VIII НАЦИОНАЛЕН ЕСЕНЕН ТУРНИР ПО ИНФОРМАТИКА И ИНФОРМАЦИОННИ ТЕХНОЛОГИИ “ДЖОН АТАНАСОВ”

Секция “Информатика”
Условия и решения

ШУМЕН, 28-30 Ноември 2008

Организатори:

Министерство на
образованието и науката

Съюз на математиците в България

Шуменски университет
“Епископ Константин Преславски”

Школа А&Б – Шумен

Със съдействието на:

ПМГ „Нанчо Попович” – Шумен

СОУ „Сава Доброплодни” – Шумен

СОУ „Йоан Екзарх Български” – Шумен

Спонсори:

Министерство на
образованието и науката

Съюз на математиците в България

Шуменски университет
“Епископ Константин Преславски”

Школа А&Б – Шумен


```
  if(s==n)cout<<"Yes"<<" "<<s;
  else cout<<"No";
  cout<<endl;
  return 0;
}
```

Автор: Бистра Танева

Решение

Необходими величини:

char c1,c2,c3,c4,c5,c6,c7,c8 – за знаците на номера;

int p,q,r,t – за ASCII кодовете на буквите;

int x3,x4,x5,x6 – за числените стойности на цифрите;

int s – за произведението от цифрите;

int n – за сумата от ASCII кодовете на буквите, разделена целочислено на десет.

Входни данни:

Въвеждаме символите **c1**, **c2**, **c3**, **c4**, **c5**, **c6**, **c7**, **c8**.

Определяме ASCII кодовете на буквите c1, c2, c7, c8:

```
p=(char)c1; r=(char)c7;
q=(char)c2; t=(char)c8;
```

Получаваме на числените стойности на c3, c4, c5, c6:

```
x3=c3-'0';
x4=c4-'0';
x5=c5-'0';
x6=c6-'0';
```

Изчисляваме s и n:

```
s=x3*x4*x5*x6;
n=(p+q+r+t)/10;
```

Проверяваме дали **s** и **n** са равни и извеждаме съответно или **Yes** и **s**, или **No**.

Окончателната програма – решение на задачата:

```
#include<iostream>
using namespace std;
int main ()
{ int n,s=0,p,q,r,t;
  char c1,c2,c3,c4,c5,c6,c7,c8;
  int x3,x4,x5,x6;
  cin.get(c1);
  cin.get(c2);
  cin.get(c3);
  cin.get(c4);
  cin.get(c5);
  cin.get(c6);
  cin.get(c7);
  cin.get(c8);
  p=(char)c1; r=(char)c7;
  q=(char)c2; t=(char)c8;
  x3=c3-'0';
  x4=c4-'0';
  x5=c5-'0';
  x6=c6-'0';
  s=x3*x4*x5*x6;
  n=(p+q+r+t)/10;
```


Задача A1. Троични таблици

Във всяка клетка на правоъгълна таблица **S** с **m** реда и **n** стълба е записано някое от числата **0**, **1** или **2**. Две клетки в таблицата наричаме съседни, когато имат обща страна. От таблицата **S** е получена нова таблица **T** със същите размери, като елементът **T[i][j]** е равен на сбора по модул **3** на елементите от всички клетки, съседни на клетка **S[i][j]**.

Напишете програма **table**, която по дадена таблица **T** намира броя на различните таблици **S**, от които по описания начин може да бъде получена таблицата **T**.

Вход

От първия ред на стандартния вход се въвеждат две числа **m** и **n** – размерите на таблиците. Следват **m** реда с по **n** числа – елементите на таблицата **T** по редове.

Изход

Програмата трябва да изведе на един ред на стандартния изход броя, по модул **333333**, на различните таблици **S**, от които по описания начин може да бъде получена таблицата **T**.

Ограничения

$2 \le m \le 32,\ 2 \le n \le 32$

Пример

Вход	Изход
3 4 0 2 2 1 1 0 2 2 1 1 0 2	1

Решение

Означаваме размерите на таблиците **S** и **T** с m_0 и n_0 и нека $n = m_0n_0$. Елементите на таблицата **S** са неизвестни. Да ги означим по редове с $x_1, x_2, \ldots, x_n$, където $x_k = S[i][j]$, $k = (i-1)n_0+j$, $i = 1, 2, \ldots, m_0$, $j = 1, 2, \ldots, n_0$. Получаваме линейна система от **n** уравнения с **n** неизвестни. Всяко уравнение има вида:

$$S[i-1][j] + S[i+1][j] + S[i][j-1] + S[i][j+1] = T[i][j],$$

като в случай, че някой от индексите е извън таблицата, съответното събираемо отсъства.

Пример. Нека **T** и **S** са съответно следните таблици:

2	1	0
0	1	2
1	1	2

x_1	x_2	x_3
x_4	x_5	x_6
x_7	x_8	x_9

Системата е следната:

$$
\begin{aligned}
x_2 + x_4 &= 2 = T[1][1]\\
x_1 + x_3 + x_5 &= 1 = T[1][2]\\
x_2 + x_6 &= 0 = T[1][3]\\
x_1 + x_5 + x_7 &= 0 = T[2][1]\\
x_2 + x_4 + x_6 + x_8 &= 1 = T[2][2]\\
x_3 + x_5 + x_9 &= 2 = T[2][3]\\
x_4 + x_8 &= 1 = T[3][1]\\
x_5 + x_7 + x_9 &= 1 = T[3][2]\\
x_6 + x_8 &= 2 = T[3][3]
\end{aligned}
$$

В матричен вид системата изглежда така (в празните клетки има **0**):

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	T[i][j]
	1		1						2
1		1		1					1
	1				1				0
1				1		1			0
	1		1		1		1		1
		1		1				1	2
			1				1		1
				1		1		1	1
					1		1		2

Да отбележим, че това е система над троичното поле – стойностите на неизвестните са **0**, **1** или **2** и аритметичните действия се извършват по модул **3**.

Провеждаме елиминация по Гаус. Обхождаме последователно уравненията на системата (т.е. редовете на матрицата) търсейки ненулев коефициент пред някое от неизвестните. Нека **a[r][c] ≠ 0**. За всяко **i = r+1**, **r+2**, …, **n**, ако **a[i][c] ≠ 0**, умножаваме уравнение **r** по подходящ коефициент и го прибавяме към уравнение **i**, така че елементът **a[i][c]** да стане равен на **0**.

След извършването на една стъпка системата добива вида (водещият елемент **a[1][2]** е маркиран):

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	
	1		1						2
1		1		1					1
	0		2		1				1
1				1		1			0
	0		0		1		1		2
		1		1				1	2
			1				1		1
				1		1		1	1
					1		1		2


```
else de=k%10+k/10%10;
```

Получава се новото число и се извежда на стандартния изход.

```
cout<<st*100+de<<endl;
```

Окончателната програма – решение на задачата:

```
#include<iostream>
using namespace std;
int main()
{
    int k;
    cin>>k;
    int st;
    st=k/10%10+k/100;
    if(st>9)st=st%10+st/10;
    int de;
    if(k%10%2==0)de=k%10*2;
    else de=k%10+k/10%10;
    cout<<st*100+de<<endl;
    return 0;
}
```

Автор: Веселка Константинова

Задача Е3. Щастлив номер

Всеки автомобил има регистрационен номер, който се състои от поредица от осем символа. Първите два и последните два са главни латински букви, а останалите – цифри. Един номер е щастлив, ако произведението от цифрите е равно на сумата от кодовете (ASCII-кодовете) на първите две и последните две букви, разделена целочислено на десет.

Напишете програма **lucky**, която проверява дали даден номер е щастлив.

Вход

На първият ред от стандартния вход се въвежда поредица от осем знака – две главни латински букви, четири цифри и две главни латински букви.

Изход

На един ред на стандартния изход програмата трябва да изведе `Yes` и произведението от цифрите, разделени с един интервал, ако номера е щастлив и `No`, в противен случай.

Примери

Вход	Изход
AH2131AA	No
AF4171QA	Yes 28

- Играчите трябва да разпознаят дали цифрата на единиците на числото, написано на картата е четно число. Ако то е четно, тогава го удвояват. Ако е нечетно, тогава към него прибавят цифрата на десетиците на числото от картата. Полученият резултат е число, десетиците и единиците на което са втората и третата цифра на новото число.

Понеже Умко бил по-умен и почти винаги бил победител в подобни игри, помогнете на Сръчко да спечели поне в тази игра, като напишете програма **winner**.

