SECOND JUNIOR BALKAN OLYMPIAD IN INFORMATICS

July 8 – 13, 2008

Shumen, Bulgaria

TASKS AND SOLUTIONS

Day 1

Task 1. TOWERS OF COINS

Statement

Asen and Boyan are playing the following game. They choose two different positive integers *K* and *L,* and start the game with a tower of *N* coins. Asen always plays first, Boyan – second, after that – Asen again, then Boyan, and so on. The boy in turn can take 1, *K* or *L* coins from the tower. The winner is the boy, who takes the last coin (or coins). After a long, long playing, Asen realizes that there are cases in which he could win, no matter how Boyan plays. And in all other cases Boyan being careful can win, no matter how Asen plays. So, before the start of the game Asen is eager to know what game case they have. Write a program **`coins`** which help Asen to predict the game result for given *K, L* and *N*.

Input

The input describes *m* games.

The first line of the standard input contains the integers *K*, *L* and *m*, $1 < K < L < 10$, $3 < m < 50$. The second line contains *m* integers $N_1, N_2, \ldots, N_m$, $1 \le N_i \le 1\,000\,000$, $i = 1, 2, \ldots, m$, representing the number of coins in each of the *m* towers.

Output

The standard output contains a string of length *m* composed of letters *A* and *B*. If Asen wins the i^{th} game (no matter how the opponent plays), the i^{th} letter of the string has to be *A*. When Boyan wins the i^{th} game (no matter how Asen plays), the i^{th} letter of the string has to be *B*.

Example

Input

```
2 3 5
3 12 113 25714 88888
```

Output

```
ABAAB
```

Solution

As it was mentioned in the statement of the problem in some games Asen could win regardless how Boyan is playing and the opposite – in some games, if Boyan is playing optimally, Asen could not win. Let us denote with N any possible number of coins in the tower – N is called *position* of the game. It is important to realize that **for given *L* and *K*, each position is either *winning* or *loosing***. The goal of the task is to decide if the position is winning or loosing for each of m given positions $N_1, N_2,\ldots, N_m$.

Let us denote with N_{max} the maximum of the numbers $N_1, N_2,\ldots, N_m$. The value N_{max} is found while reading the numbers $N_1, N_2,\ldots, N_m$ in the array $in[]$.

So we have to decide for each integer $i = 0, 1, ..., N_{max}$ if the corresponding position is winning or loosing. For that purpose we will fill an array $win[]$ with zeros and ones, where the value 0 of $win[i]$ designates the loosing position i and the value 1 designates the winning position i. Finaly, the program has to print for each number N_i the letter A when $win[N_i] = 1$ and letter B when $win[N_i] = 0$.

How to find $win[i]$? Obviously $win[0] = 0$. Suppose that we calculate $win[0]$, $win[1]$, ..., $win[i-1]$. When the player is in position i he has tree possibilities – take 1 coin, take K coins or take L coins, i.e. to transform the game to position $i-1$, $i-K$ or $i-L$. If at least one of the positions $i-1$, $i-K$ or $i-L$ is loosing (i.e. $win[i-1]$ && $win[i-K]$ && $win[i-L] == 0$) then the player in position i has a move that leads the game in a loosing position and so the position i is winning. So $win[i] = 1$. If each of the positions $i-1$, $i-K$ or $i-L$ is winning (i.e. $win[i-1]$ && $win[i-K]$ && $win[i-L] == 1$) then the player in position i has no move that leads the game in a loosing position and so the position i is loosing. So $win[i] = 0$.

Notice that it is possible to have $i-K$ or $i-L$ less than 0. This means that the corresponding move does not exist and has no influence to the results. Thus each negative position can be considered as winning!

Realization

```
#include <stdio.h>
char win[1000001];
int in[11];

int main()
{
    int K,L,M,N,Nmax=0,i;
    int one,ka,el;
    scanf("%d %d %d",&K,&L,&M);
    // reading input and finding Nmax
    for(i=1;i<=M;i++)
    {
       scanf("%d",&in[i]);
       if(Nmax<in[i]) Nmax=in[i];
    }

    win[0]=0; // position 0 is loosing
    for(i=1;i<=Nmax;i++)
    {
       one=win[i-1];
       if(i-K>=0) ka=win[i-K]; else ka=1;
       if(i-L>=0) el=win[i-L]; else el=1;
       if(one&&ka&&el) win[i]=0; else win[i]=1;
    }

    for(i=1;i<=M;i++)
       if(win[in[i]]) printf("A");
       else printf("B");
    printf("\n");
}
```

Task 2. CLOSEST

Statement

Consider two n-digit positive decimal integers A and B with no leading zeroes. We need to find the two closest to A n-digit numbers (the first one – greater or equal to A, the other – strictly less than A), with decimal writings containing all the digits of B in some order.

