Национална школа по информатика – Ямбол 2011

Делимост. Прости числа.
Пламенка Христова

1. Делимост. Намиране делителите на дадено число

Едно число **b** дели друго число **a**, ако е изпълнено равенството **a = n*b**, където **n** е естествено число, а **a** и **b** са цели числа.

В много случаи за решаването на една задача е необходимо да се намерят делителите на дадено число.

От математиката се знае, че делителите на числото **a** са в интервала [2, a/2].

Пример 1: Напишете програма **DIV1,** която прочита от клавиатурата едно цяло число **a** и намира делителите му, които са различни от 1.

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
  int a, b, c;
  cin >>a;
  c=a/2;
  for (b=2; b<=c; b++)
     if (a%b==0)cout <<b<<" ";
  cout <<endl;
     }
```

2. Прости числа

Простите числа са свързани с много интересни математически задачи и ползата от тях се простира далеч извън пределите на математиката. В информатиката те намират приложение в криптирането, архивирането, и много други области.

Определение: Едно цяло число ***p*** (p>1) се нарича просто, ако няма други делители освен 1 и ***p***. Ако ***p*** не е просто, то се нарича съставно. Редицата от простите числа започва така:

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, …

Доказано е (*Евклид* – 300 г.пр.н.е.), че простите числа са безброй много.

Когато разглеждаме редицата на простите числа, възникват следните ***въпроси***:

- Колко прости числа има в даден интервал $[a, b]$?
- Каква част от безкрайността представляват простите числа?
- Съществува ли формула за намиране на n-тото поред просто число?

Преди да преминем към алгоритмите за проверка дали едно число е просто, ще споменем още няколко интересни свойства и теореми за простите числа:

Хипотези на Голдбах

1. Всяко цяло четно число n>2 може да се представи като сума на 2 прости числа.
2. Всяко цяло число n>17 може да се представи като сума на 3 различни прости числа.
3. Всяко цяло число може да се представи като сума на най-много 6 прости числа.
4. Всяко цяло нечетно число n>5 може да се представи като сума на 3 прости числа.
5. Всяко четно число може да се представи като разлика на две прости числа.

Теорема. Съществуват безброй много прости числа от вида n^2+m^2 и n^2+m^2+1.

Хипотеза. Съществуват безброй много прости числа от вида n^2+1.

Теорема. (*Оперман*) Винаги съществува просто число между n^2 и $(n+1)^2$.

Проверка дали дадено число е просто:

Един очевиден алгоритъм, пряко следствие от дефиницията, е следният: Проверяваме всяко число от интервала [2 , $\frac{p}{2}-1$] дали дели p и ако намерим някое, което го дели, следва, че p е съставно.

Съществуват някои “формули” за проверка дали число е просто, но на практика те се оказват неприложими, тъй като реализацията им изисква много повече компютърни изчисления, отколкото токущо описаният алгоритъм. Един пример е следната ***Теорема на Уилсън:*** Числото p е просто тогава и само тогава, когато $(p–1)! = –1$ ($mod\ p$).

При нея трябва да се изчисли $(p–1)!$, което е доста по-трудно като реализация и предполага многократно повече изчисления от извършването на $\frac{p}{2}-1$ деления в горния алгоритъм. Ще се опитаме да го подобрим още.

Лесно може да се съобрази, че е излишно да проверяваме всички числа до $\frac{p}{2}-1$, а е достатъчно да проверим за делимост само до $\sqrt{p}$. Това е така, тъй като винаги, когато p има делител x, $x>\sqrt{p}$, то следва, че p се представя във вида $p=x.y$, $y<\sqrt{p}$, т.е. има и делител по-малък от $\sqrt{p}$. Ето една реализация на този алгоритъм:

Алгоритъм:

```
Ако  n = 2, то връща 1,
        Иначе i = 2
        Докато i <= корен квадратен от n прави
                Ако  n mod i = 0, то връща 0
                i = i+1
        връща 1
```