Вход

На първия ред на стандартния вход се въвежда едно цяло число **k** – трицифреното число, записано върху изтеглената карта.

Изход

Програмата извежда на единствен ред на стандартния изход едно цяло трицифрено число, което трябва да получи победителят в играта.

Примери

Вход	Изход
100	100
387	215
624	808

Решение

Необходими величини:

int k – за трицифреното число написано на картата и за новото число;
int st – за цифрата на стотиците на новото число;
int de – за числото съдържащо десетиците и стотиците на новото число;

Входни данни:

Въвежда се числото **k**.

Пресмята се сбора от десетиците и стотиците на числото k, записано на картата, като се има предвид, че ще се получат чрез целочислено деление на **10** и целочислен остатък от деление на **10** на входното число **k**.

```
st=k/10%10+k/100;
```

Ако получения сбор не е двуцифрено число, то чрез него се задава цифрата на стотиците на търсеното ново число. Ако резултатът е двуцифрено число, тогава се събират цифрите му и този резултат е цифрата на стотиците на новото число.

```
if(st>9)st=st%10+st/10;
```

Получава се числото съдържащо цифрите на десетиците и единиците на новото число. За целта се проверява дали цифрата на единиците на началното число **k** е четна и, ако това е така, нейната стойност се удвоява. В противен случай (единиците на **k** са нечетно число) числото съдържащо десетиците и единиците на новото число се получава като сбор от единиците и десетиците на входното число:

```
if(k%10%2==0)de=k%10*2;
```


След завършването на всички стъпки получаваме:

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	
	1		1						2
1		1		1					1
	0		2		1				1
0		2		0		1			2
	0		0		1		1		2
		0		1		1		1	1
			0		0		0		0
				0		0		0	0
					0		0		0

Получиха се три реда, съставени изцяло от нули. Съответно, три от неизвестните (в случая това са x_7, x_8 и x_9) могат да получават произволни стойности, а останалите неизвестни се изразяват еднозначно чрез тях. Тъй като за всяка променлива има три възможности (**0**, **1** или **2**), то възможните комбинации за трите променливи са $3^3 = 27$. Следователно задачата има **27** различни решения.

Ако в процеса на работа се получи уравнение, в което всички коефициенти пред неизвестните са нули, а свободният член е различен от нула, системата няма решение.

Ако в получената система няма редове изцяло от нули системата има единствено решение.

Автор: Стоян Капралов

Задача А2. Огледала

През последния век фотографията се развива с бързи темпове, а поради сложната техника се налагат и сложни методи за изследване на системите от лещи. Модел фотоапарати използва автоматична фокусираща система, която се състои от **N** на брой повърхности, всяка от които е огледална от едната страна и пропускаща светлина от другата. Основна трудност настъпва с определяне под какъв ъгъл трябва да се излъчи инфрачервена светлина от определена точка **A** (близо до матрицата на цифровия фотоапарат), така че да попадне в точка **B** (която желаем да заснемем), след като се отрази точно по веднъж в произволен ред във всяка от огледалните повърхности. Плъзгането на светлинен лъч по дължината на огледална повърхност се приема за отражение.

Вход

От първия ред на стандартния вход, прочетете естественото число **N**. От следващия ред прочетете координатите на точката $\mathbf{A(A_x,A_y)}$, а от по-следващия – координатите на точката $\mathbf{B(B_x,B_y)}$. Следват **N** реда, на които е записана

информация за огледалните повърхности – по четири числа, обозначаващи координатите на началото и края (в които също може да настъпи отражение) на поредното огледало. Лявата повърхност е отразяваща (ако гледаме от първата точка към втората), а дясната е прозрачна. Няма пресичащи се огледала и също така, никоя от точките **A** и **B** не лежат на никое от огледалата.

Изход

Напишете програма **mirrors**, която извежда на първия ред на стандартния изход думата "**YES**", ако съществува решение, и "**NO**" в противен случай. Ако поне едно решение съществува, на втория ред изведете ъгъла в градуси (с точност – шест знака след десетичната запетая), под който точка **B** се вижда от точка **A**. Ъгъла се отчита в положителна посока (обратно на часовниковата стрелка) от оста **Ox** и е в интервала от **0** до **360** градуса, включително.

Ограничения

0 < N < 11

Всички огледални повърхности имат дължина, не по-малка от **0.001**.

Всички координати са реални числа с не повече от шест цифри след десетичната запетая и не по-големи от **10000** по абсолютна стойност.

Пример

Вход	Изход
2 0 0 0 5 1 0 1 2 -1 4 -1 2	YES 51.340192

Решение

За всички възможни пермутации на реда за отражение в огледалата, забелязваме, че съществуват или **0** или **1** решение за ъгъл на излъчване от точка **A**, така че лъчът да се отрази в дадения ред във всички огледала и да попадне в точка **B**. За всеки фиксиран ред, последователно премахваме поредното огледало, като заменяме източника с неговия еквивалентен (получен чрез перпендикулярно симетрично изображение спрямо поредното огледало). След премахване на всички огледала, директно можем да проверим дали при излъчване от крайния еквивалент на точка **A** към точка **B**, пресичаме последното огледало (еквивалентно на отразяване в него при реална точка **A**). Така по обратния път на огледалата, заменяме точка **B** с еквивалентната на нея - пресечната точка на лъча с поредното огледало. Ако по пътя на заменяне, винаги получаваме отражения на лъча с огледалата от пермутацията и никога не получаваме отражение на лъча между две съседни отражения, то сме получили едно валидно решение и остава да изчислим ъгъла на излъчване от точка **A**.

```
#include <cstdio>
#include <cmath>
#include <algorithm>
```


Вход	Изход
36 24 72 8 10 22	23

Решение

Решението на задачата се свежда до намиране сумата на всички оценки, минималната и максималната оценка. От сумата трябва да се извадят най-малката и най-голямата стойност, понеже отпадат най-ниската и най-високата оценка и получения резултат да се раздели на **4**.

Окончателната програма – решение на задачата:

```
#include <iostream>
using namespace std;
int main()
{  long long a,b,c,d,e,f,min,max,s;
   cin>>a>>b>>c>>d>>e>>f;
   min=a; max=a;
   s=a+b+c+d+e+f;
   if (b<min) min=b;
   if (c<min) min=c;
   if (d<min) min=d;
   if (e<min) min=e;
   if (f<min) min=f;
   if (b>max) max=b;
   if (c>max) max=c;
   if (d>max) max=d;
   if (e>max) max=e;
   if (f>max) max=f;
   s=(s-min-max)/4;
   cout<<s<<endl;
   return 0;
}
```

Автор: Кинка Кирилова

Задача Е2. Победител

Умко и Сръчко са неразделни приятели. Едно любимо тяхно забавление са игрите. Новата игра, която им подарили техните родители, се състои от карти. На всяка една от тях има написани трицифрени числа. Картите се обръщат така, че играчите да не виждат числата, разбъркват се и от купчината се изтегля произволна карта. Умко и Сръчко виждат числото написано на нея и по зададените в играта правила трябва да образуват ново трицифрено число. Играта печели този от тях, който познае числото пръв.

И така, **правилата на играта** са следните:

- Играчите събират стотиците и десетиците на числото от картата. Ако резултатът е двуцифрено число, тогава отново събират цифрите му. Получената цифра се записва като първа за новото число;

```
char a[11000],c;
int n;

int solve()
{
    int i,ans=0;
    for(i=1;i<n;i++)if(a[i-1]==c && a[i]!=c)ans++;
    ans*=2;
    if(a[0]==c)ans--;
    if(a[n-1]==c)ans++;
    return ans;
}

int main()
{
    int i;
    char ch;
    cin>>n>>c;
    for(i=0;i<n;i++)cin>>a[i];
       cout<<solve()<<endl;
    return 0;
}
```

Автор: Петър Петров

Задача E1. Акробат

В някои спортове, като гимнастика, акробатика, ски-скокове и др., изпълнението на всеки спортист се оценява от няколко съдии. След това от всички оценки се премахват най-ниската и най-високата, а от останалите оценки се пресмята окончателната оценка за изпълнение по следния начин – сумата на оценките се разделя на техния брой. Съдиите гласуват така, че крайната оценка е винаги цяло число. Ако максимална или минимална оценка са поставили няколко съдии, то отпада само една от тях.