For example if $A = 3022$ and $B = 1232$, using B's digits we can obtain the following 4-digit numbers: 1223, 1232, 1322, 2123, 2132, 2213, 2231, 2312, 2321, 3122, 3212 and 3221. The least number greater or equal to A

obtained by *B*'s digits is 3122, and the biggest one, strictly less then *A* is 2321. If $A = 1232$, and $B = 3022$, the possible numbers are 2023, 2032, 2203, 2230, 2302, 2320, 3022, 3202 and 3220. The least number greater or equal to *A* obtained by *B*'s digits is 2023, and there is no number less than *A*.

Write a program **`closest`** to find these "closest to *A*" numbers for given *A* and *B*, or to determine that one of them does not exist.

Input

Two lines are read from the standard input, each of them containing an *n*-digit positive integer with no leading zeroes, with *A* read from the first, and *B* read from the second line ($1 \le n \le 60$).

Output

Write to the standard output:

- Line 1: the least *n*-digit number with no leading zeroes, not less than *A*, containing all the digits of *B* in some order. If such number does not exist, the output should be 0.
- Line 2: the biggest *n*-digit number with no leading zeroes, less than *A*, containing all the digits of *B* in some order. If such number does not exist, the output should be 0.

Examples

Input	**Input**
`3075`	`3000203`
`6604`	`4562454`
Output	**Output**
`4066`	`4244556`
`0`	`2655444`

Solution

The exhausting search through all *n*-digit numbers after *A* (and then before *A*, too) and the check, if they consist only of *B*'s digits, will be good enough just for the first test (and for another one if organized with arrays/strings). The exhaustion of all *B*'s permutations and determining the closest neighbors of *A* will suffice for three more tests. Partial considerations can solve some tests, but not all of them.

For a full solution, it can be pointed out that the algorithm for determining a non-bigger neighbor will not be principally different from the one finding the non-less ("upper") neighbor. If we subtract 1 from *A*, the second part of the problem becomes analogical to the first one: searching for the non-bigger neighbor of *A*-1. We only have to add a rule to eliminate pseudo

solutions with a leading zero. So we can concentrate on the first problem – determination of the "upper" neighbor.
We apply the following algorithm:

- If all digits of *B* are less than the first *A*'s digit, there's no solution;
- If there is a digit in *B* equal to the first digit in *A*, we can try to reduce the problem to a less dimension: we "rub out" this digit from *B*, putting it as first in the result, "delete" the first digit of *A* and solve the same problem for $(n - 1)$-digit numbers. If there is a solution for the reduced problem, this is the final result, preceded by the found digit.
- If we don't get a solution applying the previous step, the solution is found by taking the least digit in *B* bigger than the considered digit in *A* for the next digit in the result, and sorting in ascending order the digits of *B* which are not used yet. If such digit is not found, there is no solution.

