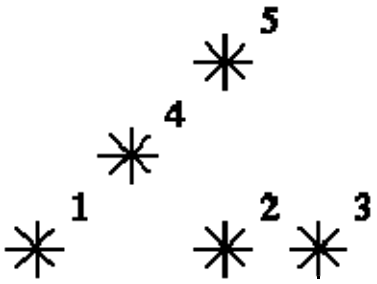
```
            if(g) h^=g>>24;
            h&=~g;
        }
        return h%MOD;
}
void insert(char a[],char b[],int w)
{
        if(!hash[w].boo)
        {
            hash[w].boo=true;
            memcpy(hash[w].akey,a,sizeof(hash[w].akey));
            memcpy(hash[w].bkey,b,sizeof(hash[w].bkey));
        }
        else
            insert(a,b,w+1);
}
int find(char b[],int w)
{
        if(hash[w].boo && strcmp(hash[w].bkey,b)==0)
        {
            printf("%s\n",hash[w].akey);
                return 1;
        }
        else
        {
            if(!hash[w].boo)
                return 0;
            else
                return find(b,w+1);
        }
}
int main()
{
        char a[12],b[12];
        scanf("%c",&a[0]);
        while(scanf("%s%s",a+1,b))
        {
            getchar();
            int w=ELFhash(b);
            insert(a,b,w);
            scanf("%c",&a[0]);
            if(a[0]=='\n')
                break;
        }
```

```
    while(scanf("%s",&b)==1)
    {
        int w=ELFhash(b);
        if(find(b,w)==0)
        printf("eh\n");
    }
    return 0;
}
```

Pku2352--**Stars**

Description

Astronomers often examine star maps where stars are represented by points on a plane and each star has Cartesian coordinates. Let the level of a star be an amount of the stars that are not higher and not to the right of the given star. Astronomers want to know the distribution of the levels of the stars.



For example, look at the map shown on the figure above. Level of the star number 5 is equal to 3 (it's formed by three stars with a numbers 1, 2 and 4). And the levels of the stars numbered by 2 and 4 are 1. At this map there are only one star of the level 0, two stars of the level 1, one star of the level 2, and one star of the level 3.

You are to write a program that will count the amounts of the stars of each level on a given map.

Input

The first line of the input file contains a number of stars N (1<=N<=15000). The following N lines describe coordinates of stars (two integers X and Y per line separated by a space, 0<=X,Y<=32000). There can be only one star at one point of the plane. Stars are listed in ascending order of Y coordinate. Stars with equal Y coordinates are listed in ascending order of X coordinate.

Output

The output should contain N lines, one number per line. The first line contains amount of stars of the level 0, the second does amount of stars of the level 1 and so on, the last line contains amount of stars of the level N-1.

Sample Input

```
5
1 1
5 1
7 1
3 3
5 5
```

Sample Output

```
1
2
1
1
0
```

```
#include <iostream>
#include <cstring>
using namespace std;
const int maxn = 32005;
int arr[maxn] = {0};
int brr[maxn] = {0};
inline int lowbit(int n) {
	return n&(n^(n-1));
}
void change(int n) {
	while (n < maxn) {
		brr[n] += 1;
		n += lowbit(n);
	}
}
int sum(int n) {
	int p = 0;
	while (n > 0) {
		p += brr[n];
		n -= lowbit(n);
	}
	return p;
}
```

```
int main(void) {
	int n, x, y;
	cin >> n;
	for (int i = 0; i < n; i++) {
		cin >> x >> y;
		x++;
		change(x);
		arr[sum(x)-1]++;
	}
	for (int i = 0; i < n; i++)
		cout << arr[i] << endl;
	return 0;
}
```

1.4.4 两个特殊的结构：线段树和Trie

Pku1177--**Picture**

Description

A number of rectangular posters, photographs and other pictures of the same shape are pasted on a wall. Their sides are all vertical or horizontal. Each rectangle can be partially or totally covered by the others. The length of the boundary of the union of all rectangles is called the perimeter.

Write a program to calculate the perimeter. An example with 7 rectangles is shown in Figure 1.

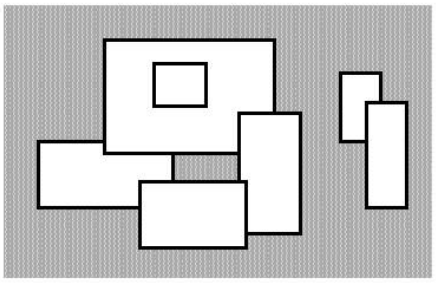


Figure 1. A set of 7 rectangles

The corresponding boundary is the whole set of line segments drawn in Figure 2.

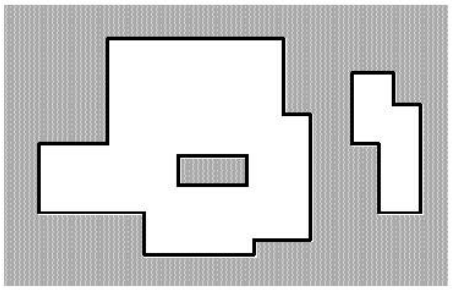


Figure 2. The boundary of the set of rectangles

The vertices of all rectangles have integer coordinates.

Input

Your program is to read from standard input. The first line contains the number of rectangles pasted on the wall. In each of the subsequent lines, one can find the integer coordinates of the lower left vertex and the upper right vertex of each rectangle. The values of those coordinates are given as ordered pairs consisting of an x-coordinate followed by a y-coordinate.
0 <= number of rectangles < 5000
All coordinates are in the range [-10000,10000] and any existing rectangle has a positive area.

Output

Your program is to write to standard output. The output must contain a single line with a non-negative integer which corresponds to the perimeter for the input rectangles.

Sample Input

```
7
-15 0 5 10
-5 8 20 25
15 -4 24 14
0 -6 16 4
2 15 10 22
30 10 36 20
34 0 40 16
```

Sample Output

228

```
#include<iostream>
#include<algorithm>
const int MAX=10010;
using namespace std;
struct NODE{
        int left,right;
        NODE *lchild,* rchild;
        int len,cover;
        int m,line;
        bool lcover,rcover;
        NODE(){lchild=rchild=NULL;lcover=rcover=false;m=line=0;}
}root;
struct LINE{
        int top,end;
        int x;
        bool tag;
}line[MAX+1];
int temp[MAX+1];
int index[MAX+1],len;
void getm(NODE* node);
void getline(NODE* node);
void build(NODE* node,int l,int r)
{
    node->right=r,node->left=l;
    node->len=index[r]-index[l];
    node->cover=0;
    if(node->right-node->left>1)
    {
        node->lchild=new NODE;
        node->rchild=new NODE;
        build(node->lchild,l,(l+r)/2);
        build(node->rchild,(l+r)/2,r);
    }
    return;
}
void insert(NODE* node,int l,int r)
{
    if(l<=node->left&&r>=node->right)
    {
        node->cover++;
```

```
        }
        else
        {
            int mid=(node->left+node->right)/2;
            if(l<mid)insert(node->lchild,l,r);
            if(r>mid)insert(node->rchild,l,r);
        }
        getm(node);
        getline(node);
        return;
}
void del(NODE* node,int l,int r)
{
        if(l<=node->left&&r>=node->right)
        {
                    node->cover--;
        }
        else
        {
            int mid=(node->left+node->right)/2;
            if(l<mid)del(node->lchild,l,r);
            if(r>mid)del(node->rchild,l,r);
        }
        getm(node);
        getline(node);
        return;
}
void getm(NODE* node)
{
        if(node->cover>0)
        {
            node->m=node->len;
        }
        else
        {
            if(node->right-node->left>1)
            {
                node->m=node->lchild->m+node->rchild->m;
            }
            else
            {
                node->m=0;
            }
        }
```

```
        return;
}
void getline(NODE* node)
{
    if(node->cover>0)
    {
        node->lcover=node->rcover=true;
        node->line=1;
    }
    else if(node->right-node->left>1)
    {
        node->lcover=node->lchild->lcover;
        node->rcover=node->rchild->rcover;

node->line=node->lchild->line+node->rchild->line-node->rchild->lcover*node->lchild->rcover;
    }
    else
    {
        node->lcover=node->rcover=false;
        node->line=0;
    }
    return;
}
int getindex(int y)
{
    int i;
    for(i=0;i<len;++i)
        if(y==index[i])return i;
}
bool cmp(LINE a,LINE b)
{
    return a.x<b.x;
}
int main()
{
    int n,i,j;
    int x1,y1,x2,y2;
    int ans=0,m=0;
    scanf("%d",&n);
    for(i=0;i<n;++i)
    {
        scanf("%d%d%d%d",&x1,&y1,&x2,&y2);
        line[2*i].x=x1,line[2*i+1].x=x2;
        line[2*i].top=line[2*i+1].top=y1;
```

```
            line[2*i].end=line[2*i+1].end=y2;
            line[2*i].tag=true,line[2*i+1].tag=false;
            temp[2*i]=y1,temp[2*i+1]=y2;
        }
        sort(temp,temp+2*n);
        sort(line,line+2*n,cmp);
        index[0]=temp[0];
        for(i=1,len=1;i<2*n;++i)
        {
            if(temp[i]!=temp[i-1])
            index[len++]=temp[i];
        }
        build(&root,0,len-1);
            for(i=0;i<2*n-1;++i)
        {
            if(line[i].tag==true)
            {
                                insert(&root,getindex(line[i].top),getindex(line[i].end));
            }
            else
            {
                                del(&root,getindex(line[i].top),getindex(line[i].end));
            }
            ans+=root.line*(line[i+1].x-line[i].x)*2;
            ans+=abs(root.m-m);
            m=root.m;
        }
        del(&root,getindex(line[i].top),getindex(line[i].end));
        ans+=abs(root.m-m);
        printf("%d\n",ans);

        return 0;
}
```

1.5 动态规划

1.5.1 动态规划的两种动机

Pku1141--Brackets Sequence

Description

Let us define a regular brackets sequence in the following way:

1. Empty sequence is a regular sequence.
2. If S is a regular sequence, then (S) and [S] are both regular sequences.
3. If A and B are regular sequences, then AB is a regular sequence.

For example, all of the following sequences of characters are regular brackets sequences:
(), [], (()), ([]), ()[], ()[()]
And all of the following character sequences are not:
(, [,),)(, ([)], ([(]
Some sequence of characters '(', ')', '[', and ']' is given. You are to find the shortest possible regular brackets sequence, that contains the given character sequence as a subsequence. Here, a string a1 a2 ... an is called a subsequence of the string b1 b2 ... bm, if there exist such indices 1 = i1 < i2 < ... < in = m, that aj = bij for all 1 = j = n.

Input

The input file contains at most 100 brackets (characters '(', ')', '[' and ']') that are situated on a single line without any other characters among them.

Output

Write to the output file a single line that contains some regular brackets sequence that has the minimal possible length and contains the given sequence as a subsequence.