Напишете програма **acrobat**, която въвежда оценките **a**, **b**, **c**, **d**, **e**, **f** (**a**, **b**, **c**, **d**, **e**, **f** са цели числа), които поставят шест съдии, намира и отпечатва крайната оценка, която получава всеки спортист.

Вход

От първия ред на стандартния вход се въвеждат стойностите на числата **a**, **b**, **c**, **d**, **e**, **f**, разделени с по един интервал.

Изход

На един ред на стандартния изход програмата трябва да изведе едно цяло число, равно на търсената оценка за изпълнение.

Ограничения

1 ≤ a, **b**, **c**, **d**, **e**, **f ≤ 1000000000000**

Пример


```
using namespace std;
const int        MAX_N = 16;
const double     eps   = 1e-9;
const double     mypi  = acos(-1.0);
// точка/радиус вектор в равнината
struct point {
    double x, y;
    point () {}
    point (double _x, double _y) : x(_x), y(_y) {}

    point operator+(point b)
{ return point(x+b.x,y+b.y); }
    point operator-(point b)
{ return point(x-b.x,y-b.y); }
    point operator*(double k)
{ return point(x*k,y*k); }
    point rot ()
{ return point(-y,x); }
    double len2 ()
{ return x*x + y*y; }
};
// двойка точки, дефиниращи насочена отсечка/вектор
struct segm {
    point a, b;
    segm () {}
    segm (point _a, point _b) : a(_a), b(_b) {}
};
int n;     // брой огледала
point A, B;// начална и крайна точка
segm M[MAX_N];// описание на огледалата
                    int order[MAX_N];  // текущ ред на разглеждане на
                //пермутацията от огледала
void input ()
{
    int i;

    scanf ("%d", &n);
    scanf ("%lf%lf", &A.x, &A.y);
    scanf ("%lf%lf", &B.x, &B.y);

    for (i=0; i<n; i++)
          scanf ("%lf%lf%lf%lf", &M[i].a.x, &M[i].a.y, &M[i].b.x,
&M[i].b.y);
    // по-бързо в средния случай
    random_shuffle (M, M+n);
}
// ориентирано лице на тройка точки в равнината
double orient (point &a, point &b, point &c)
{ return (b.x-a.x)*(c.y-b.y) - (c.x-b.x)*(b.y-a.y); }
// перпендикулярна симетрия на точката p относно правата, определена от
отсечката s
```

```
point orthoProj (point &p, segm &s)
{
    point v = s.b-s.a;
    return p - v.rot() * (2.0 * fabs(orient(p,s.a,s.b)) / v.len2());
}
// дали две отсечки се пресичат (пресечната точка ще бъде записана в p)
// ако e>0.0 -- допиране на отсечките ще се отчита за пресичане на
отсечките
// ако e<0.0 -- допиране на отсечките няма да бъде достатъчно за пресичане
на отсечките
bool segm_intersect (segm g, segm h, point &p, double e)
{
    if ( orient(g.a,g.b,h.a)*orient(g.a,g.b,h.b) > e ) return 0;
    if ( orient(h.a,h.b,g.a)*orient(h.a,h.b,g.b) > e ) return 0;

    double S1 = orient(g.a,h.a,h.b);
    double S2 = orient(h.a,g.b,h.b);
    double coef = S1 / (S1+S2);

    p = g.a + (g.b-g.a)*coef;
    return 1;
}
// дали участъка g от последователни отражения на лъча, се пресича с друго
огледало и се отразява (което не е позволено)
bool another_mirror_intersect (segm g)
{
    int i;
    static point p;

    for (i=0; i<n; i++)
        if (segm_intersect(g,M[i],p,-eps))
            if (orient(M[i].a,M[i].b,g.a) > -eps)
                return 1;

    return 0;
}

// дали е възможно да се достигне точка B за фиксирана пермутация на
отражения в огледалата
bool try_perm ()
{
    int i;
    static point proj[MAX_N];

    // прав ход - от A към B
    // proj[i] -> i-тото перпендикулярно симетрично изображение на A в
зададения ред на отражения в огледалата
    proj[0] = A;
    for (i=0; i<n; i++) {
        // ако това е неотразяващата страна
```



```
        return 0;
    }
```

Автор: Бисерка Йовчева

Задача D3. Шоколад

За коледните и новогодишните празници в магазините започнали да се продават нови луксозни видове шоколади. Те представлявали само един ред с няколко блокчета, като по интересното било, че на всяко блокче била написана една главна латинска буква (**'A' – 'Z'**). Някои от буквите се повтаряли, други не. Ако искаме да изядем от шоколада първо блокчетата с буква **'A'**, ние трябва да го разчупим така, че от целия шоколад да отделим само тези блокчета, в които има буквата **'A'**. Напишете програма **choko**, която по зададена буква да намира минимален брой разчупвания, така че от шоколада да се отделят всички блокчета със зададената буква.

Вход

На първия ред на стандартния вход, разделени с един интервал се въвеждат едно число N – броят на блокчетата на шоколада и една буква. На следващия ред се въвеждат N латински букви. Първата буква е написана на първото блокче, втората буква на второто блокче и т.н.

Изход

На стандартния изход изведете едно число – минималния брои разчупвания на шоколада.

Ограничения:

0 < N < 10000

Примери

Вход	Изход
10 A BAACAADZAZ	6
8 A ABAAAAAA	2

Решение

За да намерим минималния брой разчупвания, се интересуваме не само от блокчетата със съответната буква, а и от последователността им. Нека за яснота да наречем последователност от еднакви букви *платформа*. Например в низа **"AAABBBAABBA"** имаме три платформи образувани от буквата **'A' – "AAA", "AA", "A"** и две платформи с буквата **'B' – "BBB"** и **"BB"**. За решението на задачата е необходимо да се намери броя на всички платформи, които са образувани от съответната буква и да се съобрази, че всяка платформа се отделя с две разчупвания с изключение на тези, които са в краищата. За тях е необходимо само по едно.

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

Въвеждането може да се реализира със следната функция:

```
void read()
{
    int x,y,d,i;
    cin>>n>>m;
    for(i=1;i<=m;i++)
    {
        cin>>x>>y>>d;
        if(x<y)
        {
            a[x]+=d-d/2;
            a[y]+=d/2;
        }
        else
        {
            a[y]+=d-d/2;
            a[x]+=d/2;
        }
    }
}
```

Когато масивът **a** е запълнен, се намира максималния му елемент. Това може да стане със следната функция:

```
long long max_km()
    {
        long long mx=a[1];
        int i;
        for(i=1;i<=n;i++)
        if(a[i]>mx)mx=a[i];
        return mx;
    }
```

Когато максималното възможно количество километри за асфалтиране е намерено, се преминава през целия масив **a** и се отпечатва индекса на всеки елемент на масива, равен на намерения максимум.

На следващия ред се отпечатва намерения максимум. Това може да се опише в главната функция по следния начин:

```
    int main()
    {
        int i, l=0;
        long long mx;
        read();
        mx=max_km();
        for(i=1;i<=n;i++)
        if(a[i]==mx)
        {
            if(l)cout<<' ';
            else l=1;
            cout<<i;
        }
        cout<<endl<<mx<<endl;
```



```
                if ( orient(M[ order[i] ].a, M[ order[i] ].b, proj[i]) < -eps )
return 0;
                // генериране на еквивалентния източник зад огледалото
                proj[i+1] = orthoProj(proj[i], M[ order[i] ]);
            }

            // обратен ход
            // proj[i] -> точката на (i-1)-тото огледало, в която лъчът се
отразява
            proj[n+1] = B;
            for (i=n; i>0; i--) {
                // ако лъчът не се отразява в огледалото (а трябва)
                if(!segm_intersect(segm(proj[i],proj[i+1]),M[order[i-
1]],proj[i],eps))
                    return 0;
                // ако преждевременно лъчът пресича друго огледало по време на
пътя си между две поредни отражения
               if(another_mirror_intersect(segm(proj[i],proj[i+1]))) return 0;
            }
            // лъчът се разлага на общо n+1 отсечки и това е последната/първата -
най-близка до източника A
            if ( another_mirror_intersect( segm(A,proj[1]) ) )
        return 0;

            // ако ъгълът живее в III-ти или IV-ти квадрант
        double ang = 180.0/mypi * atan2(proj[1].y-A.y, proj[1].x-A.x);
            printf ("YES\n");
            printf ("%.6lf\n", ang + (ang<0.0 ? 360.0 : 0.0));
            return 1;
        }
        // опитва всички пермутации за реда на огражение в огледалата
        void solve ()
        {
            int i;
            for (i=0; i<n; i++) order[i] = i;
            do {
                if (try_perm ())
                    return;
            }
        while ( next_permutation(order,order+n) );
            printf ("NO\n");
        }
        int main ()
        {
            input ();
            solve ();
            return 0;
        }
```

Автор: Петър Иванов

Задача А3. Низ

Разглеждаме низове, съставени от малки латински букви и лексикографската наредба между такива низове. Подниз, който съдържа всички букви на даден низ от определено място до края, наричаме суфикс на дадения низ. Един низ наричаме прост, ако той е строго по-малък от всеки свой (различен от самия низ) суфикс. Всички еднобуквени низове са прости. Напишете програма **string**, която за даден низ, намира негово представяне като конкатенация от негови последователни прости поднизове, за които е изпълнено, че всеки от поднизовете (освен първият) е по-малък или равен на предишния.