Realization

```
#include <iostream>
using namespace std;
int const N=64;
int a[N],r[N],st[10],n;
void strToArr(const char *s,int *a,int *n)
{*n=0;
 while (*s) a[(*n)++]=*s++-'0';
}
void makeStat(int n,const int *a,int *st)
{memset(st,0,10*sizeof(int));
 for (int i=0;i<n;i++) st[a[i]]++;
}
void show(int n,int *a)
{for (int i=0;i<n;i++) cout<<a[i];
 cout<<endl;
}
bool findHi(int p)
{int d,i;
 if (p==n) return true;
 for (d=a[p]; d<=9; d++) if (st[d]) break;
 if (d>9) return false;
 if (d==a[p]){r[p]=d;
              st[d]--;
              if (findHi(p+1)) return true;
              st[d]++;
```

```
               for (d=a[p]+1; d<=9; d++) if (st[d])
break;
               if (d>9) return false;
              }
 r[p++]=d;
 st[d]--;
 for (d=0;d<=9;d++)
  for (i=0;i<st[d];i++) r[p++]=d;
 return true;
}
bool findLo(int p)
{int d,i;
 if (p==n) return true;
 for (d=a[p]; d>=0; d--) if (st[d]) break;
 if (d<0) return false;
 if (!p && !d){do d++; while (!st[d]); if (d>a[p])
return false;}
 if (d==a[p]){r[p]=d;
               st[d]--;
               if (findLo(p+1)) return true;
               st[d]++;
               for (d=a[p]-1; d>=0; d--) if (st[d])
break;
               if (d<0) return false;
              }
 r[p++]=d;
 st[d]--;
 for (d=9;d>=0;d--)
  for (i=0;i<st[d];i++) r[p++]=d;
 return true;
}
bool Dec(int n, int *a)
{int i=n-1;
 while (!a[i]) a[i--]=9;
 a[i]--;
 return (bool)*a;
}
int main(void)
{char buf[64];
 int b[64];
 cin>>buf;
 strToArr(buf,a,&n);
 cin>>buf;
```

```
 strToArr(buf,b,&n);
 makeStat(n,b,st);
 if (findHi(0)) show(n,r); else cout<<"0\n";
 makeStat(n,b,st);
 if (!Dec(n,a)) cout<<"0\n";
 else if (findLo(0)) show(n,r);
      else cout<<"0\n";
 return 0;
}
```

Task 3. RECTANGLES

Statement

Given are n rectangles, numbered from 1 to n. We place them tightly on the axis OX, from left to right, according to rectangles' numbers. Each rectangle stays on the axis OX either by its shorter or by its longer side (see the picture below). Compute the length of the upper envelop line, i.e. perimeter's length of the obtained figure minus the length of the left, right and bottom straight line segments of the picture. Write program **`rec`** to find the maximum possible length of the upper envelop line.

Input

On the first line of the standard input, the value of n is written. On each of the next n lines, two integers are given – a_i and b_i – the side lengths of the i^{th} rectangle.

Output

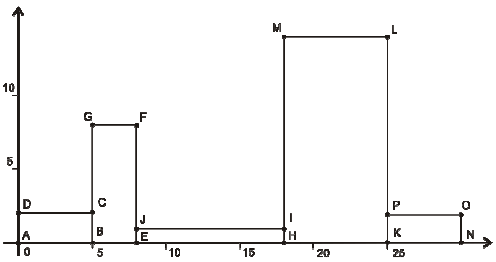
On a line of the standard output, your program should write the result as a positive integer.

Constraints: $0 < n < 1000;\ 0 < a_i < b_i < 1000$, for each $i = 1,\ 2,\ ...,\ n$.

Example

Input

```
5
2 5
3 8
1 10
7 14
2 5
```



Output

```
68
```

Explanation: A configuration, that yields the maximum length of the upper envelop line, is presented on the picture.
The upper envelop line consists of segments *DC, CG*, *GF*, *FJ*, *JI*, *IM*, *ML*, *LP*, and *PO*. The total length is 5 + 6 + 3 + 7 + 10 + 13 + 7 + 12 + 5 = 68.

Solution

Dynamic programming approach: Let us denote by $f(i)$ the maximal length of the upper envelop line for the task involving the first *i* rectangles only, the last of them being placed on the base line *OX* with its *short* side. Analogously, let us denote by $g(i)$ the maximal length of the upper envelop line for the task involving the first *i* rectangles with the last of them placed on its *long* side. We have $f(1) = a_1$ and $g(1) = b_1$. *If we have already computed values* $f(i)$ *and* $g(i)$ *for some i,* $1 \le i < n$, *we may compute:*

$$f(i+1) = a_i + \max\left\{f(i) + |b_i - b_{i+1}|,\ g(i) + |a_i - b_{i+1}|\right\},$$
$$g(i+1) = b_i + \max\left\{f(i) + |b_i - a_{i+1}|,\ g(i) + |a_i - a_{i+1}|\right\}.$$

Iteratively, we obtain values of pairs $\big(f(1), g(1)\big)$, $\big(f(2), g(2)\big)$, ..., $\big(f(n), g(n)\big)$. The solution is equal to the largest of $f(n)$ and $g(n)$.

In the program implementation below, the values of $f(i)$ and $g(i)$ are stored as array's elements *t*[*i*][0] and *t*[*i*][1].