Sample Input

```
([(]
```

Sample Output

```
()[()]
```

```
#include <iostream>
#include <string>
using namespace std;
const int N=2500;
int a[N][N], f[N][N];
string arr;
void find(int xx, int yy) {
	if (xx > yy)
		return;
```

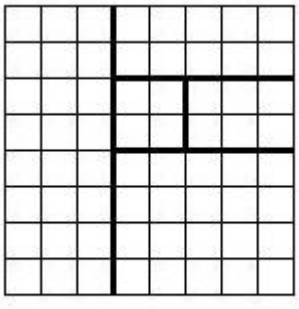
```
    if (f[xx][yy] == -1) {
        if (xx == yy) {
            if (arr[xx] == '(' || arr[xx] == ')')
                cout << "()";
            else if (arr[xx] == '[' || arr[xx] == ']')
                cout << "[]";
        } else {
            cout << arr[xx];
            find(xx+1, yy-1);
            cout << arr[yy];
        }
    } else {
        int tmp = f[xx][yy];
        find(xx, tmp);
        find(tmp+1, yy);
    }
    return ;
}
int main(void) {
    int len, i, j, k;
    cin >> arr;
    len = arr.length();
    memset(a, 0, sizeof(a));
    memset(f, -1, sizeof(f));
    for (k = 0; k < len; k++) {
    for (i=0,j=k; j < len; i++, j++) {
        if (i == j) {
            a[i][j] = 1;
        } else {
            int tmp = 10000000;
            if (arr[i] == '(' && arr[j] == ')')
                tmp = min(tmp, a[i+1][j-1]);
            else if (arr[i] == '[' && arr[j] ==']')
                tmp = min(tmp, a[i+1][j-1]);
            for (int m = i; m < j; m++) {
                if (tmp > a[i][m] + a[m+1][j]) {
                    tmp = a[i][m] + a[m+1][j];
                    f[i][j] = m;
                }
            }
            a[i][j] = tmp;
        }
    }
    }
```

```
    find(0, len-1);
    cout << endl;
    return 0;
}
```

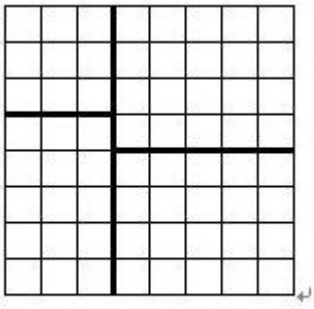
Pku1191--棋盘分割

Description

将一个 8 * 8 的棋盘进行如下分割：将原棋盘割下一块矩形棋盘并使剩下部分也是矩形，再将剩下的部分继续如此分割，这样割了(n-1)次后，连同最后剩下的矩形棋盘共有 n 块矩形棋盘。(每次切割都只能沿着棋盘格子的边进行)



允许的分割方案



不允许的分割方案

原棋盘上每一格有一个分值，一块矩形棋盘的总分为其所含各格分值之和。现在需要把棋盘按上述规则分割成 n 块矩形棋盘，并使各矩形棋盘总分的均方差最小。

均方差 $\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}}$，其中平均值 $\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$，$x_i$ 为第 i 块矩形棋盘的总分。

请编程对给出的棋盘及 n，求出 O'的最小值。

Input

第 1 行为一个整数 n(1 < n < 15)。

第 2 行至第 9 行每行为 8 个小于 100 的非负整数，表示棋盘上相应格子的分值。每行相邻两数之间用一个空格分隔。

Output

仅一个数，为 O'（四舍五入精确到小数点后三位）。

Sample Input

```
3
1 1 1 1 1 1 1 3
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 0
1 1 1 1 1 1 0 3
```

Sample Output

```
1.633
```

```
#include   <stdio.h>
#include <math.h>
const     int   INF   =     2000000000 ;
int   f[ 9 ][ 9 ][ 9 ][ 9 ][ 15 ];
int   s[ 9 ][ 9 ][ 9 ][ 9 ];
int   d[ 9 ][ 9 ];
void   init()
  {
      int   x1, y1, x2, y2;
      int   i, j;
      int   sum;
      for   (x1 = 1 ; x1 <= 8 ; x1 ++ )
          for   (y1 = 1 ; y1 <= 8 ; y1 ++ )
              for   (x2 = x1; x2 <= 8 ; x2 ++ )
                  for   (y2 = y1; y2 <= 8 ; y2 ++ )
                    {
                      sum   =     0 ;
                       for   (i = x1; i <= x2; i ++ )
                           for   (j = y1; j <= y2; j ++ )
                               sum   +=   d[i][j];
                      s[x1][y1][x2][y2]   = sum;
                      f[x1][y1][x2][y2][ 1 ]   =   sum   *   sum;
                    }
}
int   main()
  {
      int   n;
      int   i, j, k;
```

```
    int   x1, y1, x2, y2;
    int   a, b;
    int   t, tmp;
    double   p   =    0 ;
  scanf( "%d" ,   &n);
    for   (i = 1 ; i <= 8 ; i ++ )
        for   (j = 1 ; j <= 8 ; j ++ )
         {
           scanf( "%d" ,   & d[i][j]);
           p   +=   d[i][j];
       }
  p   /=   n;
  init();
    for   (k = 2 ; k <= n; k ++ )
     {
        for   (x1 = 1 ; x1 <= 8 ; x1 ++ )
            for   (y1 = 1 ; y1 <= 8 ; y1 ++ )
                for   (x2 = x1; x2 <= 8 ; x2 ++ )
                    for   (y2 = y1; y2 <= 8 ; y2 ++ )
                     {
                       tmp   =   INF;
                        // 竖切
                         for   (a = x1; a < x2; a ++ )
                         {
                           t   =   f[x1][y1][a][y2][k - 1 ] + s[a + 1 ][y1][x2][y2] * s[a +
1 ][y1][x2][y2];
                            if   (tmp   >   t)
                               tmp   =   t;
                               t =   f[a + 1 ][y1][x2][y2][k - 1 ] + s[x1][y1][a][y2] *
s[x1][y1][a][y2];
                               if   (tmp   >   t)
                               tmp   =   t;
                         }
                        // 横切
                         for   (b = y1; b < y2; b ++ )
                         {
                           t   =   f[x1][y1][x2][b][k - 1 ] + s[x1][b + 1 ][x2][y2] * s[x1][b
+ 1 ][x2][y2];
                            if   (tmp   >   t)
                               tmp   =   t;
                             t = f[x1][b + 1 ][x2][y2][k - 1 ] + s[x1][y1][x2][b] *
s[x1][y1][x2][b];
                            if   (tmp   >   t)
                               tmp   =   t;
```

```
                        }
                        f[x1][y1][x2][y2][k]    =    tmp;
                    }
        }
        printf( "%.3f\n" , sqrt( (double)(f[ 1 ][ 1 ][ 8 ][ 8 ][n]) / (double) (n) - p * p));
         return      0 ;
}
```

Pku1390--**Blocks**

Description

Some of you may have played a game called 'Blocks'. There are n blocks in a row, each box has a color. Here is an example: Gold, Silver, Silver, Silver, Silver, Bronze, Bronze, Bronze, Gold.
The corresponding picture will be as shown below:

Figure 1

If some adjacent boxes are all of the same color, and both the box to its left(if it exists) and its right(if it exists) are of some other color, we call it a 'box segment'. There are 4 box segments. That is: gold, silver, bronze, gold. There are 1, 4, 3, 1 box(es) in the segments respectively.

Every time, you can click a box, then the whole segment containing that box DISAPPEARS. If that segment is composed of k boxes, you will get k*k points. for example, if you click on a silver box, the silver segment disappears, you got 4*4=16 points.

Now let's look at the picture below:

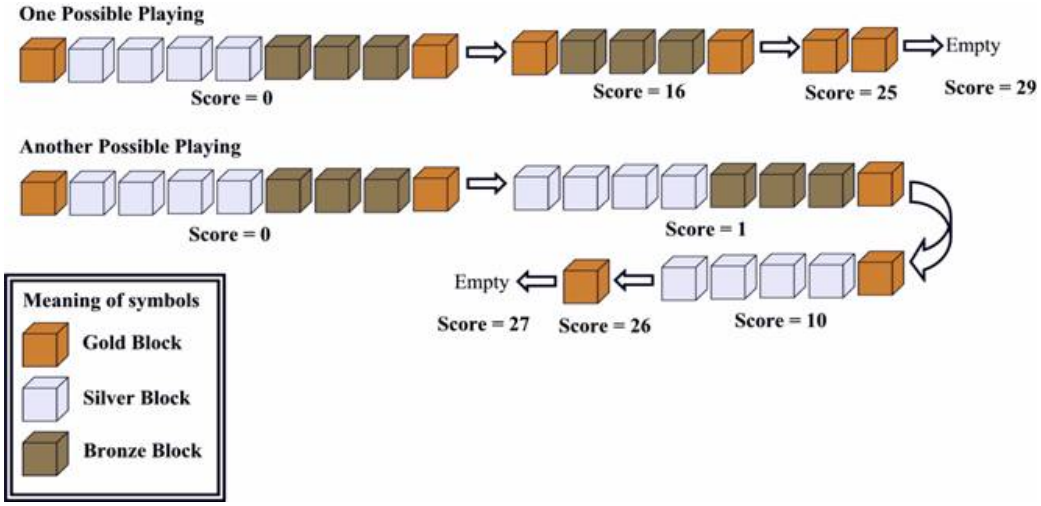


Figure 2

The first one is OPTIMAL.
Find the highest score you can get, given an initial state of this game.

Input

The first line contains the number of tests t(1<=t<=15). Each case contains two lines. The first line contains an integer n(1<=n<=200), the number of boxes. The second line contains n integers, representing the colors of each box. The integers are in the range 1~n.

Output

For each test case, print the case number and the highest possible score.

Sample Input

```
2
9
1 2 2 2 2 3 3 3 1
1
1
```

Sample Output

```
Case 1: 29
Case 2: 1
```

```
#include <iostream>
using namespace std;
int n, arr[205], len[205], ll;
int s[205][205][205];
void input(void) {
    int tmp;
    cin >> n;
    ll = 0;
    arr[0] = 0;
    for (int i = 1; i <= n; i++) {
        cin >> tmp;
        if (tmp == arr[ll]) {
            len[ll]++;
        } else {
            ll++;
```

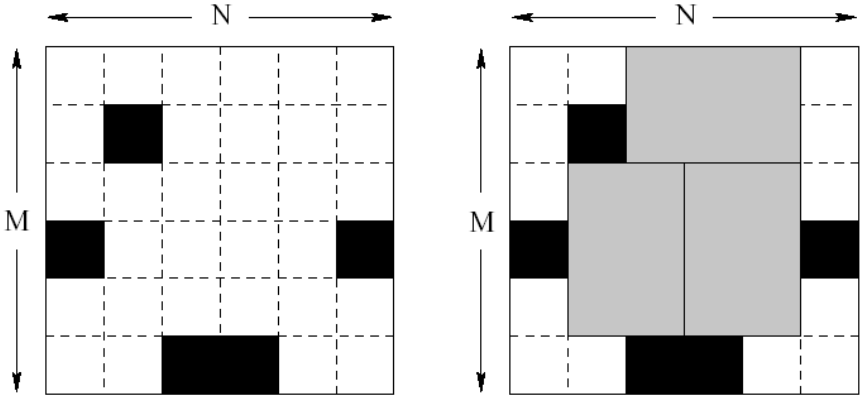
```
                arr[ll] = tmp;
                len[ll] = 1;
            }
        }
        for (int i = 0; i <= ll; i++)
        for (int j = 0; j <= ll; j++)
        for (int k = 0; k <= ll; k++)
            s[i][j][k] = -1;
        for (int i = 1; i <= ll; i++)
            s[i][i-1][0] = 0;
        return;
}
int f(int i, int j, int k) {
        int tmp;
        if (s[i][j][k] > -1)
            return s[i][j][k];
        tmp = f(i, j-1, 0) + (k+len[j])*(k+len[j]);
        for (int p = i; p < j; p++) {
            if (arr[p] == arr[j]) {
                tmp = max(tmp,
                    f(i, p, k+len[j]) + f(p+1, j-1, 0));
            }
        }
        s[i][j][k] = tmp;
        return tmp;
}
int main(void) {
        int t, tmp;
        cin >> t;
        for (int i = 1; i <= t; i++) {
            input();
            tmp = f(1, ll, 0);
            cout << "Case " << i << ": " << tmp << endl;
        }
        return 0;
}
```

Pku1038--Bugs Integrated, Inc.