Вход

От първия ред на стандартния вход се въвежда дадения низ.

Изход

Програмата извежда на един ред в стандартния изход броя на намерените низове и съответните им дължини, отделени с точно един интервал.

Ограничение

Дължината на входният низ не надминава **1000000** знака.

Пример

Вход	Изход
baabaaabba	4 1 3 5 1

Пояснение

Входният низ се представя като конкатенация на следните 4 низа:
b
aab
aaabb
a

Решение

Прилагаме алчен алгоритъм (известен като алгоритъм на Дювал), който по време на работата си поддържа представяне на входния низ **S** като конкатенация на три низа $S=S_1S_2S_3$. Описаната в условието на задачата декомпозия вече е намерена в низа S_1, низът S_2 е междинна част, която обработваме, а S_3 необработена част. На всяка стъпка вземаме първия знак от S_3 и го дописваме към S_2. Ако тогава за някакъв префикс на S_2, декомпозицията му е станала известна, тази част преминава към S_1. По време на работата за S_2 поддържане специално представяне като конкатенация от вида **WW…WP**, където **W** е прост низ, а **P** е някакъв префикс на **W**.

Означаваме с **i** указател, който показва началото на S_2. В алгоритъма, с този указател се извършва цикъл, докато **i < N**, където **N** е дължината на **S**. Вътре в този цикъл подържаме два указателя **j** (към началото на S_3) и **k** (към текущия знак в S_2), и сравняваме **S[j]** и **S[k]**, за да определим какво действие да извършим и как да променяме стойностите на указателите. Пълното описание на възможните 3 случая се вижда от кода на програмата:

Областната управа решила да подпомогне селището, на което ще се падне да асфалтира най-много километри. За целта ви възлагат да напишете програма **roads**, която по данните от проведеното измерване определя селището, което трябва да се подпомогне.

Вход

От първия ред на стандартния вход се въвеждат две цели числа **N** и **M**, съответно броя на селищата и броя на всички пътища в областта. Следват **M** реда, на всеки, от които са зададени три **цели** числа, описващи поредния път между две селища, като първото от тях е номера на първото селище, второто е номера на второто селище, а третото е дължината на пътя в километри. Двата края на всеки път винаги са различни селища.

Изход

Програмата извежда на първия ред на стандартния изход селището, което ще трябва да асфалтира най-много километри път, а на втория ред количеството километри, които трябва да асфалтира. Ако има няколко селища с един и същ най-голям брой километри, се извеждат всички във възходящ ред, разделени с интервал.

Ограничения

$2 \le N \le M \le 100000$

дължина на пътя между две селища е не повече от **500000** км.

Пример

Вход	Изход
5 8 1 3 8 4 1 5 4 5 19 2 4 11 3 2 18 1 5 10 5 3 9 2 1 4	3 5 18

Решение

За да се намери количеството километри, които ще асфалтира всяко селище, се създава един масив от цели числа, в който елемента **a[i]** съдържа километрите, които трябва да асфалтира града с номер **i**. Тук трябва да се съобрази, че може да се получи число, което не се събира в стандартния тип **int** и затова е необходимо елементите на масива да са от тип **long long**.

```
long long a[100001];
```

Масива **a** може да се запълни още при четенето на входните данни. При въвеждането на всяка тройка числа **x,y** и **d,** се определя по-голямото от **x** и **y** и на елемента на масива **a**, който му съответства се присвоява **d/2**, а на елемента на масива **a**, който съответства на по-малкото число, се присвоява **d–d/2**.

Алгоритъмът за едно число (в случая **x**) се прилага за всички цели числа от интервала **[a;b]** .

Отпечатва се получената стойност br.

Окончателната програма – решение на задачата:

```
#include <iostream>
using namespace std;
int main()
{
    int a, b, p1, p2, br=0;
   int i, x, c;
   cin>>a>>b>>p1>>p2;
   for (i=a; i<=b; i++)
   {
       x=i;
       while (x!=0)
             {
              c=x%10;
              if ( c%p1!=0 && c%p2!=0) br++;
              x=x/10;
             }
   }
cout<<br<<endl;
return 0;
}
```

Автор: Галя Неделчева

Задача D2. Асфалтови пътища

В една област решили да асфалтират пътищата. За целта се наложило да измерят дължините на всички съществуващи пътища. За да запишат резултатите от измерването по-съкратено, номерирали градовете с цели числа от **1** до **N (N<10000)**. След това всички резултати били записани по следния начин: за всеки път между две селища се задавали три числа – първото селище, второто селище и дължината на пътя между тях. При това се приема, че пътя между две селища е единствен, независимо от това кое от тях е избрано за първо и кое за второ. За да може всяко селище да даде своя принос при асфалтирането, било взето решение, пътят между две селища да се раздели на две и всяко кметство да асфалтира своята половина. Освен това, тъй като на кметовете не им се смятало с дробни числа, за свое улеснение те решили, че ако между две селища дължината на пътя е нечетно число, то селището с по-малък номер ще асфалтира по-дългата част, получена при целочисленото деление на две на дължината на пътя.

Например, ако пътя между селище с номер 3 и селище с номер 2 има дължина 9 км, селище с номер 2 ще асфалтира 5 км, а селище с номер 3 ще асфалтира 4 км.

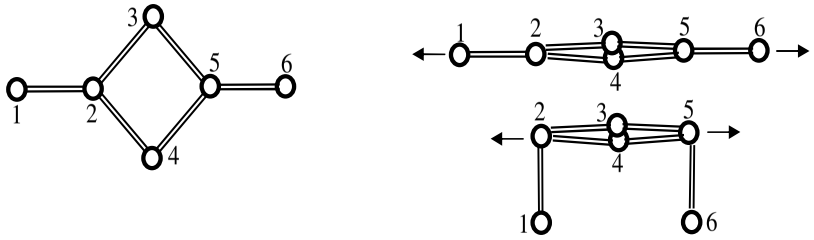

```
#include<stdio.h>
#include<string.h>

char s[1000001]; int  b[1000001]; int c=0;

void Duval()
{int n = strlen(s); int i=0;
 bool flag=false;
 while (i < n)
 {
   int j=i+1, k=i;
   while (j < n && s[k] <= s[j])
    {
     if (s[k] < s[j]) k = i;
     else ++k;
     ++j;
    }
    while (i <= k)
    {
//   if(flag) printf(" ");
//   for(int p=0; p< j-k; p++) printf("%c", s[i+p]);
     i += j - k;
//   flag=true;
     b[c]=j-k; c++;
    }
 }
}
int main()
{ scanf("%s",s); Duval();
//  printf("\n");
  printf("%d",c);
  for(int i=0;i<c;i++) printf(" %d",b[i]);
  printf("\n");
}
```

Автор: Емил Келеведжиев

Задача B1. Мрежа от въженца

Мрежа е изплетена от **N** пръстена, номерирани с числата от **1** до **N**, и **M** въженца с еднакви дължини така, че не се разпада на части. Станчо и Пешо избират два пръстена с номера **A** и **B**. Станчо хваща пръстена **A**, а Пешо – пръстена **B**, и двамата опъват мрежата в противоположни посоки докато може. В резултат, някои от пръстените (или всички), застават на една линия с **A** и **B**. Останалите пръстени

(или нито един), увисват. На фигурата е показан пример на мрежа от въженца с **N = 6** и **M = 6**.