Realization

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

const int N=1024;
int n;
int a[N], b[N], t[N][2];

int main()
{
 cin >> n;
 for(int i=1;i<=n;i++) cin >> a[i] >> b[i];
 t[1][0]=a[1]; t[1][1]=b[1];
 for(int i=2;i<=n;i++)
 {
  t[i][0]=a[i]+
   max(t[i-1][0]+abs(b[i-1]-b[i]),t[i-
1][1]+abs(a[i-1]-b[i]));
```

```
  t[i][1]=b[i]+
   max(t[i-1][0]+abs(b[i-1]-a[i]),t[i-
1][1]+abs(a[i-1]-a[i]));
 }
 cout << max(t[n][0], t[n][1]) << endl;
}
```

Day 2

Task 1. JUMPS

Statement

A bunny has to pass n meters with jumps of lengths 3, 2 or 1 meters. In how many different ways this can be done, if the lengths of successive jumps form a non-increasing sequence?
Write program **`jumps`**, which computes the number we are looking for.

Input

The value of n should be entered from the standard input ($1 \leq n \leq 10^9$).

Output

The program should write to the standard output one integer, equal to the remainder of the found number, divided by 1000000.

Remark: In 50% of test cases, $n \leq 10^5$.

Example

Input

```
6
```

Output

```
7
```

Explanation: The number of different ways is 7, and its remainder modulo 1000000 is also 7. The different sequences of jumps are:

1) 3+3
2) 3+2+1
3) 3+1+1+1
4) 2+2+2
5) 2+2+1+1
6) 2+1+1+1+1
7) 1+1+1+1+1+1

Solution A

Denote by x, y, z the number of jumps of length 3, 2, and 1, respectively.
In this way we have to compute the number of different solution in non-negative integers of the equation $3x + 2y + z = n$**.** It is clear that $x \le n/3$, and for fixed value of x the possible values of y are 0, 1, …, $(n - 3x)/2$.

```
int cnt=0;
for(int x=0; x<= n/3; x++)
  for(int y=0; y<=(n-3*x)/2; y++)
  { cnt++;
    if(cnt>=M) cnt = cnt - M;
  }
cout << cnt << endl;
```

Solution B

We may remove the inner loop in Solution A. For fixed value of x there are exactly $1+(n-3x)/2$ possibilities for y. In addition we may avoid multiplication, just denote $a = 3x$.

```
int cnt=0;
for(int a=0; a<=n; a=a+3)
{ cnt = cnt + (n-a)/2 + 1;
  if(cnt>M) cnt = cnt%M;
}
cout << cnt << endl;
```

Solution C

Using Solution A or Solution B we may do some observations.
Consider the computed numbers for $n = 1, 7, 13, 19, \ldots, 6k + 1, \ldots$

n	1	7	13	19	25
cnt	1	8 = 1 + 7	21 = 8 + 13	40 = 21 + 19	65 = 40 + 25

The conjecture we could make is that $cnt(x) = cnt(x - 6) + x$.
It turns out that the same relationship holds for any value of $x \ge 6$.
The assumption that $cnt(0) = 1$ is quite natural.

```
int x = n%6;
int cnt=0;
if(x==0) cnt=1; else cnt=x;

while(x<n)
{ x = x + 6;
  cnt = cnt + x;
  if(cnt >= M) cnt = cnt%M;
}

cout << cnt << endl;
```

Solution D

Consider the example in Solution C: $cnt(25) = 1 + 7 + 13 + 19 + 25$. More generally $cnt(6k + 1) = 1 + 7 + 13 + \ldots + (6k + 1)$. The increment of the sequence is 6, and the number of terms in the sum is $k + 1$. We may apply the Gauss' trick for computing such a sum. For example:

$$S = 1 + 7 + 13 + 19 + 25$$
$$S = 25 + 19 + 13 + 7 + 1$$

$$2S = 26 + 26 + 26 + 26 + 26 = 26 * 5$$
$$S = 26 * 5 / 2 = 65.$$