Description

Bugs Integrated, Inc. is a major manufacturer of advanced memory chips. They are launching production of a new six terabyte Q-RAM chip. Each chip consists of six

unit squares arranged in a form of a 2*3 rectangle. The way Q-RAM chips are made is such that one takes a rectangular plate of silicon divided into N*M unit squares. Then all squares are tested carefully and the bad ones are marked with a black marker.



Finally, the plate of silicon is cut into memory chips. Each chip consists of 2*3 (or 3*2) unit squares. Of course, no chip can contain any bad (marked) squares. It might not be possible to cut the plate so that every good unit square is a part of some memory chip. The corporation wants to waste as little good squares as possible. Therefore they would like to know how to cut the plate to make the maximum number of chips possible.

Task

You are given the dimensions of several silicon plates and a list of all bad unit squares for each plate. Your task is to write a program that computes for each plate the maximum number of chips that can be cut out of the plate.

Input

The first line of the input file consists of a single integer D (1 <= D <= 5), denoting the number of silicon plates. D blocks follow, each describing one silicon plate. The first line of each block contains three integers N (1 <= N <= 150), M (1 <= M <= 10), K (0 <= K <= MN) separated by single spaces. N is the length of the plate, M is its height and K is the number of bad squares in the plate. The following K lines contain a list of bad squares. Each line consists of two integers x and y (1 <= x <= N, 1 <= y <= M) ?coordinates of one bad square (the upper left square has coordinates [1, 1], the bottom right is [N,M]).

Output

For each plate in the input file output a single line containing the maximum number of memory chips that can be cut out of the plate.

Sample Input

```
2
6 6 5
1 4
4 6
2 2
3 6
6 4
6 5 4
3 3
6 1
6 2
6 4
```

Sample Output

```
3
4
```

```
#define STATE_NUM 59049
#include "stdio.h"
#include "memory.h"
int D;
int N, M, K;
int plate[151][11], num[2][STATE_NUM];
int P[11], Q[11];
void search(int i, int j, int *P, int *Q, int chipNum, int PNum, int QNum);
inline int getnum(int *p, int n);
inline void getCode(int *p, int n, int state);
int maxNum;
int main(int argc, char *argv[])
{
	int i, j, state, k;
	scanf("%d", &D);
	while (D--) {
	maxNum = 0;
	scanf("%d %d %d", &N, &M, &K);
	memset(plate, 0, sizeof(plate));
	memset(num, -1, sizeof(num));
	for (i = 1; i <= K; ++i) {
		scanf("%d %d", &plate[0][0], &plate[0][1]);
		plate[plate[0][0]][plate[0][1]] = 1;
	}
	state = 1;
	for (i = 1; i <= M; ++i) {
		P[i] = plate[1][i] + 1;
```

```
            state *= 3;
        }
        num[0][getnum(P, M)] = 0;
        for (i = 1; i < N; ++i) {
            memset(num[i % 2], -1, sizeof(num[i % 2]));
            for (j = 0; j < state; ++j) {
            if (num[(i - 1) % 2][j] == -1)
                continue;
            getCode(P, M, j);
            for (k = 1; k <= M; ++k) {
                Q[k] = P[k] == 0 ? P[k] : P[k] - 1;
                if (plate[i + 1][k] == 1)
                Q[k] = 2;
            }
            search(i, 1, P, Q, num[(i - 1) % 2][j], getnum(P, M),
                    getnum(Q, M));
            }
        }
        printf("%d\n", maxNum);
        }
        return 0;
}
void search(int i, int k, int *PP, int *QQ, int chipNum, int PNum,
            int QNum)
{
        for (int j = k; j <= M + 3; ++j) {
        if (chipNum > num[i % 2][QNum])
            num[i % 2][QNum] = chipNum;
        if (chipNum > maxNum)
            maxNum = chipNum;
        if (j >= M)
            return;
        else {
            if (PP[j] == 0 && PP[j + 1] == 0 && QQ[j] == 0
            && QQ[j + 1] == 0) {
            QQ[j] = QQ[j + 1] = 2;
            search(i, j + 2, PP, QQ, chipNum + 1, PNum, getnum(QQ, M));
            QQ[j] = QQ[j + 1] = 0;
            }
            if (j + 2 <= M) {
            if (PP[j] <= 1 && PP[j + 1] <= 1 && PP[j + 2] <= 1 &&
                QQ[j] == 0 && QQ[j + 1] == 0 && QQ[j + 2] == 0) {
                QQ[j] = QQ[j + 1] = QQ[j + 2] = 2;
                search(i, j + 3, PP, QQ, chipNum + 1, PNum,
```

```
                    getnum(QQ, M));
                QQ[j] = QQ[j + 1] = QQ[j + 2] = 0;
            }
            }
        }
        }
}
int getnum(int *p, int n)
{
    int sum = 0;
    for (int i = 1; i <= n; ++i)
    sum = sum * 3 + p[i];
    return sum;
}
void getCode(int *p, int n, int state)
{
    int i = n;
    while (state) {
    p[i] = state % 3;
    state /= 3;
    --i;
    }
    for (; i >= 0; --i)
    p[i] = 0;
}
```

Pku1947--Rebuilding Roads

Description

The cows have reconstructed Farmer John's farm, with its N barns (1 <= N <= 150, number 1..N) after the terrible earthquake last May. The cows didn't have time to rebuild any extra roads, so now there is exactly one way to get from any given barn to any other barn. Thus, the farm transportation system can be represented as a tree.

Farmer John wants to know how much damage another earthquake could do. He wants to know the minimum number of roads whose destruction would isolate a subtree of exactly P (1 <= P <= N) barns from the rest of the barns.

Input

* Line 1: Two integers, N and P
* Lines 2..N: N-1 lines, each with two integers I and J. Node I is node J's parent in the tree of roads.

Output

A single line containing the integer that is the minimum number of roads that need to be destroyed for a subtree of P nodes to be isolated.

Sample Input

```
11 6
1 2
1 3
1 4
1 5
2 6
2 7
2 8
4 9
4 10
4 11
```

Sample Output

```
2
```

Hint

[A subtree with nodes (1, 2, 3, 6, 7, 8) will become isolated if roads 1-4 and 1-5 are destroyed.]

```
#include <iostream>
#include <algorithm>
using namespace std;
const int MAXN = 200;
const int maxint = 0x7f7f7f7f;
int son[MAXN+1], bro[MAXN+1];
int f[MAXN+1][MAXN+1];
int n, p, ans;

void readln() {
```

```
    int a, b;
    cin >> n >> p;
    memset(son, 0, sizeof(son));
    memset(bro, 0, sizeof(bro));
    for (int i = 1; i <= n-1; i++) {
        cin >> a >> b;
        bro[b] = son[a];
        son[a] = b;
    }
}
int dp(int v, int j) {
    int re, t1, t2, k;
    if (f[v][j] < maxint)
        return f[v][j];
    if (v == 0 && j == 0)
        return f[0][0] = 0;
    if (v == 0 && j != 0)
        return f[0][j] = -1;
    re = dp(bro[v], j);
    if (re != -1)
        re++;
    else
        re = maxint;
    for (k = 0; k <= j-1; k++) {
        t1 = dp(son[v], k);
        t2 = dp(bro[v], j-k-1);
        if (t1 < 0 || t2 < 0)
            continue;
        if (t1+t2 < re)
            re = t1+t2;
    }
    if (re == maxint)
        re = -1;
     return f[v][j] = re;
}
void work() {
    memset(f, 127, sizeof(f));
    int i, t;
    ans = dp(son[1], p-1);
    for (i = 2; i <= n; i++) {
        t = dp(son[i], p-1);
        if (t > -1 && t+1 < ans)
            ans = t+1;
    }
```

```
    cout << ans << endl;
}
int main(void) {
    readln();
    work();
    return 0;
}
```

Pku1691--**Painting A Board**

Description

The CE digital company has built an Automatic Painting Machine (APM) to paint a flat board fully covered by adjacent non-overlapping rectangles of different sizes each with a predefined color.

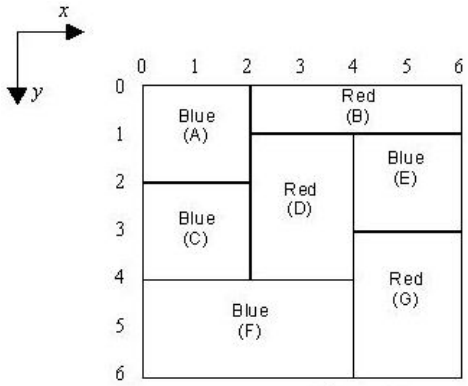


Figure 1

To color the board, the APM has access to a set of brushes. Each brush has a distinct color C. The APM picks one brush with color C and paints all possible rectangles having predefined color C with the following restrictions:
To avoid leaking the paints and mixing colors, a rectangle can only be painted if all rectangles immediately above it have already been painted. For example rectangle labeled F in Figure 1 is painted only after rectangles C and D are painted. Note that each rectangle must be painted at once, i.e. partial painting of one rectangle is not allowed.
You are to write a program for APM to paint a given board so that the number of brush pick-ups is minimum. Notice that if one brush is picked up more than once, all pick-ups are counted.

Input

The first line of the input file contains an integer M which is the number of test cases to solve (1 <= M <= 10). For each test case, the first line contains an integer N, the number of rectangles, followed by N lines describing the rectangles. Each rectangle R is specified by 5 integers in one line: the y and x coordinates of the upper left corner of R, the y and x coordinates of the lower right corner of R, followed by the color-code of R.
Note that:

1. Color-code is an integer in the range of 1 .. 20.
2. Upper left corner of the board coordinates is always (0,0).
3. Coordinates are in the range of 0 .. 99.
4. N is in the range of 1..15.

Output

One line for each test case showing the minimum number of brush pick-ups.