Ако Станчо и Пешо опънат мрежата за пръстените **1** и **6**, тогава всички пръстени ще застанат в линия и нито един няма да увисне (на фигурата вдясно горе). Ако опънат мрежата за пръстените **2** и **5**, тогава пръстените **3** и **4** ще застанат в линия с **2** и **5**, а **1** и **6** ще увиснат (на фигурата вдясно долу).

Напишете програма **rings**, която по зададена мрежа и номерата **A** и **B** на два различни пръстена, да намира колко пръстена ще увиснат, когато Станчо и Пешо опънат мрежата в **A** и **B**.

Вход

На първия ред на стандартния вход ще бъдат зададени числата **N** и **M**. На всеки от следващите **M** реда ще бъдат зададени номерата на два пръстена, които са свързани с въженце. На всеки от последните два реда ще бъде зададена по една двойка номера на пръстени **A** и **B**.

Изход

На единствения ред на стандартния изход програмата трябва да изведе две цели числа, разделени с един интервал – броят на пръстените, които ще увиснат когато Станчо и Пешо опънат мрежата във всяка от двете зададени двойки пръстени. Броят висящи пръстени за първата двойка трябва да бъде изведен на първо място, а за втората двойка – на второ.

Ограничения

5 < N < 1000
1 ≤ A ≤ N
1 ≤ B ≤ N

Пример

Вход	Изход
6 6 1 2 2 3 2 4	0 2

Задача D1. Цифри

Пешо Хакера си открил следната игра: намислял си две цели числа **a** и **b** и две цифри p_1 и p_2. След това записвал всички цели числа от интервала **[a;b]** и задрасквал цифрите им, които се делят на p_1 или на p_2. После преброявал колко цифри са останали незадраскани.

Например, при **a = 15**, **b = 23**, p_1 **= 2** и p_2 **= 3**, в интервала **[15;23]** има общо 8 цифри, които не се делят на 2 или на 3: **15 1~~6~~ 17 1~~8~~ 1~~9~~ ~~20~~ ~~2~~1 ~~22~~ ~~23~~**.

Напишете програма **digit**, която помага на г-н Хакера за решаване на тази задача.

Вход

На първия ред на стандартния вход се въвеждат **a**, **b**, p_1 и p_2, разделени с интервал.

Изход

Програмата извежда на един ред на стандартния изход едно цяло число – броя на останалите незадраскани цифри.

Ограничения

1 ≤ a ≤ b ≤ 1000
p_1 **> 1**
p_2 **> 1**

ПРИМЕР

Вход	Изход
94 106 4 5	20

Решение:

Необходими величини:

int a,b; – за границите на интервала;

int p1,p2; – за намислените две цифри;

int br; – за броя незадрасканите цифри;

int c; – работна променлива, в която се съсържа текуща цифра от числото;

Входни данни:

Въвеждат се числата a, b, p1 и p2.

Пресмята се боя на незадрасканите цифри е едно цяло число (в случая означено с x)

```
while (x!=0)
{
    c=x%10;
    if (c%p1!=0 && c%p2!=0) ) br++;
    x=x/10;
}
```

```
int b[1024],pointer;
void gen_perm()
{
    int i,p=0,cn=0;
    for(i=0;i<j;i++)
    {
        while(cn!=koeficienti[i])
        {
            if(a[p]>=0)cn++;
            p++;
        }
    //  printf("%d ",a[p-1]);
        b[pointer++]=a[p-1];
        a[p-1]=-1;
        p=0;
        cn=0;
    }
    //printf("\n");
}
 int main()
{
    int i,f,os,ans;
    scanf("%d %d %d",&n,&k,&m);
    for(i=0;i<n;i++)a[i]=i+1;
    ans=k;
    ans--;f=n-1;
    while(f)
    {
        koeficienti[j++]=(ans/fac[f])+1;
        ans%=fac[f];
        f--;
    }
    koeficienti[j++]=1;
    gen_perm();
    for(j=k;j<=m;j++)
    {
        for(i=0;i<pointer-1;i++)printf("%d ",b[i]);
        printf("%d\n",b[i]);
        next_permutation(b,b+pointer);
    }
    return 0;
}
```

Автори: Галина Момчева

Петър Петров

3 5 4 5 5 6 1 6 2 5	

Решение

Да разгледаме мрежата като краен неориентиран граф **G**. Нека **X** и **Y** са два от върховете на **G**. Да означим с **ρ(X,Y)** дължината на един най-къс път в **G** от **X** до **Y**. Ще наричаме **ρ(X,Y)** ***разстояние*** между върховете **X** и **Y**. Операцията, описана в задачата, ще наричаме ***разтягане*** на **G**. Не е трудно да се съобрази, че е в сила следното:

Твърдение. *Нека графът е **G** е разтегнат чрез върховете си **A** и **B**. Върхът **X** ще бъде висящ при това разтягане тогава и само тогава, когато*

$$\rho(A,X) + \rho(X,B) > \rho(A,B).$$

Елементарен подход за решаване на задачата е, да намерим разстоянията между всички двойки върхове на графа, с алгоритъма на Флойд например, и след това да проверим за всеки от върховете условието от твърдението. Проверките на условието очевидно ще изискват $O(2N)$ операции, независимо от това как намираме разстоянията. Затова сложността на този алгоритъм ще бъде доминирана от сложността на алгоритъма на Флойд – $O(N^3)$.

Всъщност, за решаването на задачата не са нужни разстоянията между всеки два върха на графа – достатъчно е за всеки връх **X** да намерим разстоянията само до двата върха, чрез които е разтегнат графа. Ако направим това с обхождане на графа в ширина, за всеки от **N**-те върха ще са необходими по $O(M)$ операции или сложността на алгоритъма ще падне до $O(N.M)$. Забележете, че с едно обхождане в ширина можем да намерим разстоянията от **X** до всеки от върховете на двете двойки, за които разпъваме мрежата и не трябва да се прави отделно обхождане за всяка от двойките. Нещо повече, за да има този алгоритъм шанс да реши повече тестови примери, обхождането в ширина трябва да бъде прекратено в момента, в който четирите върха, чрез които разтягаме графа, са обходени.

Сега да определим нужните за решаването на задачата разстояния по друг начин: достатъчно е да намерим разстоянията от двата върха, чрез които е разтегнат графа до всеки от върховете **X**. Затова ще са ни достатъчни само четири обхождания в ширина – по едно за всеки от върховете, чрез които ще разтягаме графа. Като прибавим и проверките от твърдението получаваме $O(4.M) + O(2.N)$ операции. Тъй като графът е свързан, броят на ребрата му не може да бъде по-малък от **N − 1**, затова полученият алгоритъм ще бъде със сложност $O(M)$.

В заключение остава да добавим само, че за да бъде ефектът от използването на обхождане в ширина пълен, представянето на графа трябва да бъде във вида ***списъци на съседите***.

Окончателната програма – решение на задачата:

```
#include <stdio.h>
#define MAXN 1001
typedef struct
{  int N,M;
   int T[MAXN+1][MAXN+1]; } LN_Graph;
typedef struct
{  int e[MAXN+1],begin,end; }int_Queue;
LN_Graph G; int TA[MAXN],TB[MAXN]; int A,B;
/* Обхождане в ширина за намиране в D
   разстоянието от r да всеки връх  */
int BFS(int* D,int r)
{  int x,y,i;  int_Queue Q;
   for(i=1;i<=G.N;i++) D[i]=-1;
   Q.begin=0;Q.end=-1;
   Q.e[++Q.end]=r;D[r]=0;
   while(!(Q.begin>Q.end))
   {  x=Q.e[Q.begin++];
      for(i=1;i<=G.T[x][0];i++)
      {  y=G.T[x][i];
         if(D[y]== -1)
         {  D[y]=D[x]+1;Q.e[++Q.end]=y; }
      }
    }
}
/* Въвеждане на граф в представяне
   със списъци на съседите   */
void input_LN()
{  int i,v,w;
   scanf("%d %d",&G.N,&G.M);
   for(i=1;i<=G.N;i++) G.T[i][0]=0;
   for(i=1;i<=G.M;i++)
   {  scanf("%d %d", &v,&w);
      G.T[v][++G.T[v][0]]=w;
      G.T[w][++G.T[w][0]]=v;
   }
}
int main()
{
    int i,j,pend=0;
    input_LN();
    for(j=1;j<=2;j++)
    {  scanf("%d %d",&A,&B);
       BFS(TA,A); BFS(TB,B);
       for(i=1;i<=G.N;i++)
```


Решение

Най-лесното решение на тази задача би било да се генерират в лексикографски ред всички пермутации на първите **K** числа, като се отпечатат само тези, които имат пореден номер от интервала **[N,M]**. При това успешно може да се използва функцията **next_permutation** от стандартната библиотека **algorithm**. Подобно решение ще получи **40%** от точките.