```
#include <iostream>
#include <math.h>
using namespace std;
/* 1-prosto, 0-ne e prosto */
int isPrime(int &n) {
  if (n==2) return 1;
  int i = 2;
  while (i <= sqrt(n)) {
    if (n % i == 0) return 0;
    i++;
  }
  return 1;
}
int main() {
  int n;
  cout<<"N=";
  cin >>n;
  if (isPrime(n)==1) cout <<"Number is Prime"<<endl;
     else cout <<"Number is not Prime"<<endl;
}
```

Можем да разширим още малко последния резултат: за да установим, че p е просто, е достатъчно да сме сигурни, че не се дели на нито едно *друго просто число* от интервала [2, $\sqrt{p}$]. Така, ако разполагаме с първите k прости числа, ще можем да проверяваме дали произволно число от интервала [2, k^2] е просто. Следващата програма реализира това:

```
#include <iostream>
using namespace std;
const int K=25;
```

```
int prime[K] = {2,3,5,7,11,13,17,19,23,29,31,37,41,
                43,47,53,59,61,67,71,73,79,83,89,97};
int checkprime(int n) {
  int i = 0;
  while (i < K && prime[i]*prime[i] <= n) {
    if (n % prime[i] == 0) return 0;
    i++;
  }
  return 1;
}
int main() {
  int n;
  cout << "N=";
  cin >>n;
  if (checkprime(n)) cout<<"Chisloto e prosto."<<endl;
     else cout<<"Chisloto e sustavno. "<<endl;
}
```

3. Търсене на прости числа в интервал

Намиране на всички прости числа по-малки от n:

***Задача*:** За дадено цяло число n, да се намерят всички прости числа в интервала [2, n].

Очевиден подход за решение на задачата е да проверяваме дали всяко число от дадения интервал е просто. Така извършваме n проверки, като за всяко число k ще бъдат необходими най-много $\sqrt{k}$ на брой проверки за делимост.

Алгоритъм за търсене на прости числа в интервала [a,b]

Всяко естествено число N може да се представи така:

N=30q + r, където r е едно от числата 0, +/-1, +/-2, …, +/-15 , а коефициентът 30=2*3*5 е получен от произведението на първите три прости числа.

Тогава на всеки 30 последователни числа евентуално прости са само следните осем:
30q + /-1, 30q + /-7, 30q + /-11, 30q + /-13

4. Решето на Ератостен

При условие, че разполагаме с достатъчно памет, можем да приложим по-бърз метод за намиране на простите числа в интервал, наречен "*решето на Ератостен*". Както подсказва името, "*решетото*" е метод за програмиране, при който се изключват всички елементи от крайно множество, които не ни интересуват. В случая идеята на този подход идва от предишната точка: едно число е просто, ако няма прости делители (освен себе си). Решетото на Ератостен се състои в следното:

Записваме числата от 2 до n в редица:

2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, ... , n

Намираме първото незачертано и немаркирано число — това е 2. Маркираме го, след което задраскваме всяко второ число:

(2), 3, ~~4~~, 5, ~~6~~, 7, ~~8~~, 9, ~~10~~, 11, ~~12~~, 13, ~~14~~, 15, ~~16~~, 17, ... , n

По-нататък, отново намираме първото незачертано и немаркирано число — това е числото 3. Маркираме го и задраскваме всички числа в редицата, кратни на 3:

(2), (3), ~~4~~, 5, ~~6~~, 7, ~~8~~, ~~9~~, ~~10~~, 11, ~~12~~, 13, ~~14~~, ~~15~~, ~~16~~, 17, ... , n

След това, на ред е числото 5 — маркираме го и задраскваме всяко 5-то:

(2), (3), ~~4~~, (5), ~~6~~, 7, ~~8~~, ~~9~~, ~~10~~, 11, ~~12~~, 13, ~~14~~, ~~15~~, ~~16~~, 17, ... , n

Така всички съставни числа се "отсяват" и винаги сме сигурни, че най-малкото незадраскано i е просто. Процесът продължава, докато не остане нито едно незадраскано или немаркирано число — тогава всички числа, които са маркирани са прости, а всички задраскани са съставни.

Алгоритъм на решетото на Ератостен

1.) Инициализираме един масив с N елемента с нули. По-късно, когато задраскаме някое число, на съответната позиция в масива ще записваме 1. I показва кое е първото незадраскано или немаркирано число. Започваме от 2

2.) Увеличаваме I докато съответния елемент от масива стане 0. Тогава числото I е просто и го извеждаме.

3.) Маркираме с 1 всички стойности в масива за К=2I, 3I, …, (N/i) I – всички кратни на I стойности.

4.) Ако I<= N, то се връщаме на стъпка 2, иначе приключваме.