Sample Input

```
1
7
0 0 2 2 1
0 2 1 6 2
2 0 4 2 1
1 2 4 4 2
1 4 3 6 1
4 0 6 4 1
3 4 6 6 2
```

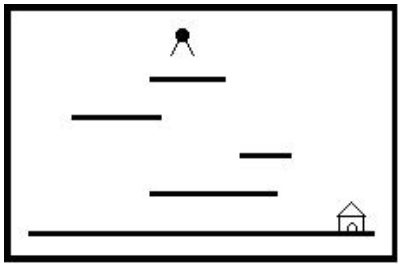
Sample Output

```
3
```

Pku1661--Help Jimmy

Description

"Help Jimmy" 是在下图所示的场景上完成的游戏。



场景中包括多个长度和高度各不相同的平台。地面是最低的平台，高度为零，长度无限。

Jimmy 老鼠在时刻 0 从高于所有平台的某处开始下落，它的下落速度始终为 1 米/秒。当 Jimmy 落到某个平台上时，游戏者选择让它向左还是向右跑，它跑动的速度也是 1 米/秒。当 Jimmy 跑到平台的边缘时，开始继续下落。Jimmy 每次下落的高度不能超过 MAX 米，不然就会摔死，游戏也会结束。

设计一个程序，计算 Jimmy 到底地面时可能的最早时间。

Input

第一行是测试数据的组数 t（0 <= t <= 20）。每组测试数据的第一行是四个整数 N，X，Y，MAX，用空格分隔。N 是平台的数目（不包括地面），X 和 Y 是 Jimmy 开始下落的位置的横竖坐标，MAX 是一次下落的最大高度。接下来的 N 行每行描述一个平台，包括三个整数，X1[i]，X2[i]和 H[i]。H[i]表示平台的高度，X1[i]和 X2[i]表示平台左右端点的横坐标。1 <= N <= 1000，-20000 <= X, X1[i], X2[i] <= 20000，0 < H[i] < Y <= 20000（i = 1..N）。所有坐标的单位都是米。

Jimmy 的大小和平台的厚度均忽略不计。如果 Jimmy 恰好落在某个平台的边缘，被视为落在平台上。所有的平台均不重叠或相连。测试数据保证问题一定有解。

Output

对输入的每组测试数据，输出一个整数，Jimmy 到底地面时可能的最早时间。

Sample Input

```
1
3 8 17 20
0 10 8
0 10 13
4 14 3
```

Sample Output

```
23
```

```
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#define MAX_N 1000
#define INFINITE 1000000
int t, n, x, y, max;
struct platform {
	int lx, rx, h;
};
platform aplatform[MAX_N + 10];
int aleftmintime[MAX_N + 10];
int arightmintime[MAX_N + 10];
int comp (const void *a, const void *b) {
	platform *p1, *p2;
	p1 = (platform *)a;
	p2 = (platform *)b;
	return p2->h - p1->h;
}
int mintime(int L, bool bleft) {
	int y = aplatform[L].h;
	int x, i;
	if (bleft) {
		x = aplatform[L].lx;
	} else {
		x = aplatform[L].rx;
	}

	for (i = L+1; i <=n; i++) {
		if (aplatform[i].lx <=x && aplatform[i].rx >=x)
			break;
	}
	if (i <= n) {
		if (y - aplatform[i].h > max)
			return INFINITE;
	} else {
		if (y > max) {
			return INFINITE;
		} else {
			return y;
		}
	}
	int nlefttime = y - aplatform[i].h + x - aplatform[i].lx;
	int nrighttime = y - aplatform[i].h + aplatform[i].rx - x;
```

```
        if (aleftmintime[i] == -1)
            aleftmintime[i] = mintime(i, true);
        if (arightmintime[i] == -1)
            arightmintime[i] = mintime(i, false);
        nlefttime += aleftmintime[i];
        nrighttime += arightmintime[i];

        if (nlefttime < nrighttime)
            return nlefttime;
        return nrighttime;
}
int main(void) {
        scanf ("%d", &t);
        for (int i = 0; i < t; i++) {
            memset (aleftmintime, -1, sizeof(aleftmintime));
            memset (arightmintime, -1, sizeof(arightmintime));
            scanf ("%d%d%d%d", &n, &x, &y, &max);
            aplatform[0].lx = x;
            aplatform[0].rx = x;
            aplatform[0].h = y;
            for (int j = 1; j <= n; j++)
                scanf ("%d%d%d", &aplatform[j].lx, &aplatform[j].rx, &aplatform[j].h);
            qsort (aplatform, n+1, sizeof(platform), comp);
            printf ("%d\n", mintime(0, true));
        }
        return 0;
}
```

Pku1631--**Bridging signals**

Description

'Oh no, they've done it again', cries the chief designer at the Waferland chip factory. Once more the routing designers have screwed up completely, making the signals on the chip connecting the ports of two functional blocks cross each other all over the place. At this late stage of the process, it is too expensive to redo the routing. Instead, the engineers have to bridge the signals, using the third dimension, so that no two signals cross. However, bridging is a complicated operation, and thus it is desirable to bridge as few signals as possible. The call for a computer program that finds the maximum number of signals which may be connected on the silicon surface without crossing each other, is imminent. Bearing in mind that there may be thousands of signal ports at the boundary of a functional block, the problem asks quite a lot of the programmer. Are you up to the task?

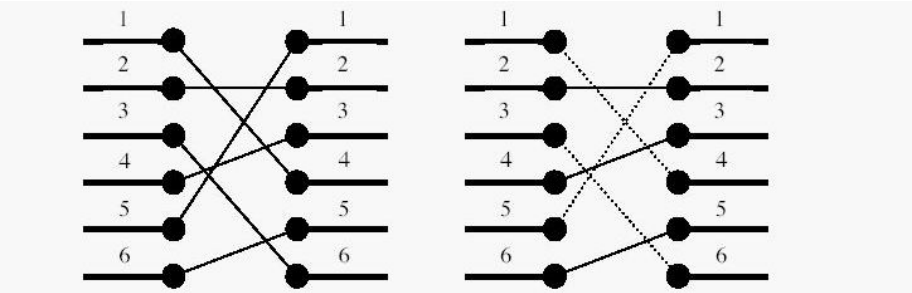


Figure 1. To the left: The two blocks' ports and their signal mapping (4,2,6,3,1,5). To the right: At most three signals may be routed on the silicon surface without crossing each other. The dashed signals must be bridged.

A typical situation is schematically depicted in figure 1. The ports of the two functional blocks are numbered from 1 to p, from top to bottom. The signal mapping is described by a permutation of the numbers 1 to p in the form of a list of p unique numbers in the range 1 to p, in which the i:th number specifies which port on the right side should be connected to the i:th port on the left side.Two signals cross if and only if the straight lines connecting the two ports of each pair do.

Input

On the first line of the input, there is a single positive integer n, telling the number of test scenarios to follow. Each test scenario begins with a line containing a single positive integer p < 40000, the number of ports on the two functional blocks. Then follow p lines, describing the signal mapping:On the i:th line is the port number of the block on the right side which should be connected to the i:th port of the block on the left side.

Output

For each test scenario, output one line containing the maximum number of signals which may be routed on the silicon surface without crossing each other.

Sample Input

```
4 6 4 2 6 3 1 5 10 2 3 4 5 6 7 8 9 10 1 8 8 7 6 5 4 3 219589231746
```

Sample Output

```
3 9 1 4
```

```
#include <stdio.h>
#include <string.h>
#define N 40005
```

```
int arr[N];
int main(void) {
	int ncas, p, c, t, i;
	int left ,right, mid;
	scanf ("%d", &ncas);
	while (ncas--) {
		scanf ("%d", &p);
		if (p == 0) {
			printf ("0\n");
			continue;
		}
		memset (arr, 0, sizeof(arr));
		c = 1;
		scanf ("%d", &t);
		arr[0] = t;
		for (i = 1; i < p; i++) {
			scanf ("%d", &t);
			if (t > arr[c - 1]) {
				arr[c++] = t;
			} else {
				left = 0;
				right = c - 1;
				mid = (left + right)/2;
				while (left < right) {
					if (arr[mid] < t)
						left = mid + 1;
					else if (arr[mid] > t)
						right = mid;
					mid = (left + right)/2;
				}
				arr[mid] = t;
			}
		}
		printf ("%d\n", c);
	}
	return 0;
}
```

Pku1458--Common Subsequence

Description

A subsequence of a given sequence is the given sequence with some elements (possible none) left out. Given a sequence X = < x1, x2, ..., xm > another sequence Z = < z1, z2, ..., zk > is a subsequence of X if there exists a strictly increasing sequence < i1, i2, ..., ik > of indices of X such that for all j = 1,2,...,k, x_{ij} = zj. For example, Z = < a, b, f, c > is a subsequence of X = < a, b, c, f, b, c > with index sequence < 1, 2, 4, 6 >. Given two sequences X and Y the problem is to find the length of the maximum-length common subsequence of X and Y.

Input

The program input is from the std input. Each data set in the input contains two strings representing the given sequences. The sequences are separated by any number of white spaces. The input data are correct.

Output

For each set of data the program prints on the standard output the length of the maximum-length common subsequence from the beginning of a separate line.

Sample Input

```
abcfbc         abfcab
programming    contest
abcd           mnp
```

Sample Output

```
4
2
0
```

```
#include          <stdio.h>
#include          <string.h>
int lcslength(char x[], char y[])
{
        int m, n;
        int i, j;
        int c[1100][1100];
        m = strlen(&x[1]);
        n = strlen(&y[1]);
        for(i=0; i<m; i++)
                c[i][0]=0;
```

```
        for(j=0; j<n; j++)
                c[0][j]=0;
        for(i=1; i<=m; i++)
                for(j=1; j<=n; j++)
                {
                        if(x[i]==y[j])
                        {
                                c[i][j]=c[i-1][j-1]+1;
                        }
                        else if(c[i-1][j]>=c[i][j-1])
                        {
                                c[i][j] = c[i-1][j];
                        }
                        else
                        {
                                c[i][j] = c[i][j-1];
                        }
                }
        return c[m][n];
}
int main()
{
        char a[1000], b[1000];
        int len, i;
        int an;
        while(scanf("%s", &a[1])!=EOF)
        {
                scanf("%s", &b[1]);
                an = lcslength(a, b);
                printf("%d\n", an);
        }
        return 0;
}
```

Pku1161--**Post Office**

Description

There is a straight highway with villages alongside the highway. The highway is represented as an integer axis, and the position of each village is identified with a single integer coordinate. There are no two villages in the same position. The distance between two positions is the absolute value of the difference of their integer coordinates.

Post offices will be built in some, but not necessarily all of the villages. A village and the post office in it have the same position. For building the post offices, their positions should be chosen so that the total sum of all distances between each village and its nearest post office is minimum.
You are to write a program which, given the positions of the villages and the number of post offices, computes the least possible sum of all distances between each village and its nearest post office.

Input

Your program is to read from standard input. The first line contains two integers: the first is the number of villages V, 1 <= V <= 300, and the second is the number of post offices P, 1 <= P <= 30, P <= V. The second line contains V integers in increasing order. These V integers are the positions of the villages. For each position X it holds that 1 <= X <= 10000.

Output

The first line contains one integer S, which is the sum of all distances between each village and its nearest post office.

Sample Input

```
10 5
1 2 3 6 7 9 11 22 44 50
```

Sample Output

```
9
```

```
#include<cstdio>
#include <algorithm>
using namespace std;
int n, m, a[301];
int d[31][301], //多少邮局，前村庄
	c[301][301];
int main()
{
	int i, j, k;
	scanf("%d%d", &n, &m);
	for (i = 1; i <= n; i++)
```

```
        scanf("%d", &a[i]);
        for (i = 1; i < n; i++) {
        for (j = i + 1; j <= n; j++) {
                c[i][j] = c[i][j - 1] + a[j] - a[(i + j) / 2];
                if (i == 1)
                d[1][j] = c[i][j];
        }
        }
        for (i = 2; i <= m; i++)
        for (j = i + 1; j <= n; j++) {
                d[i][j] = 2147483647;
                for (k = i; k < j; k++)
                d[i][j] = min(d[i - 1][k] + c[k + 1][j], d[i][j]);
        }
        printf("%d", d[m][n]);
        return 0;
}
#include <iostream>
#include <cmath>
using namespace std;
int p, v, arr[302], cost[302][302];

int Dis(int x, int y) {
        int m = (x + y) >> 1;
        int sum = 0;
        for (int i = x; i <= y; i++) {
                sum += abs(arr[i] - arr[m]);
        }
        return sum;
}
int main(void) {
        cin >> v >> p;
        for (int i = 1; i <= v; i++)
                cin >> arr[i];
        for (int i = 1; i <= v; i++)
                cost[i][1] = Dis(1, i);
        for (int j = 1; j <= p; j++)
                cost[j][j] = 0;
        for (int i = 3; i <= v; i++) {
                int jj = (i <= p)?i:p;
                for (int j = 2; j <= jj; j++) {
                        int min = Dis(j, i);
                        for (int k = j; k <= i-1; k++) {
                                int t = cost[k][j-1] + Dis(k+1, i);
```

```
                    if (min > t)
                        min = t;
                }
                cost[i][j] = min;
            }
        }
        cout << cost[v][p] << endl;
        return 0;
}
```

Pku1159--**Palindrome**

Description

A palindrome is a symmetrical string, that is, a string read identically from left to right as well as from right to left. You are to write a program which, given a string, determines the minimal number of characters to be inserted into the string in order to obtain a palindrome.