Ако вземем предвид ограничението, че разликата между **N** и **M** не надвишава **1000000**, става ясно, че е по-добре да се генерират само пермутациите, които имат пореден номер от интервала **[N,M]**. За целта се налага да се получи **N**-тата пермутация. Използва се алгоритъма за декодиране на пермутации, който може да се илюстрира със следния пример:

Нека да се налага да се намери пермутация с номер **4**, получена от първите **3** числа. Тази пермутация е: **2 3 1**, защото:

123
132
213
231
312
321

За този случай алгоритъмът е следния:

Избира се числото (в случая горе **4**). От него се изважда **1**. Получава се **3**. Числото **3** може да се представи по следния начин:

3= 1.2!+1.1!+0.0!

Тук пред числата **2!, 1!** и **0!** коефициентите са **1 1 0**. Към всеки един от тях де прибавя **1**, при което се получават **2 2 1**.

Търсената пермутация се получава от тези числа по следния начин:

За първо число се поставя **2**–то по ред от **1, 2, 3** – това е **2**

За второ число се поставя **2**-то по ред от останалите числа (в случая **1, 3**) – това е **3**

3-тото число е първото число в редицата – т.е. **1**. Четвъртата пермутация е **2 3 1**.

След като се получи **N**-тата пермутация, останалите до **M** –тата се генерират. Най-ефективно това става с функцията **next_permutation** от стандартната библиотека **algorithm**.

Ето програма, която решава задачата:

```
#include <cstdio>
#include <algorithm>
using namespace std;
int a[1024],koeficienti[1024],n,br,k,j,m;
int
fac[13]={1,1,2,6,24,120,720,5040,40320,362880,3628800,399168
00,479001600};
```

Задача C3. Банка

Достъп до сейфа на една банка има само управителят ѝ. Сейфът се отваря с **K**-числов код, в който са включени всички числа от **1** до **K**, записани в някакъв ред. Числата не се повтарят. Спазвайки политиката за сигурност на фирмата управителят не си е записал кода и след командировка в чужбина изведнъж осъзнава, че го е забравил. Като бивш състезател по информатика си спомня само, че кодът е с номер, задаващо число от интервала **[N,M]** и разликата на тези две числа не е по-голяма от **1000000**.Номер на код е поредният номер на кода, ако те са подредени в нарастващ (лексикографски) ред.

Пример

За **K=3**

номера	Кодове
1	1 2 3
2	1 3 2
3	2 1 3
4	2 3 1
5	3 1 2
6	3 2 1

За да не стане объркване при записа на едноцифрени и многоцифрени числа, между отделните числа има интервал.

Напишете програма **banka**, която по дадени **N** и **M** извежда всички кодове с номера от **N** до **M**, които управителят на банката ще опита, за да отвори сейфа.

Вход

От стандартния вход се въвеждат:

- на първия ред число **K**;
- на вторият ред **N** и **M**, отделени с интервал.

Изход

На стандартния изход се извеждат на отделни редове кодовете с номера от зададения интервал.

Ограничения

2 < K < 13
0 < N < M < 400000000
0 < M – N ≤ 1000000

Пример

Вход	Изход
3 2 4	1 3 2 2 1 3 2 3 1


```
            if(TA[i]+TB[i]!= TA[B]) pend++;
          if(j==1) {printf("%d ",pend); pend=0;}
          else printf("%d\n",pend);
      }
}
```

Автор: Красимир Манев

Задача B2. Отсечки и прави

В равнината са дадени **N** различни отсечки. Напишете програма **lines**, която намира най-много колко от дадените отсечки може да пресече вертикална или хоризонтална права.

Вход

На първия ред на стандартния вход е записано естественото число **N**. Следват **N** реда, като всеки от тях съдържа по четири числа, разделени с по един интервал – абсцисата и ординатата на единия край и абсцисата и ординатата на другия край на поредната отсечка.

Изход

На един ред на стандартния изход програмата трябва да изведе търсения брой отсечки.

Ограничения

0 < N < 300000. Абсцисите и ординатите на краищата на дадените отсечки са цели числа от интервала $[-10^8;10^8]$.

Пример

Вход	Изход
4 -2 -2 2 2 2 1 2 4 1 5 3 5 -4 0 0 -4	3

Решение

Да разгледаме следната по-проста задача:

*Върху числова ос са дадени **N** отсечки, краищата на които са точки с целочислени координати, като някои от отсечките може да съвпадат. Възможно е също двата края на една отсечка да съвпадат (такава „отсечка” всъщност е точка). Да се намери най-много колко от дадените отсечки имат обща точка.*

Да означим отсечките с $\mathbf{A_0B_0}$, $\mathbf{A_1B_1}$, …, $\mathbf{A_{N-1}B_{N-1}}$, като координатите на краищата им са съответно $\mathbf{a_0}$ и $\mathbf{b_0}$ ($\mathbf{a_0 \le b_0}$), $\mathbf{a_1}$ и $\mathbf{b_1}$ ($\mathbf{a_1 \le b_1}$), …, $\mathbf{a_{N-1}}$ и $\mathbf{b_{N-1}}$ ($\mathbf{a_{N-1} \le b_{N-1}}$). Да разгледаме отсечките $\mathbf{A'_0B'_0}$, $\mathbf{A'_1B'_1}$, …, $\mathbf{A'_{N-1}B'_{N-1}}$ с координати на краищата им съответно $\mathbf{a'_0 = 2a_0}$ и $\mathbf{b'_0 = 2b_0+1}$, $\mathbf{a'_1 = 2a_1}$ и $\mathbf{b'_1 = 2b_1+1}$, …, $\mathbf{a'_{N-1} = 2a_{N-1}}$ и $\mathbf{b'_{N-1} = 2b_{N-1}+1}$. Ясно е, че отговорите на разглежданата задача за двете групи

отсечки съвпадат. Да подредим в растящ ред числата $a'_0, b'_0, a'_1, b'_1, \ldots, a'_{N-1}, b'_{N-1}$. Да означим с s_i разликата на броя на четните и броя на нечетните членове на сортираната редица с номера от 0 до i за i = 0, 1, …, 2N−1. Най-големият брой отсечки от дадените, имащи обща точка, е равен на най-голямата стойност на s_i.

Да се върнем към основната задача.

Нека са дадени отсечките P_0Q_0, P_1Q_1 …, $P_{N-1}Q_{N-1}$ с краища $P_0(x_0, y_0)$, $Q_0(x_1, y_1)$, $P_1(x_2, y_2)$, $Q_1(x_3, y_3)$, …, $P_{N-1}(x_{2N-2}, y_{2N-2})$, $Q_{N-1}(x_{2N-1}, y_{2N-1})$. За да решим задачата е достатъчно да решим два пъти предходната задача – веднъж за ортогоналните проекции на дадените отсечки върху абсцисната ос и втори път – за ортогоналните проекции на дадените отсечки върху ординатната ос, след което да изведем по-големия от двата получени резултата.