```
#include <iostream>
#include <math.h>
using namespace std;
const int maxN=30000;
char a[maxN];
void eratosten(int n) {
  int i = 2;
  while (i <= n) {
    if (a[i] == 0) {
       cout<<i<<" ";
       int j = i;
       while (j <= n) {
         a[j] = 1;
         j += i;
       }
    }
    i++;
   }
}
int main() {
  int n;
  for (int i=0; i<maxN; i++) a[i]=0;
  cout<<"N=";cin>>n;
  eratosten(n);
}
```

Съществува още по-ефективен алгоритъм за търсене на всички прости числа в интервал. При него не е необходим масив както и не е необходимо на всяка стъпка да се обхожда целия интервал.

При реализацията на алгоритъма за проверка дали едно число е просто използвахме масив А, съдържащ първите К прости числа и с негова помощ проверявахме дали едно число от интервала $[к+1,к^2]$ е просто. Сега ще направим следното: ще започнем от празен списък, който ще попълваме последователно. Например, поставяйки в него първото число (2) можем да намерим всички прости числа в интервала [3,4] – такова е 3 и го добавяме в списъка. След това [4,9] – това са 5 и 7, които също добавяме в списъка. Продължаваме този процес, докато в масива се добавят достатъчно прости числа, за да покрият проверката дали всяко число от интервала [2, n] е просто.

Ето програмата:

```
#include <iostream>
using namespace std;
int primes[10000];
void findPrimes(int nn) {
  int i = 2,key;int max = 0;
  while (i < nn) {
     int j = 0;key=1;
  while (j < max && primes[j]*primes[j] <= i) {
```

```
      if (i % primes[j] == 0) key=0;
      j++;
    }
      if (key==1) {
         primes[max] = i;
         max++;
         cout <<i<<" ";
      }
      i++;
    }
}
int main() {
  int n;
  cout<<"N=";
  cin >>n;
  findPrimes(n);
}
```

5. Намиране простите делители на едно число

Факторизация

Всяко цяло число P може да се представи (*факторизира*) по единствен начин във вида: $P_1^{q1}.\ P_2^{q2}.\ \ldots\ .P_n^{qn}$, където q_i>0 и P_1<P_2< ...<P_n са прости числа.

Следват няколко примера, илюстриращи това свойство:

$520 = 2^3.5^1.13^1$

$64 = 2^6$

$2345 = 5^1.7^1.67^1$

Алгоритъмът, за получаване на подобно разлагане (необходимо при някои изчислителни задачи), е следния:

Нека разлагаме числото P.

1) Полагаме i=2.
2) Полагаме k=0. Докато P се дели на i, извършваме делението и увеличаваме k с единица. Преминаваме към стъпка 3).
3) Ако k>0, то сме получили поредния член от разлагането — той е i^k. Преминаваме на стъпка 4).
4) Ако P>1 увеличаваме i с единица и се връщаме на стъпка 2).

Следващата програма **NUMDEV** извежда разлагането на дадено число на прости множители. Пример:

24=2 2 2 3

```
#include <iostream>
using namespace std;
int main() {
  int n,how,i;
  cout <<"N= ";
  cin >>n;
  cout <<n<<" = ";
  i=1;
  while (n != 1) {
     i++;
     how=0;
     while ((n % i)==0) {
        how++;
        n = n / i;
     }
```