As an example, by inserting 2 characters, the string "Ab3bd" can be transformed into a palindrome ("dAb3bAd" or "Adb3bdA"). However, inserting fewer than 2 characters does not produce a palindrome.

Input

Your program is to read from standard input. The first line contains one integer: the length of the input string N, 3 <= N <= 5000. The second line contains one string with length N. The string is formed from uppercase letters from 'A' to 'Z', lowercase letters from 'a' to 'z' and digits from '0' to '9'. Uppercase and lowercase letters are to be considered distinct.

Output

Your program is to write to standard output. The first line contains one integer, which is the desired minimal number.

Sample Input

```
5
Ab3bd
```

Sample Output

```
2
#include <stdio.h>
#include <stdlib.h>
#define N 5002
char arr[N];
short min[N][N];
int main(void) {
	int n, an;
	int i, j, k;
	while (scanf("%d", &n) != EOF) {
		scanf ("%s", arr);
		for (i = 0; i < n; i++)
			min[i][i] = 0;
		for (k = 1; k < n; k++)
		for (i = 0; i < n-k; i++) {
			if (arr[i] == arr[i+k])
				min[i][i+k] = min[i+1][i-1+k];
			else
				min[i][i+k] = 1 + (min[i][i-1+k]
				< min[i+1][i+k]?min[i][i-1+k]:min[i+1][i+k]);
		}
		printf("%d\n", min[0][n-1]);
	}
	return 0;
}
```

1.6 状态空间搜索

1.6.1 状态空间

Pku1480--Optimal Programs

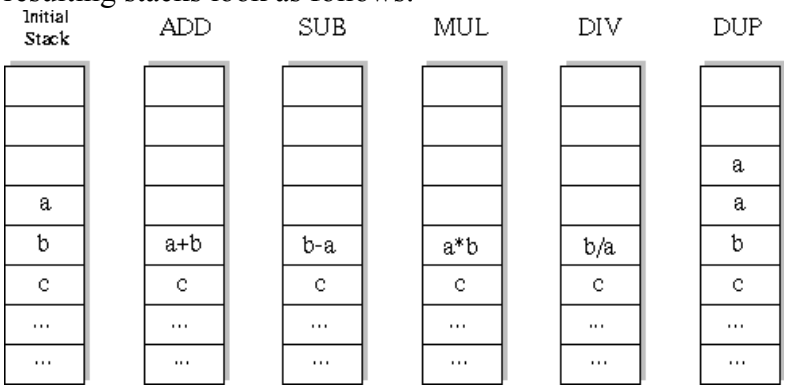
Description

As you know, writing programs is often far from being easy. Things become even harder if your programs have to be as fast as possible. And sometimes there is reason for them to be. Many large programs such as operating systems or databases have "bottlenecks" - segments of code that get executed over and over again, and make up

for a large portion of the total running time. Here it usually pays to rewrite that code portion in assembly language, since even small gains in running time will matter a lot if the code is executed billions of times.
In this problem we will consider the task of automating the generation of optimal assembly code. Given a function (as a series of input/output pairs), you are to come up with the shortest assembly program that computes this function.

The programs you produce will have to run on a stack based machine, that supports only five commands: ADD, SUB, MUL, DIV and DUP. The first four commands pop the two top elements from the stack and push their sum, difference, product or integer quotient , respectively, on the stack. The DUP command pushes an additional copy of the top-most stack element on the stack.

So if the commands are applied to a stack with the two top elements a and b (shown to the left), the resulting stacks look as follows:



At the beginning of the execution of a program, the stack will contain a single integer only: the input. At the end of the computation, the stack must also contain only one integer; this number is the result of the computation.
There are three cases in which the stack machine enters an error state:
A DIV-command is executed, and the top-most element of the stack is 0.
A ADD, SUB, MUL or DIV-command is executed when the stack contains only one element.
An operation produces a value greater than 30000 in absolute value.

Input

The input consists of a series of function descriptions. Each description starts with a line containing a single integer n (n <= 10), the number of input/output pairs to follow. The following two lines contains n integers each: x1, x2, ..., xn in the first line (all different), and y1, y2, ..., yn in the second line. The numbers will be no more than 30000 in absolute value.
The input is terminated by a test case starting with n = 0. This test case should not be processed.

Output

You are to find the shortest program that computes a function f , such that f(xi) = yi for all 1 <= i <= n. This implies that the program you output may not enter an error state if executed on the inputs xi (although it may enter an error state for other inputs). Consider only programs that have at most 10 statements.
For each function description, output first the number of the description. Then print out the se- quence of commands that make up the shortest program to compute the given function. If there is more than one such program, print the lexicographically smallest. If there is no program of at most 10 statements that computes the function, print the string `` Impossible". If the shortest program consists of zero commands, print ``Empty Sequence".
Output a blank line after each test case.

Sample Input

```
4
1 2 3 4
0 -2 -6 -12
3
1 2 3
1 11 1998
1
1998
1998
0
```

Sample Output

```
Program 1
DUP DUP MUL SUB
Program 2
Impossible
Program 3
Empty sequence
```

```
1 /* Accepted 1480 C++ 00:00.48 28620K */
2 #include <iostream>
3
4 using namespace std;
5
6 enum { ADD = 1, DIV = 2, DUP = 3, MUL = 4, SUB = 5, maxStackSize = 11 };
7
```

```
 8 inline int abs(int n)
 9 {
10         return n > 0 ? n : -n;
11 }
12
13 template <class T>
14 class stack
15 {
16 public:
17         T x[maxStackSize]; char p;
18
19         stack() { p = -1; }
20         int size() { return p + 1;}
21         bool empty() { return p == -1; }
22         void push(const T & item) { x[++p] = item; }
23         void pop() { p--; }
24         void clear() { p = -1; }
25         T & top() { return x[p]; }
26
27         void operator = (const stack & st)
28         {
29                 memcpy(x, st.x, sizeof(st.x));
30                 p = st.p;
31         }
32 };
33
34 struct QUEUE
35 {
36         int last;
37         char cnt, op;
38         stack <short> st;
39 }queue[888888];
40
41 string opSeq;
42 int n, x[10], y[10];
43
44 void getPath(int i)
45 {
46         if(queue[i].last)
47                 getPath(queue[i].last);
48         opSeq += queue[i].op;
49 }
50
51 bool checkOthers()
```

```
52 {
53      for(int i = 1; i < n; i++)
54      {
55          stack <short> st;
56          st.push(x[i]);
57          int a, b;
58          for(int k = 0; k < opSeq.size(); k++)
59              switch(opSeq[k])
60              {
61                  case 1 : //ADD
62                      a = st.top(); st.pop();
63                      b = st.top(); st.pop();
64                      if(abs(int(a) + int(b)) > 30000)
65                          return false;
66                      st.push(a + b);
67                      break;
68                  case 2 : //DIV
69                      a = st.top(); st.pop();
70                      b = st.top(); st.pop();
71                      if(a == 0) return false;
72                      st.push(b / a);
73                      break;
74                  case 3 : //DUP
75                      st.push(st.top()); break;
76                  case 4 : //MUL
77                      a = st.top(); st.pop();
78                      b = st.top(); st.pop();
79                      if(abs(int(a) * int(b)) > 30000)
80                          return false;
81                      st.push(a * b);
82                      break;
83                  case 5 : //SUB
84                      a = st.top(); st.pop();
85                      b = st.top(); st.pop();
86                      if(abs(int(b) - int(a)) > 30000)
87                          return false;
88                      st.push(b - a);
89                      break;
90              }
91          if(st.top() != y[i])
92              return false;
93      }
94      return true;
95 }
```

```
 96
 97 int main()
 98 {
 99         int program = 0;
100         while((cin >> n) && n)
101         {
102
103                 for(int i = 0; i < n; i++)
104                         cin >> x[i];
105                 for(int i = 0; i < n; i++)
106                         cin >> y[i];
107
108                 program++;
109                 cout << "Program " << program << endl;
110
111                 bool empty_sequence = true;
112                 for(int i = 0; i < n; i++)
113                         if(x[i] != y[i])
114                         {
115                                 empty_sequence = false; break;
116                         }
117                 if(empty_sequence)
118                 {
119                         cout << "Empty sequence" << endl << endl; continue;
120                 }
121
122                 int front = -1, rear = 0;
123                 int best = INT_MAX; string bestSeq(10, 255);
124                 bool found = false;
125
126                 queue[0].st.push(x[0]);
127                 queue[0].last = 0;
128                 queue[0].cnt = 0;
129                 queue[0].op = 0;
130                 while(front < rear)
131                 {
132                         front++;
133
134                         if(queue[front].cnt > best)
135                                 continue;
136
137                         stack <short> st = queue[front].st;
138
139                         if(st.size() == 1 && st.top() == y[0])
```

```
140             {
141                 opSeq.clear();
142                 getPath(front);
143                 if(checkOthers())
144                 {
145                     if(queue[front].cnt == best)
146                         if(opSeq < bestSeq)
147                             bestSeq = opSeq;
148                     if(queue[front].cnt < best)
149                     {
150                         best = queue[front].cnt;
151                         bestSeq = opSeq;
152                     }
153                     found = true;
154                 }
155             }
156
157             int a, b;
158             if(queue[front].cnt < 10)
159             {
160                 //ADD
161                 if(st.size() > 1)
162                 {
163                     rear++;
164                     queue[rear].st = st;
165                     queue[rear].op = ADD;
166                     queue[rear].cnt = queue[front].cnt + 1;
167                     queue[rear].last = front;
168                     a = queue[rear].st.top(); queue[rear].st.pop();
169                     b = queue[rear].st.top(); queue[rear].st.pop();
170                     queue[rear].st.push(a + b);
171                     if(abs(int(a) + int(b)) > 30000)
172                         rear--;
173                 }
174
175                 //DIV
176                 if(st.size() > 1 && st.top() != 0)
177                 {
178                     rear++;
179                     queue[rear].st = st;
180                     queue[rear].op = DIV;
181                     queue[rear].cnt = queue[front].cnt + 1;
182                     queue[rear].last = front;
183                     a = queue[rear].st.top(); queue[rear].st.pop();
```

```
184                         b = queue[rear].st.top(); queue[rear].st.pop();
185                         queue[rear].st.push(b / a);
186                     }
187
188                     //DUP
189                     rear++;
190                     queue[rear].st = st;
191                     queue[rear].op = DUP;
192                     queue[rear].cnt = queue[front].cnt + 1;
193                     queue[rear].last = front;
194                     queue[rear].st.push(st.top());
195
196                     //MUL
197                     if(st.size() > 1)
198                     {
199                         rear++;
200                         queue[rear].st = st;
201                         queue[rear].op = MUL;
202                         queue[rear].cnt = queue[front].cnt + 1;
203                         queue[rear].last = front;
204                         a = queue[rear].st.top(); queue[rear].st.pop();
205                         b = queue[rear].st.top(); queue[rear].st.pop();
206                         queue[rear].st.push(a * b);
207                         if(abs(int(a) * int(b)) > 30000)
208                             rear--;
209                     }
210
211                     //SUB
212                     if(st.size() > 1)
213                     {
214                         rear++;
215                         queue[rear].st = st;
216                         queue[rear].op = SUB;
217                         queue[rear].cnt = queue[front].cnt + 1;
218                         queue[rear].last = front;
219                         a = queue[rear].st.top(); queue[rear].st.pop();
220                         b = queue[rear].st.top(); queue[rear].st.pop();
221                         queue[rear].st.push(b - a);
222                         if(abs(int(b) - int(a)) > 30000)
223                             rear--;
224                     }
225                 }
226             }
227
```

```
228             if(found == false)
229                     cout << "Impossible" << endl;
230             else
231                     for(int i = 0; i < bestSeq.size(); i++)
232                     {
233                             switch(bestSeq[i])
234                             {
235                                     case 1 : cout << "ADD"; break;
236                                     case 2 : cout << "DIV"; break;
237                                     case 3 : cout << "DUP"; break;
238                                     case 4 : cout << "MUL"; break;
239                                     case 5 : cout << "SUB"; break;
240                             }
241                             cout << (i == bestSeq.size() - 1 ? '\n': ' ');
242                     }
243             cout << endl;
244
245             for(int i = 0; i <= rear; i++)
246                     queue[i].st.clear();
247         }
248
249         return 0;
250 }
```

Pku1011--**Sticks**

Description

George took sticks of the same length and cut them randomly until all parts became at most 50 units long. Now he wants to return sticks to the original state, but he forgot how many sticks he had originally and how long they were originally. Please help him and design a program which computes the smallest possible original length of those sticks. All lengths expressed in units are integers greater than zero.