```
long long cnt = ((n%6 + n)/2%M * ((1 + n/6)%M));
if(n%6==0) cnt++;
cout << cnt%M << endl;
```

Similarly to Solution C a small correction has to be made for the case when n is divisible by 6.

Task 2. SQUARES

Statement

Let R be a rectangle with integer side lengths. The rectangle is divided into unit squares. Considering one of the diagonals, we denote by $f(R)$ the number of squares which have a common interior point with it. For example, if the side lengths of R are 2 and 4 then $f(R)=4$. Write a program **squ** to find out the number of all different rectangles R for which $f(R)=N$. Two rectangles with sides $a\times b$ and $b\times a$ are not different.

Input

In a single line of the standard input the integer N ($0 < N < 10^6$) is given.

Output

The only line of the standard output should contain an integer – the calculated number of rectangles.

Remark: In 50% of test cases, $N \le 2000$.

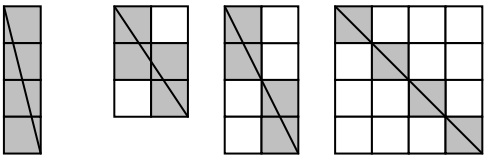
Example

Input

```
4
```

Output

```
4
```



Explanation: The different rectangles R for which $f(R)=4$ are 4: with side lengths 1 and 4, 2 and 3, 2 and 4, 4 and 4.

Solution

Let the lengths of sides of the rectangle R be a and b and $d=\gcd(a,b)$. Then $f(R)=a+b-d$. So, we search the number of integer solutions of the equation $a+b-d=N$, for which $a \le b$.

Let $a=x.d$, $b=y.d$. Then $1 \le x \le y$ and $\gcd(x,y)=1$.

Write the equation in the form $(x+y-1)d=N$.

Hence d is a divisor of N. Let $N=m.d$. Consequently $x+y=m+1$.

Now for each m, which is a divisor of N we have to find the number of integer solutions of the equation $x + y = m + 1$, satisfying $1 \le x \le y$ and $\gcd(x, y) = 1$.

Note that $\gcd(x, y) = \gcd(x, m + 1 - x) = \gcd(x, m + 1)$.

Realization

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

int nod(int a, int b)
{ int r = a%b;
  while(r > 0)
  { a = b; b = r; r = a%b; }
  return b;
}

int main()
{ int n;
  cin >> n;
  int s = 0;
  for(int m = 1; m*m <= n; m++)
  {  if (n % m == 0)
       if (m*m == n)
         {  for(int x = 1; x <= (m + 1)/2; x++)
              if (nod(x, m + 1) == 1) s++;
         }
       else
         {  for(int x = 1; x <= (m + 1)/2; x++)
              if (nod(x, m + 1) == 1) s++;
            for(int x = 1; x <= (n/m + 1)/2; x++)
              if (nod(x, n/m + 1) == 1) s++;
         }
  }
  cout << s << endl;
  return 0;
}
```

Task 3. SUMX

Statement

Consider a set of n distinct positive integers a_1, a_2, …, a_n, having values between 1 and 1000000 and an integer x. Write a program **sumx** to determine the number of pairs (a_i, a_j), where $1 \le i < j \le n$ and $a_i + a_j = x$.

Input

The first line of the standard input contains the integer n ($1 \le n \le 100000$). The second line contains n integers – the elements of the set. On the third line the integer x is given ($1 \le x \le 2000000$).

Output

The program should output on a single line of the standard output an integer – the calculated number of pairs.

Remark: In 50% of test cases, $n \le 1000$.

Example

Input

```
9
5 12 7 10 9 1 2 3 11
13
```

Output

```
3
```

Explanation: The different pairs with sum 13 are: (12, 1), (10, 3) and (2, 11).

Solution

Assume the sequence is sorted in an increasing order. Consider a_1 and a_n and let $y = a_1 + a_n$. If $y > x$, then a_n cannot be part of the solution because $a_n + a_i > x$ for all i. Therefore, we eliminate a_n from consideration and continue with a_{n-1}. If $y < x$ we eliminate a_1 according to the similar argument. If $y = x$, then the first pair is found and we continue with a_2 and a_{n-1}.

According to stated above we have to sort the sequence first.

Realization

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

```
#include <vector>
using namespace std;
int main ()
{
 int n, b, x, c=0;
 vector <int> a;
 scanf ("%d", &n);
 for (int i = 0; i < n; i++){
  scanf ("%d", &b);
  a.push_back (b);
 }
 scanf ("%d", &x);
 sort(a.begin(),a.end());
 int k=0, m=n-1;
 while (k<m){
  if (a[k]+a[m]==x){c++; k++; m--;}
  else if (a[k]+a[m]<x)k++;
  else if (a[k]+a[m]>x)m--;
 }
 printf("%d\n", c);
 return 0;
}
```