Окончателната програма – решение на задачата:

```
#include<cstdio>
#include<cstdlib>
using namespace std;

const int MAX2N = 600002;

int x[MAX2N], y[MAX2N];

int compare(const void *p, const void *q)
{ return *(int *)p - *(int *)q;
}

int main()
{  int x0, x1, y0, y1;
   int N;
   scanf("%d", &N);
   for(int i=0; i<N; i++)
     {  scanf("%d %d %d %d",&x0, &y0, &x1, &y1);
        if (x0 < x1)
          { x[2*i] = 2*x0; x[2*i+1] = 2*x1 + 1;
          }
        else
          { x[2*i] = 2*x1; x[2*i+1] = 2*x0 +1;
          }
        if (y0 < y1)
          { y[2*i] = 2*y0; y[2*i+1] = 2*y1 + 1;
          }
        else
          { y[2*i] = 2*y1; y[2*i+1] = 2*y0 +1;
          }
     }
   qsort(x, 2*N, sizeof(int *), compare);
```



```
                { level[v] = level[p]+1;
                  q.push(v);
                }
            }
          if (p1 >= p2)
            { v = 10000*(p1-p2) + 200*p2 + p3;
              if (level[v] == 0)
                { level[v] = level[p]+1;
                  q.push(v);
                }
            }
          if (p1 <= p3)
            { v = 20000*p1 + 100*p2 + p3-p1;
              if (level[v] == 0)
                { level[v] = level[p]+1;
                  q.push(v);
                }
            }
          if (p1 >= p3)
            { v = 10000*(p1-p3) + 100*p2 + 2*p3;
              if (level[v] == 0)
                { level[v] = level[p]+1;
                  q.push(v);
                }
            }
          if (p2 <= p3)
            { v = 10000*p1 + 200*p2 + p3-p2;
              if (level[v] == 0)
                { level[v] = level[p]+1;
                  q.push(v);
                }
            }
          if (p2 >= p3)
            { v = 10000*p1 + 100*(p2-p3) + 2*p3;
              if (level[v] == 0)
                { level[v] = level[p]+1;
                  q.push(v);
                }
            }
        } while (q.size() > 0 && level[s3] == 0);
}

int main()
{    cin >> a >> b >> c;
     s = 10000*a + 100*b +c;
     s3 = a+b+c;
     bfs(s);
     cout << level[s3]-1 << endl;
     return 0;
}
```

Автор: Младен Манев

Решение

За решаване на задачата може да използваме ориентиран граф. Всеки връх на графа съответства на едно разположение на камъчетата в трите купчини. Нека в трите купчини има съответно p_1, p_2 и p_3 камъчета. На това разположение на камъчетата съпоставяме връх, номериран с числото $10000p_1+100p_2+p_3$. Тъй като общият брой на камъчетата е по-малък от 100, по зададен номер на връх на графа може еднозначно да възстановим разположението на камъчетата в трите купчини. В графа от връх с номер **p** към връх с номер **v** има ребро, когато от купчините, съответстващи на върха **p** могат да се получат купчините, съответстващи на върха **v** по зададените правила в условието на задачата. За да намерим търсения най-малък брой ходове е достатъчно да намерим дължините на най-късите по брой ребра пътища от върха с номер **10000a+100b+c** съответно до върховете с номера **10000(a+b+c)**, **100(a+b+c)** и **a+b+c** и от тези три дължини да изберем най-малката. За намиране на дължините на търсените пътища използваме обхождане на графа в ширина, започвайки от връх с номер **10000a+100b+c**. В заключение ще отбележим, че ако камъчетата могат да се съберат в някоя от купчините с определен брой ходове, то те могат да се съберат и във всяка една от останалите две купчини със същия брой ходове. Това лесно следва от факта, че преди една купчина да остане без камъчета, в нея има толкова камъчета, колкото в една от останалите две купчини. Следователно за решаването на задачата е достатъчно да се намери дължината на най-късия път от връх с номер **10000a+100b+c** до връх с номер **a+b+c**.

Окончателната програма – решение на задачата:

```
#include<iostream>
#include<queue>

using namespace std;

int level[1000000];
int a, b, c;
int s, s3;

void bfs(int u)
{    queue<int> q;
     int p, p1, p2, p3, v;
     level[u] = 1;
     q.push(u);
     do
       {  p = q.front();
          q.pop();
          p1 = p/10000;
          p2 = (p%10000)/100;
          p3 = p%100;
          if (p1 <= p2)
            { v = 20000*p1 + 100*(p2-p1) + p3;
              if (level[v] == 0)
```



```
     qsort(y, 2*N, sizeof(int *), compare);
     int sx = 0, sy = 0;
     int maxi = 0;
     for(int i=0; i < 2*N; i++)
       {  if (x[i]%2)
            sx--;
          else
            { sx++;
              if (maxi < sx) maxi = sx;
            }
          if (y[i]%2)
            sy--;
          else
            { sy++;
              if (maxi < sy) maxi = sy;
            }
       }
     printf("%d\n",maxi);
     return 0;
}
```

Автор: Младен Манев

Задача В3. Двоично-десетична игра

След като се научиха как се преобразува едно число от десетична в двоична бройна система и обратно и написаха подпрограми за това, Асен и Боян измислиха и програмираха следната интересна игра:

1. Въвежда се в десетичен запис без водещи нули едно естествено число;

2. Компютърът преобразува това число в съответния му двоичен запис;

3. Показват се двата записа;

4. Играчът, който е на ход, въвежда цифрата **1** или цифрата **2**. Значението на ходовете е следното: **1** означава, че се изтрива най-дясната цифра от двоичния запис, а **2** – че се изтрива най-дясната цифра от десетичния запис. Компютърът извършва изтриването;

5. Ако след хода в избрания запис не е останало нищо, играта завършва с победа на този, който е играл последен;

6. Иначе компютърът преобразува записа, от който не е изтривал в точка **4**, в съответната бройна система (така двата записа отново са на едно и също число в двете бройни системи), на ход е следващият играч и се преминава към точка **3**.

Ето едно примерно развитие на играта:

```
Input: 273
100010001
273
Player #1: 2
```

```
11011
27
Player #2: 1
```

```
1101
13
Player #1: 1
```

```
110 6
Player #2: 2
Player #2
wins.
```

Понеже е по-голям, Боян се изхитрява почти винаги да бие. При това, уверен в себе си, той даже позволява на Асен да избере кой да играе първи. Помогнете на Асен, като напишете програма **decbin**, която да му казва дали да иска да играе първи и, ако е така, какъв ход да играе, за да си осигури победа.

Вход

От стандартния вход се въвеждат четири реда. Всеки от тях представлява входа за една игра и съдържа едно естествено десетично число без водещи нули.

Изход

Запишете на стандартния изход един ред с 4 цифри (без разделители между тях), като всяка от цифрите дава отговор за съответната игра, зададена на входа. Цифрата е **0**, ако с цел осигуряване на победата си Асен трябва да предостави първия ход на Боян; **1**, ако Асен може да постигне победа с ход **1**; или **2**, ако ход **2** ще го доведе до сигурна победа.

Ограничения

Всяка от входните данни е с не повече от 60 десетични цифри.

Забележка

В 40% от тестовите данни броят на цифрите не надминава 18.

Пример

Вход	Изход
31 9 5000 65001	1202

Решение

Начало	Край	Отг.
10	19	0
20	39	1
40	79	0
80	159	1
160	199	2
200	399	0
400	799	1
800	1599	0
1600	3199	1
3200	3999	2
4000	7999	0
8000	15999	1
16000	31999	0
32000	63999	1
64000	79999	2

Пълното изчерпване ще се затрудни сериозно, ако на всяка стъпка се извършват превръщанията от една бройна система в друга. Напротив – съобразяването, че ход тип **1** всъщност е целочислено деление на 2, а ход тип **2** е целочислено деление на 10, позволява изчерпването да завърши сравнително бързо за 64 битови числа, а при подходяща реализация – и до 20 цифрени. За по-големите входни данни трябва да се проследи цикълът на интервалите, в които се получава смяна на отговора.

Ще спомена, че печелившият ход не е винаги еднозначно определен. Например, при вход **120**, както ход **1**, водещ до **60**, така и ход **2**, водещ до **12**, са печеливши. Това обаче не е важно при тази постановка на задачата.

Множеството на естествените числа очаквано


```
  k=10;
 for ( i=3;i>-2;i-=2) //извеждане на другата половина на палиндрома
  { s=(a[k]-(q==k))/2;
    for (j=0;j<(s-(a[k-1]>0));j++)     cout<<k/2;
     if (s>0 &&  k==2 && a[k-1]>0)
            { for ( j=0;j<(a[k-1]/2);j++) cout<<k/2*10;
              //десетки се извеждат само при наличие на единица
                cout<<k/2;
              }
    k-=i*2;
    }
  }
return 0;
}
```

Автор: Теодоси Теодосиев

Задача C2. Камъчета

Три купчини съдържат съответно **a**, **b** и **c** камъчета. Иванчо се опитва да събере всички камъчета в една от трите купчини като извършва няколко пъти следния ход: избира две купчини и от по-голямата премества в по-малката толкова камъчета, колкото има в по-малката (така броят на камъчетата в по-малката купчина се удвоява); ако в двете купчини има равен брой камъчета, той премества всички камъчета от едната купчина в другата. Напишете програма **peb**, която намира колко хода най-малко трябва да направи Иванчо, за да събере всички камъчета в някоя от трите купчини.