Input

The input contains blocks of 2 lines. The first line contains the number of sticks parts after cutting, there are at most 64 sticks. The second line contains the lengths of those parts separated by the space. The last line of the file contains zero.

Output

The output should contains the smallest possible length of original sticks, one per line.

Sample Input

```
9
5 2 1 5 2 1 5 2 1
4
1 2 3 4
0
```

Sample Output

```
6
5
```

```
#include <iostream>
#include <algorithm>
#include <cstring>
using namespace std;
const int N = 100;
int n, arr[N], flag[N];
int sum, star, len;
bool cmp(const int a, const int b) {
	return a > b;
}
void input(void) {
	sum = 0;
	for (int i = 0; i < n; i++) {
		cin >> arr[i];
		sum += arr[i];
	}
	memset(flag, 0, sizeof(flag));
	sort(arr, arr+n, cmp);
	star = arr[0];
	return ;
}
int dfs(int length, int number, int t) {
	if (length == 0 && number == 0)
		return 1;
	if (length == 0) {
		length = len;
		t = 0;
	}
	for (int i = t; i < n; i++) {
```

```
            if (flag[i])
                continue;
            if (arr[i] > length)
                continue;
            flag[i] = 1;
            if (dfs(length - arr[i], number - 1, i))
                return 1;
            flag[i] = 0;
            if (length == arr[i] || length == len)
                break;
        }
        return 0;
}
int main(void) {
        while (cin >> n, n) {
            input();
            for (len = star; len < sum; len++) {
                if (sum % len)
                    continue;
                if (dfs(0, n, 0))
                    break;
            }
            cout << len << endl;
        }
        return 0;
}
```

Pku2362--**Square**

Description

Given a set of sticks of various lengths, is it possible to join them end-to-end to form a square?

Input

The first line of input contains N, the number of test cases. Each test case begins with an integer 4 <= M <= 20, the number of sticks. M integers follow; each gives the length of a stick - an integer between 1 and 10,000.

Output

For each case, output a line containing "yes" if is is possible to form a square; otherwise output "no".

Sample Input

```
3
4 1 1 1 1
5 10 20 30 40 50
8 1 7 2 6 4 4 3 5
```

Sample Output

```
yes
no
yes
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cstring>
using namespace std;
vector < int >sticks;
int mark[25];
int s;
int dfs(int val, int h, int cnt)
{
    for (int i = h; i < sticks.size(); ++i) {
    if (mark[i] == 0) {
        mark[i] = 1;
        if (val + sticks[i] == s) {
        cnt++;
        if (cnt == 4)
            return 1;
        int ans = dfs(0, 1, cnt);
        if (ans == 1)
            return 1;
        mark[i] = 0;
        return 0;
        } else if (val + sticks[i] < s) {
        int ans = dfs(val + sticks[i], i, cnt);
        if (ans == 1)
            return 1;
        mark[i] = 0;
        } else if (val + sticks[i] > s) {
```

```
            mark[i] = 0;
            return 0;
            }
        }
        if (h > 0 && mark[0] == 0)
            return 0;
        }
        return 0;
}
int main()
{
        int c;
        scanf("%d", &c);
        while (c--) {
        memset(mark, 0, sizeof(mark));
        sticks.clear();
        int num;
        int k;
        scanf("%d", &k);
        int sum = 0;
        int tmp = 0;
        while (k--) {
            scanf("%d", &num);
            sticks.push_back(num);
            sum += num;
            if (num > tmp)
            tmp = num;
        }
        if (sum % 4 != 0 || sticks.size() < 4) {
            printf("no\n");
            continue;
        }
        s = sum / 4;
        if (s < tmp) {
            printf("no\n");
            continue;
        }
        sort(sticks.begin(), sticks.end());
        int ans = dfs(0, 0, 1);
        if (ans) {
            printf("yes\n");
        } else
            printf("no\n");
        }
```

```
    return 0;
}
```

第2章　数学方法与常见模型

2.1　代数方法和模型

2.2　数论基础

2.2.1 素数和整除问题

Pku1811--Prime Test

Description

Given a big integer number, you are required to find out whether it's a prime number.

Input

The first line contains the number of test cases T (1 <= T <= 20), then the following T lines each contains an integer number N ($2 < N < 2^{54}$).

Output

For each test case, if N is a prime number, output a line containing the word "Prime", otherwise, output a line containing the smallest prime factor of N.

Sample Input

```
2
5
10
```

Sample Output

Prime
2

```
/*************************************************************************
这里包含了 rabinmiller 素数测试与 pollard 算法求解最小质因数的方法
*************************************************************************/
#include <stdio.h>
#include <stdlib.h>
typedef unsigned __int64 hugeint;
//求出最大公约数
hugeint gcd(hugeint A, hugeint B)
{
    while (A != 0){
        hugeint C = B % A;
        B = A;
        A = C;
    }
    return B;
}
//求 a*b%c，要求:a,b 的范围在 hugeint 范围的一般以内，在 hugeint 为 unsigned __int64 时，
a,b 需要是__int64 能表示的数
hugeint product_mod(hugeint A, hugeint B, hugeint C)
{
    hugeint R, D;
    R = 0;
    D = A;
    while (B > 0){
        if ( B&1 ) R = (R + D) % C;
        D = (D + D) % C;
        B >>=1;
    }
    return R;
}
//求 a^b%c，要求:a,b 的范围在 hugeint 范围的一般以内，在 hugeint 为 unsigned __int64 时，
a,b 需要是__int64 能表示的数
hugeint power_mod(hugeint A, hugeint B, hugeint C)
{
    hugeint R = 1, D = A;
    while (B ){
        if (B&1) R = product_mod(R, D, C);
        D = product_mod(D, D, C);
        B >>=1;
    }
    return R;
```

```
}

//给出随机数，可以简单的用 rand()代替
hugeint rAndom()
{
        hugeint a;
        a    = rand();
        a *= rand();
        a *= rand();
        a *= rand();
        return a;
}

//rabinmiller 方法测试 n 是否为质数
int pri[]={2,3,5,7,11,13,17,19,23,29};
bool isprime(hugeint n)
{
    if(n<2)
        return false;
    if(n==2)
        return true;
    if(!(n&1))
        return false;
    hugeint k = 0, i, j, m, a;
    m = n - 1;
    while(m % 2 == 0)
        m = (m >> 1), k++;
    for(i = 0; i < 10; i ++){
        if(pri[i]>=n)return 1;
        a = power_mod( pri[i], m, n );
        if(a==1)
            continue;
        for(j = 0; j < k; j ++){
            if(a==n-1)break;
            a = product_mod(a,a,n);
        }
        if(j < k)
            continue;
        return false ;
    }
    return true;
}
//pollard_rho 分解，给出 N 的一个非 1 因数，返回 N 时为一次没有找到
hugeint pollard_rho(hugeint C, hugeint N)
```

```
{
    hugeint I, X, Y, K, D;
    I = 1;
    X = rand() % N;
    Y = X;
    K = 2;
    do{
        I++;
        D = gcd(N + Y - X, N);
        if (D > 1 && D < N) return D;
        if (I == K) Y = X, K *= 2;
        X = (product_mod(X, X, N) + N - C) % N;
    }while (Y != X);
    return N;
}
//找出 N 的最小质因数
hugeint rho(hugeint N)
{
    if (isprime(N)) return N;
    do{
        hugeint T = pollard_rho(rand() % (N - 1) + 1, N);
        if (T < N){
            hugeint A, B;
            A = rho(T);
            B = rho(N / T);
            return A < B ? A : B;
        }
    }while (true);
}
int main ()
{
    int t;
    hugeint n, ans;
    scanf ( "%d", &t );
    while ( t -- ){
        scanf ( "%I64d", &n );
        ans = rho ( n );
        if ( ans == n ){
            printf ( "Prime\n"    );
        }else{
            printf ( "%I64d\n", ans );
        }
    }
    return 0;
```

}

Pku1061--青蛙的约会

Description

两只青蛙在网上相识了，它们聊得很开心，于是觉得很有必要见一面。它们很高兴地发现它们住在同一条纬度线上，于是它们约定各自朝西跳，直到碰面为止。可是它们出发之前忘记了一件很重要的事情，既没有问清楚对方的特征，也没有约定见面的具体位置。不过青蛙们都是很乐观的，它们觉得只要一直朝着某个方向跳下去，总能碰到对方的。但是除非这两只青蛙在同一时间跳到同一点上，不然是永远都不可能碰面的。为了帮助这两只乐观的青蛙，你被要求写一个程序来判断这两只青蛙是否能够碰面，会在什么时候碰面。
我们把这两只青蛙分别叫做青蛙 A 和青蛙 B，并且规定纬度线上东经 0 度处为原点，由东往西为正方向，单位长度 1 米，这样我们就得到了一条首尾相接的数轴。设青蛙 A 的出发点坐标是 x，青蛙 B 的出发点坐标是 y。青蛙 A 一次能跳 m 米，青蛙 B 一次能跳 n 米，两只青蛙跳一次所花费的时间相同。纬度线总长 L 米。现在要你求出它们跳了几次以后才会碰面。

Input

输入只包括一行 5 个整数 x，y，m，n，L，其中 x≠y < 2000000000，0 < m、n < 2000000000，0 < L < 2100000000。

Output

输出碰面所需要的跳跃次数，如果永远不可能碰面则输出一行"Impossible"

Sample Input

```
1 2 3 4 5
```

Sample Output

```
4
```

```
#include<iostream>
using namespace std;
long long ext_euclid( long long a, long long b, long long &x, long long &y)
{
        long long t, d;
```

```
        if(b==0){
                x = 1;
                y = 0;
                return a;
        }
        d = ext_euclid(b, a%b, x, y);
        t = x;
        x = y;
        y = t - a/b*y;
        return d;
}
int main()
{
         long long m, n, l, x, y, M, ax, ay;
        cin>>x>>y>>m>>n>>l;
        M = ext_euclid(n-m, l, ax, ay);
        if((x-y)%M || m==n)
                cout<<"Impossible"<<endl;
        else{
                 long long s = l/M;
                 long long c = x-y;
                ax = ax*c/M;
                ax = (ax%s + s)% s;
                cout<<ax<<endl;
        }
        return 0;
}
```

2.3 组合数学初步

2.3.1 鸽笼原理和Ramsey原理

2.3.2 排列组合和容斥原理

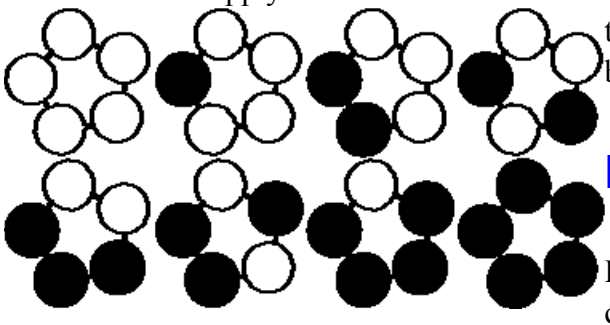
2.3.3 群论与polya定理

Pku2409--Let it Bead

Description

"Let it Bead" company is located upstairs at 700 Cannery Row in Monterey, CA. As you can deduce from the company name, their business is beads. Their PR department found out that customers are interested in buying colored bracelets. However, over 90 percent of the target audience insists that the bracelets be unique. (Just imagine what happened if two women showed up at the same party wearing identical bracelets!) It's a good thing that bracelets can have different lengths and need not be made of beads of one color. Help the boss estimating maximum profit by calculating how many different bracelets can be produced.

A bracelet is a ring-like sequence of s beads each of which can have one of c distinct colors. The ring is closed, i.e. has no beginning or end, and has no direction. Assume an unlimited supply of beads of each color. For different values of s and c, calculate the number of different bracelets that can be made.



Input

Every line of the input file defines a test case and contains two integers: the number of available colors c followed by the length of the bracelets s. Input is terminated by c=s=0. Otherwise, both are positive, and, due to technical difficulties in the bracelet-fabrication-machine, cs<=32, i.e. their product does not exceed 32.

Output

For each test case output on a single line the number of unique bracelets. The figure below shows the 8 different bracelets that can be made with 2 colors and 5 beads.

Sample Input

```
1 1
2 1
2 2
5 1
2 5
2 6
6 2
0 0
```

Sample Output

```
1
2
3
5
8
13
21
#include <iostream>
#include <cstring>
using namespace std;
const int maxn = 35;
int n, m;
long long ans;
int arr[maxn], brr[maxn], flag[maxn];
void input(void) {
	ans = 0;
	for (int i = 1; i <= n; i++)
		arr[i] = i;
	return ;
}
void chang2(void) {
	int tmp;
	tmp = arr[n];
	for (int i = n-1; i >= 1; i--)
		arr[i+1] = arr[i];
	arr[1] = tmp;
	return ;
}
int cul(void) {
	int tmp = 0, x;
	memset(flag, 0, sizeof(flag));
	for (int i = 1; i <= n; i++) {
		if (flag[i] == 0) {
			flag[i] = 1;
			tmp++;
			x = i;
			while (flag[brr[x]] == 0) {
				flag[brr[x]] = 1;
				x = brr[x];
			}
		}
	}
	return tmp;
}
```

```
void out(int x) {
	long long tmp = 1;
	for (int i = 1; i <= x; i++)
		tmp *= m;
	ans += tmp;
	return ;
}
void chang1(void) {
	int t, tmp;
	t = (n+1)/2;
	if (n%2) {
		for (int i = 2; i <= t; i++) {
			tmp = brr[n+2-i];
			brr[n+2-i] = brr[i];
			brr[i] = tmp;
		}
	} else {
		for (int i = 1; i <= t; i++) {
			tmp = brr[n+1-i];
			brr[n+1-i] = brr[i];
			brr[i] = tmp;
		}
	}
	return ;
}
/*
void ok(void) {
	for (int i = 1; i <= n; i++)
		cout << brr[i] << " ";
	cout << endl;
	return ;
}*/
void opt(void) {
	int tmp;
	for (int i = 1; i <= n; i++) {
		memcpy(brr, arr, sizeof(arr));
		tmp = cul();
		out(tmp);
		chang1();
		tmp = cul();
		out(tmp);
		chang2();
	}
}
```

```
int main(void) {
	while (cin >> m >> n, n&&m) {
		input();
		opt();
		cout << ans/(n*2) << endl;
	}
	return 0;
}
```

Pku1286--Necklace of Beads

Description

Beads of red, blue or green colors are connected together into a circular necklace of n beads (n < 24). If the repetitions that are produced by rotation around the center of the circular necklace or reflection to the axis of symmetry are all neglected, how many different forms of the necklace are there?

Input

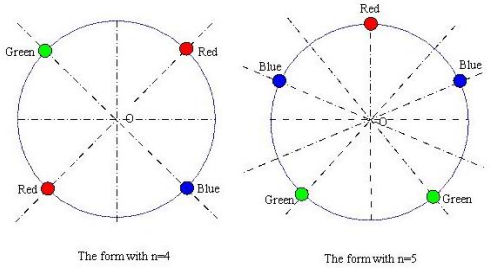
The input has several lines, and each line contains the input data n.
-1 denotes the end of the input file.

Output

The output should contain the output data: Number of different forms, in each line correspondent to the input data.

Sample Input

```
4
5
```




```
-1
```

Sample Output

```
21
39
```

Pku2154--Color

Description

Beads of N colors are connected together into a circular necklace of N beads (N<=1000000000). Your job is to calculate how many different kinds of the necklace can be produced. You should know that the necklace might not use up all the N colors, and the repetitions that are produced by rotation around the center of the circular necklace are all neglected.
You only need to output the answer module a given number P.

Input

The first line of the input is an integer X (X <= 3500) representing the number of test cases. The following X lines each contains two numbers N and P (1 <= N <= 1000000000, 1 <= P <= 30000), representing a test case.

Output

For each test case, output one line containing the answer.

Sample Input

```
5
1 30000
2 30000
3 30000
4 30000
5 30000
```

Sample Output

```
1
3
```

```
11
70
629
```

分析：这题和 1286，2409 都有点类似，不同之处在于，只考虑旋转，不考虑翻转；因此相对前面两个题
目应该说是更简单，但一看数据范围，就不是这么回事了，1286 和 2409 完全可以直接循环处理，但这题目
n 最大达 100000000，显然会 TLE，故需寻求更佳的解决方案。可以用欧拉函数进行优化，或者用 Mobius 反
演定理进行优化。下面讲解一下用欧拉优化的方法：
旋转：顺时针旋转 i 格的置换中，循环的个数为 gcd（i，n），每个循环的长度为 n/gcd(i,n)。
如果枚举旋转的格数 i，复杂度显然较高。有没有好方法呢？可以不枚举 i，反过来枚举 L。
由于 L|N,枚举了 L，再计算有多少个 i 使得 0<=i<=n-1 并且 L＝gcd(i, n)。即 gcd（i,n）＝n/L。
不妨设 a＝n/L=gcd(i, n),
不妨设 i＝a*t 则当且仅当 gcd(L,t)=1 时
Gcd(i,n)=gcd(a*L,a*t)=a。
因为 0<=i<n,所以 0<=t<n/a=L.
所以满足这个条件的 t 的个数为 Euler(L).
现在结果已经很明显了。Ans＝Σ(Euler(L)*(n.^(L-1)))%p。(L 为符合上面假设条件的所有数).
复杂度分析：线性筛选素数，线性筛选欧拉函数因子，枚举 L，这些都是在线性的复杂度内完成的。

```
#include <iostream>
#include <memory>
#include <cstdlib>
#define M      35005
#define mr 35000
using namespace std;
/*X (X <= 3500),N and P (1 <= N <= 1000000000, 1 <= P <= 30000)*/
int n, p, l=1, pn=0, test;
int prim[10001], notp[M];/*n/logn<15000*/
//prim[]存储素数，notp[]用来标记素数，pn 用来计数。
int prime()
{//线性筛选素数
    int i, j;
    memset(notp, 0, sizeof(notp));
    for(i=2; i<mr; i++) {
        if(notp[i]==0) {
            prim[pn++]=i;
        }
        for(j=0; prim[j]*i<mr&&j<pn; j++)      {
            notp[i*prim[j]]=1;
            if(i%prim[j]==0)
            break;//线性的关键体现
        }
```

```
    }
    return 0;
}
int phi(int n)
{//线性欧拉函数，Caclulate,Euler(n)%p.
//欧拉公式 Euler(n)=n(1-1/p)(1-1/q)(……).
    int tem=n, i;
    for(i=0; i<pn&&prim[i]*prim[i]<=tem; i++) {
        if(tem%prim[i]==0)      {
            n-=(n/prim[i]);
            do {
                tem/=prim[i];
            }while(tem%prim[i]==0);
        }
    }
    if(tem!=1)
    n-=n/tem;
    return n%p;
}
int exp_mod(int m)
{//Calculate n.^m%p.
    int tem, s, ans;
    tem=m;
    ans=1;
    s=n%p;
    while(tem>0) {
        if(tem%2==1)
        ans=(ans*s)%p;
        tem/=2;
        s=(s*s)%p;
    }
    return ans;
}
int main()
{
    prime();
    scanf("%d",&test);
    while(test--) {
        int i, ans=0;
        scanf("%d%d",&n, &p);
        for(l=1; l*l<=n; l++)
        if(l*l==n)//枚举循环长度 l，找出相应的 i 的个数：gcd(i,n)=n/l.
        ans=(ans+phi(l)*exp_mod(l-1))%p;
        else
```

```
            if(n%l==0)//有长度为 l 的循环，就会有长度为 n/l 的循环。
                ans=(ans+phi(l)*exp_mod(n/l-1)+phi(n/l)*exp_mod(l-1))%p;
             printf("%d\n",ans);
    }
    return 0;
}
```

Pku1037--A decorative fence

Description

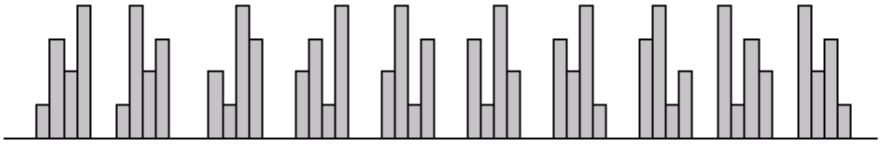
Richard just finished building his new house. Now the only thing the house misses is a cute little wooden fence. He had no idea how to make a wooden fence, so he decided to order one. Somehow he got his hands on the ACME Fence Catalogue 2002, the ultimate resource on cute little wooden fences. After reading its preface he already knew, what makes a little wooden fence cute.

A wooden fence consists of N wooden planks, placed vertically in a row next to each other. A fence looks cute if and only if the following conditions are met:

- The planks have different lengths, namely 1, 2, . . . , N plank length units.
- Each plank with two neighbors is either larger than each of its neighbors or smaller than each of them. (Note that this makes the top of the fence alternately rise and fall.)

It follows, that we may uniquely describe each cute fence with N planks as a permutation a1, . . . , aN of the numbers 1, . . . ,N such that (any i; 1 < i < N) (ai − ai−1)*(ai − ai+1) > 0 and vice versa, each such permutation describes a cute fence.

It is obvious, that there are many di erent cute wooden fences made of N planks. To bring some order into their catalogue, the sales manager of ACME decided to order them in the following way: Fence A (represented by the permutation a1, . . . , aN) is in the catalogue before fence B (represented by b1, . . . , bN) if and only if there exists such i, that (any j < i) aj = bj and (ai < bi). (Also to decide, which of the two fences is earlier in the catalogue, take their corresponding permutations, find the first place on which they differ and compare the values on this place.) All the cute fences with N planks are numbered (starting from 1) in the order they appear in the catalogue. This number is called their catalogue number.