Вход

На един ред на стандартния вход се въвеждат **a**, **b** и **c** – броя на камъчетата в първата, втората и третата купчина.

Изход

Програмата извежда на един ред на стандартния изход най-малкия брой ходове, които трябва да направи Иванчо, за да се събере всички камъчетата в една от трите купчини. Ако камъчетата не могат да бъдат събрани в никоя от купчините, програмата трябва да извежда **-1**.

Ограничения

a, b, c > 0
a + b + c < 100

Примери

Вход	Изход
1 2 1	2
1 3 1	-1

Решение

Решението на задачата може да се разбие на няколко подзадачи:

- Преброяване на различните елементи;
  - ✓ използваме индекса на масива за извеждане на необходимото число
- Намиране на най-дългия среден елемент;
  - ✓ при нечетен брой 1-ци тя участва в средния елемент евентуално с 10-ките;
  - ✓ при четен брой единици е възможно, ако има нечетен брой 2-ки или 5-ци среден елемент да са и 1-ците и 2-ките и 5-ците.
- Извеждане на лявата част на редицата;
  - ✓ 10-те участват само в комбинация с единиците;
- Извеждане на дясната част на редицата.
  - ✓ 10-те участват само в комбинация с единиците;
  - ✓ накрая трябва да се изведе една 1-ца.

```
#include <iostream>
using namespace std;
int  main()
{int a[11]={0},n,r,k, q=0,br,i,j;
   cin>>n;
  for (i=0;i<n;i++)
  {cin>>r;
   if (r>9)a[2*(r/10)-1]++; //броене на различните елементи
   else a[2*r]++;   }
  if ((a[2]>0)+(a[4]>0)+(a[10]>0)== 0)cout<<"No";
  else
   {k=2;
     for (i=1;i<6;i+=2) //търсене на най-дълъг среден елемент
       {if (a[k]%2 !=0 ) {q=k;break;}
         k+=i*2;       }
     if (q==2) br=(a[2]==1)? a[1]:a[1]%2;
     else  if (q!=0 && (a[4]>1 || a[10]>1)) br=1;
            else
               if (a[1]%2!=0 && a[2]>0)
                        { q=2; br= (a[2]==2)? a[1]: a[1]%2;}
    k=1; int s;
    for (i=1;i<6;i+=2) //извеждане на половината палиндром
     { s=(a[k+1]-(k+1==q))/2;
       for (j=0;j<a[k]/2;j++)
          if (s>0) cout<<(k+1)/2*10;
       //десетки се извеждат само при наличие на единица
        for (j=0;j<s;j++)  //оставаме 1 единичен за средния елемент
         cout<<(k+1)/2;
       k+=i*2;
      }
    //извеждане на средния елемент
     if (q==2) for (i=0;i<br;i++)    cout<<q/2*10;
     if (q!=0)cout<<q/2;
```


се разделя на интервали със съответен отговор. Разбира се, всички едноцифрени десетични числа са с отговор **2**. За следващите числа се получават следните данни:

Забелязват се следните зависимости в интервалите:

1. Има период от пет интервала, в които се сменят отговорите: **01012**;
2. Началото на всеки период очаквано е 20 пъти по-голямо от началото на предишния;
3. Всяко начало в рамките на периода е два пъти по-голямо от предишното;
4. Краят на всеки интервал, естествено, е с 1 по-малък от следващото начало.

Тези наблюдения ни водят към следния алгоритъм:

Получаваме последователно по горните правила всеки следващ затворен интервал. Проверяваме дали някоя от входните данни попада в него, ако да – запомняме отговора за нея. Процесът продължава, докато и четирите входни данни получат своите решения.

Окончателната програма – решение на задачата:

```
#include <iostream>
using namespace std;
typedef struct
{int n;
 char c[64];
} Long;
Long game[4];
void mulDig(int d, Long *b)
{int carry=0,r;
 for (int i=0;i<b->n;i++)
 {r=(b->c[i]-'0')*d+carry;
  b->c[i]=r%10+'0';
  carry=r/10;
 }
 if (carry) b->c[b->n++]=carry+'0';
}
void mul20(Long *b)
{mulDig(2,b);
 memmove(&b->c[1],b->c,b->n++);
 b->c[0]='0';
}
void Dec(Long *b)
{int i;
 for (i=0;i<b->n;i++)
  if (b->c[i]=='0') b->c[i]='9';
  else break;
 b->c[i]--;
}
int cmp(const Long *a, const Long *b)
{if (a->n<b->n) return -1;
 if (a->n>b->n) return 1;
 for (int i=a->n-1;i>=0;i--)
 {if (a->c[i]<b->c[i]) return -1;
  if (a->c[i]>b->c[i]) return 1;
```

```
 }
 return 0;
}
void setLong(Long *a, char *s)
{a->n=strlen(s);
 for (int i=a->n-1,j=0;i>=0;i--,j++)
  a->c[j]=s[i];
}
void Solve(void)
{Long Start,CurrStart,CurrEnd;
 int i,r[4];
 for (i=0;i<4;i++) r[i]=-1;
 setLong(&Start,"10");
 for (i=0;i<4;i++) if (cmp(&game[i],&Start)<0) r[i]=2;
 while (r[0]<0 || r[1]<0 || r[2]<0 || r[3]<0)
 {CurrStart=Start;
  CurrEnd=Start;
  mulDig(2,&CurrEnd);
  Dec(&CurrEnd);
  for (i=0;i<4;i++)
   if (cmp(&game[i],&CurrStart)>=0 &&
       cmp(&game[i],&CurrEnd)<=0) r[i]=0;
  mulDig(2,&CurrStart);
  CurrEnd=CurrStart;
  mulDig(2,&CurrEnd);
  Dec(&CurrEnd);
  for (i=0;i<4;i++) if (cmp(&game[i],&CurrStart)>=0 &&
                        cmp(&game[i],&CurrEnd)<=0) r[i]=1;
  mulDig(2,&CurrStart);
  CurrEnd=CurrStart;
  mulDig(2,&CurrEnd);
  Dec(&CurrEnd);
  for (i=0;i<4;i++) if (cmp(&game[i],&CurrStart)>=0 &&
                        cmp(&game[i],&CurrEnd)<=0) r[i]=0;
  mulDig(2,&CurrStart);
  CurrEnd=CurrStart;
  mulDig(2,&CurrEnd);
  Dec(&CurrEnd);
  for (i=0;i<4;i++) if (cmp(&game[i],&CurrStart)>=0 &&
                        cmp(&game[i],&CurrEnd)<=0) r[i]=1;
  mulDig(2,&CurrStart);
  mul20(&Start);
  CurrEnd=Start;
  Dec(&CurrEnd);
  for (i=0;i<4;i++) if (cmp(&game[i],&CurrStart)>=0 &&
                        cmp(&game[i],&CurrEnd)<=0) r[i]=2;
 }
 for (i=0;i<4;i++) cout<<r[i];
 cout<<endl;
}
void inp(void)
```



```
{char b[64];
 for (int i=0;i<4;i++)
 {cin>>b;
  setLong(&game[i],b);
 }
}
int main(void)
{int p;
 inp();
 Solve();
 return 0;
}
```

Автор: Павлин Пеев

Задача C1. Огледална последователност

Програмко Сиплюсов се готви сериозно за Есенния турнир в Шумен. Всички решени задачи записва в папка “ET_Shumen”. Веднъж, докато чака да приключи поредния тест на Програмко му прави впечатление, че всички файлове в папката са с размер 1K, 2K, 5K или 10K и още нещо го изненадва: файловете са подредени така, че размерите им, записани като цифрова последователност, образуват огледална редица (една и съща четена отляво-надясно и обратно). Това му дава идея за интересна задача: да се изведе най-дългата огледална последователност от дадените размери. За целта вие трябва да напишете програма **pal**, която да прави това.

Вход

На първия ред на стандартния вход се въвежда цяло число **N**, показващо броя на файловете. На следващия ред се въвеждат **N** числа **(1**, **2**, **5** или **10)**, които показват размера на файловете.

Изход

Програмата извежда на стандартния изход най-дългата симетрична редица от тези числа. При няколко последователности с еднаква дължина се извежда първата по лексикографска наредба. Ако не съществува огледална редица се извежда “**No**”.

Ограничения

1 < N < 10000

Примери

Вход	Изход
8 5 1 5 2 5 2 1 2	125251
9 10 1 1 5 10 1 5 10 10	1010151510101
5 10 10 10 10 10	No