All cute fences made of $N = 4$ planks, ordered by their catalogue numbers.

After carefully examining all the cute little wooden fences, Richard decided to order some of them. For each of them he noted the number of its planks and its catalogue number. Later, as he met his friends, he wanted to show them the fences he ordered, but he lost the catalogue somewhere. The only thing he has got are his notes. Please help him find out, how will his fences look like.

Input

The first line of the input file contains the number K (1 <= K <= 100) of input data sets. K lines follow, each of them describes one input data set.
Each of the following K lines contains two integers N and C (1 <= N <= 20), separated by a space. N is the number of planks in the fence, C is the catalogue number of the fence.
You may assume, that the total number of cute little wooden fences with 20 planks fits into a 64-bit signed integer variable (long long in C/C++, int64 in FreePascal).
You may also assume that the input is correct, in particular that C is at least 1 and it doesn 抉 exceed the number of cute fences with N planks.

Output

For each input data set output one line, describing the C-th fence with N planks in the catalogue. More precisely, if the fence is described by the permutation a1, . . . , aN, then the corresponding line of the output file should contain the numbers ai (in the correct order), separated by single spaces.

Sample Input

```
2
2 1
3 3
```

Sample Output

```
1 2
2 3 1
```

```
#include <iostream>
#include<stdio.h>
#include<string.h>
using namespace std;
const int MAX = 21;
long long W[MAX][MAX];
long long M[MAX][MAX];
int used[MAX];              //用于输出时判断木条的使用情况
long long c;
int n;
void Input (){
        scanf("%d%lld", &n, &c);
```

```
}
//利用递归进行初始化
__int64 getDown (int x, int n);
__int64 getUp (int x, int n);

__int64 getDown (int x, int n)
{
	if ( W[x][n] != -1 )
		return W[x][n];
	int i;
	__int64 ans;
	ans = 0;
	for (i=1; i<x; i++)
		ans += getUp (i, n-1);
	return ( W[x][n] = ans );
}
__int64 getUp (int x, int n)
{
	if ( M[x][n] != -1 )
		return M[x][n];
	return ( M[x][n] = getDown(n-x+1, n) );
}
void Init ()
{
	int i, j;
	memset(W, -1, sizeof(W));
	memset(M, -1, sizeof(M));
	for (i=1; i<=MAX; i++)
		W[1][i] = 0;
	W[1][1] = 1;
	M[1][1] = 1;
	W[1][2] = 0;
	M[1][2] = 1;
	W[2][2] = 1;
	M[2][2] = 0;
	for (i=3; i<=MAX; i++)
	{
		for (j=1; j<=i; j++)
		{
			getDown(j, i);
			getUp(j, i);
		}
	}
}
```

```
//子问题中，1～k 范围内第 x 根木条实际的位置
void Output (int x, int k)
{
	int i;
	do
	{
		for (i=1; i<=k; i++)
		{
			//若 1～k 中有以前已确定的木条，并且 x 木条比已确定的木
			//条长度要长，那么 x 木条的长度应加 1
			if ( used[i] && x >= i )
			{
				x++;
				k++;
			}
		}
	} while ( used[x] );
	if ( k > 1 )
		printf("%d ", x);
	else
		printf("%d", x);
	used[x] = 1;
}
void Solve ()
{
	int up, i, first;
	up = 1;
	i = 1;
	first = 1;
	memset(used, 0, sizeof(used));
	while ( n > 0 )
	{
		if ( up == 1 ){
			if ( c > M[i][n] ){
				if ( first )    //是否是在查找第 1 根木条
				{
					up = !up;
				}
				c -= M[i][n];
				i++;
			}
			else{
				Output(i, n);
				first = 0;
```

```
                                        //i = 1;
                                        up = !up;
                                        n--;
                                }
                        }else{
                                if ( c > W[i][n] ){
                                        if ( first ){
                                                up = !up;
                                        }
                                        c -= W[i][n];
                                        if ( !first ){
                                                i ++;
                                        }
                                }else{
                                        Output(i, n);
                                        first = 0;
                                        up = !up;
                                        n--;
                                        i = 1;
                                }
                        }
                }
                printf("\n");
}
int main ()
{
                int test;
                cin>>test;
                Init ();
                while ( test-- ){
                        Input ();
                        Solve ();
                }
                return 0;
}
```

2.4 图的基本知识和算法

2.5 图论与模型

Pku1275--Cashier Employment

Description

A supermarket in Tehran is open 24 hours a day every day and needs a number of cashiers to fit its need. The supermarket manager has hired you to help him, solve his problem. The problem is that the supermarket needs different number of cashiers at different times of each day (for example, a few cashiers after midnight, and many in the afternoon) to provide good service to its customers, and he wants to hire the least number of cashiers for this job.
The manager has provided you with the least number of cashiers needed for every one-hour slot of the day. This data is given as R(0), R(1), ..., R(23): R(0) represents the least number of cashiers needed from midnight to 1:00 A.M., R(1) shows this number for duration of 1:00 A.M. to 2:00 A.M., and so on. Note that these numbers are the same every day. There are N qualified applicants for this job. Each applicant i works non-stop once each 24 hours in a shift of exactly 8 hours starting from a specified hour, say ti (0 <= ti <= 23), exactly from the start of the hour mentioned. That is, if the ith applicant is hired, he/she will work starting from ti o'clock sharp for 8 hours. Cashiers do not replace one another and work exactly as scheduled, and there are enough cash registers and counters for those who are hired.
You are to write a program to read the R(i) 's for i=0..23 and ti 's for i=1..N that are all, non-negative integer numbers and compute the least number of cashiers needed to be employed to meet the mentioned constraints. Note that there can be more cashiers than the least number needed for a specific slot.

Input

The first line of input is the number of test cases for this problem (at most 20). Each test case starts with 24 integer numbers representing the R(0), R(1), ..., R(23) in one line (R(i) can be at most 1000). Then there is N, number of applicants in another line (0 <= N <= 1000), after which come N lines each containing one ti (0 <= ti <= 23). There are no blank lines between test cases.

Output

For each test case, the output should be written in one line, which is the least number of cashiers needed.
If there is no solution for the test case, you should write No Solution for that case.

Sample Input

```
1
1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
5
0
23
22
1
10
```

Sample Output

```
1
```

```
#include <iostream>
using namespace std;
const int M = 25;
const int INF = 21000000;
int r[M];  //每小时需要的出纳员的数目
int t[M];  //每小时应征的申请者的数目
int s[M];  //0～i 时刻雇用的出纳员的数目
int main(void) {
	int ts;
	cin >> ts;
	while (ts--) {
		for (int i = 1; i <= 24; i++) {
			cin >> r[i];
			t[i] = 0;
		}
		int m, a;
		cin >> m;
		for (int i = 0; i < m; i++) {
			cin >> a;
			t[a+1]++;
		}
		int sum;
		for (sum = 0; sum <= m; sum++) {//枚举 sum
			s[0] = 0;
			for (int i = 1; i <= 24; i++)
```

```
                s[i] = INF;
            int count = 0;
            bool flag = 1;
            while (flag&&count < M) {   //进行 m-1 松弛操作
                flag = 0;
                count++;
                for (int i = 1; i <=    24; i++)
                    if (s[i] < s[i-1]) {
                        s[i-1] = s[i];
                        flag = 1;
                    }
                for (int i = 24; i > 0; i--)
                    if (s[i-1] + t[i] < s[i]) {
                        s[i] = s[i-1]+t[i];
                        flag = 1;
                    }
                if (s[24] - sum < s[0]) {
                    s[0] = s[24] - sum;
                    flag = 1;
                }
                for (int i = 1; i <= 24; i++) {
                    if (i > 8 && s[i] < s[i-8] + r[i]) {
                        s[i-8] = s[i] - r[i];
                        flag = 1;
                    }
                    if (i <= 8 && s[i] < s[i+16] + r[i] - sum) {
                        s[i+16] = s[i] - r[i] + sum;
                        flag = 1;
                    }
                }
            }
            if (count < M && s[24] >= sum)
                break;
        }
        if (sum > m)
            cout << "No Solution" << endl;
        else
            cout << sum << endl;
    }
    return 0;
}
```

Pku1273--Drainage Ditches

Description

Every time it rains on Farmer John's fields, a pond forms over Bessie's favorite clover patch. This means that the clover is covered by water for awhile and takes quite a long time to regrow. Thus, Farmer John has built a set of drainage ditches so that Bessie's clover patch is never covered in water. Instead, the water is drained to a nearby stream. Being an ace engineer, Farmer John has also installed regulators at the beginning of each ditch, so he can control at what rate water flows into that ditch.
Farmer John knows not only how many gallons of water each ditch can transport per minute but also the exact layout of the ditches, which feed out of the pond and into each other and stream in a potentially complex network.
Given all this information, determine the maximum rate at which water can be transported out of the pond and into the stream. For any given ditch, water flows in only one direction, but there might be a way that water can flow in a circle.

Input

The input includes several cases. For each case, the first line contains two space-separated integers, N (0 <= N <= 200) and M (2 <= M <= 200). N is the number of ditches that Farmer John has dug. M is the number of intersections points for those ditches. Intersection 1 is the pond. Intersection point M is the stream. Each of the following N lines contains three integers, Si, Ei, and Ci. Si and Ei (1 <= Si, Ei <= M) designate the intersections between which this ditch flows. Water will flow through this ditch from Si to Ei. Ci (0 <= Ci <= 10,000,000) is the maximum rate at which water will flow through the ditch.

Output

For each case, output a single integer, the maximum rate at which water may emptied from the pond.

Sample Input

```
5 4
1 2 40
1 4 20
2 4 20
2 3 30
```

```
3 4 10
```

Sample Output

```
50
```

```
#include <cstdio>
#include <cstring>
#define MAX 210
int map[MAX][MAX], dist[MAX], pre[MAX];
int m, n, sum, Min;
void Dijstra() {
	int s[MAX], i, j, u, min;
	for (i = 1; i <= n; i++) {
		dist[i] = map[1][i];
		s[i] = 0;
		if (i != 1 && dist[i] > 0) {
			pre[i] = 1;
		} else {
			pre[i] = -1;
		}
	}
	s[1] = 1;
	for (i = 2; i <= n; i++) {
		min = 0;
		for (j =1 ; j <= n; j++)
			if (s[j] == 0 && dist[j] > min) {
				u = j;
				min = dist[j];
			}
		if (min == 0) return;
		s[u] = 1;
		for (j = 1; j <= n; j++) {
			if (s[j] == 0 && map[u][j] > 0 && dist[u] + map[u][j] > dist[j]) {
			dist[j] = dist[u] + map[u][j];
			pre[j] = u;
			}
		}
	}
}
void solve() {
	int t, s;
	Min = 2000000000; s = n; t = pre[n];
	while (t != -1) {
		if (map[t][s] < Min)Min = map[t][s];
```

```
            s = t; t = pre[t];
        }
}
void change() {
        int s, t;
        s = n; t = pre[n];
        while (t != -1) {
            map[t][s] -= Min;
            map[s][t] += Min;
            s = t;
            t = pre[t];
        }
}
int main() {
        int a, b, c;
        while (scanf("%d%d", &m, &n) != EOF) {
            memset(map, 0, sizeof(map));
            while (m -- ) {
                scanf ("%d%d%d", &a, &b, &c);
                map[a][b] += c;
            }
            sum = 0;
            while (1) {
                Dijstra();
                if (pre[n] == -1)      break;
                solve();
                sum += Min;
                change();
            }
            printf("%d\n",sum);
        }
        return 0;
}
```