```
        for (int j=0; j<how; j++) cout << i<<" ";
    }
    cout <<endl;
  }
```

Намиране броя на нулите, на които завършва произведение

Задача: Дадена редица от цели числа $a_1, ..., a_n$, и търсим броя на нулите, на които завършва произведението $P=a_1.a_2. ... .a_n$.

Извършването на умножението е нежелателно и не винаги ще доведе до получаване на търсения резултат. За да решим задачата, ще обърнем внимание на следния факт: Единствените прости числа, чието произведение завършва на нула, са 2 и 5, или произведение на число, кратно на 2 с число, кратно на 5.

Така алгоритъмът, който решава задачата без извършване на умножението, има следния вид:

1) За всяко a_i (i=1, ..., n) представяме a_i във вида $a_i = 2^{Mi}.5^{Ni}.b_i$
2) Резултатът от произведението ще бъде $P= 2^{\sum_{i=1..n} M_i}.5^{\sum_{i=1..n} N_i}.c$, ($c$ е константа), а броят на нулите в края на произведението ще бъде минималното от $\sum_{i=1}^{n} Mi$ и $\sum_{i=1}^{n} Ni$.

Например, за редицата
25, 4, 20, 11, 13, 15
след разлагането ще получим:
$2^0.5^2.1$, $2^2.5^0.1$, $2^2.5^1.1$, $2^0.5^0.11$, $2^0.5^0.13$, $2^0.5^1.3$,
което дава 4 нули. Правилността на получения резултат можем да проверим директно: 25.4.20.11.13.15=4290000.

Реализацията на току-що описания алгоритъм е лека модификация на вече разгледания алгоритъм за разлагане на число на прости делители от предишната точка.

Следващата програма намира броя на нулите, на които завършва произведението на две числа.

```
#include <iostream>
using namespace std;
int br2(int a){
  int i, how;
  i=2;
   how=0;
     while ((a % i)==0) {
        how++;
        a = a / i;
     }
 return how;
}
int br5(int a){
  int i, how;
  i=5;
   how=0;
     while ((a % i)==0) {
        how++;
        a = a / i;
     }
 return how;
}
int main() {
  int n,m,br2n,br2m,br5n,br5m,sum2,sum5;
  cout <<"N= ";
```

```
  cin >>n;
  cout <<"M= ";
  cin >>m;
  br2n=br2(n);
  br2m=br2(m);
  br5n=br5(n);
  br5m=br5(m);
  sum2=br2n+br2m;
  sum5=br5n+br5m;
  if (sum2<sum5) cout <<sum2;
  else cout <<sum5;
  cout <<endl;
}
```

Взаимно прости числа

Определение: Две цели положителни числа a и b се наричат взаимно прости, ако нямат общ делител, по-голям от едно.

Задачи:

Задача 1. Напишете програма **DIVSUMN**, която чете от стандартния вход едно цяло положително число a, и извежда сумата от тези негови делители, които са различни от 1 и a.

Пример:	**Вход**	**Изход**
	18	20

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
  int b,c,a,i,s=0;
  cin >>a;
  c=a/2;
  for (b=2; b<=c; b++)
     if (a%b==0)s+=b;
  cout <<s<<endl;
}
```

Задача 2. Напишете програма **SEVEN**, която чете от всеки ред на стандартния вход по едно цяло положително число.Въвеждането продължава докато се въведе 0. Програмата да намери най-голямото от дадените числа, което се дели на 7 и да се запише в единствения ред на стандартния изход. Ако никое от числата не се дели на 7, да се запише числото нула.

Пример 1:

Вход
12345
277
777
77749
33
234
0

Изход
77749

Пример 2:

Вход
582576
340579
6780
9327
0

Изход
0

```
#include <iostream>
#include <string>
```

```
using namespace std;
int main()
{
  int a,max=0;
  cin >>a;
  do{
   if (a%7==0){
    if (max==0)max=a;
    else
     if (max<a)max=a;}
   cin>>a;
  } while(a!=0);
   cout <<max<<endl;
}
```

Задача 3. (ПРОСТИ ЧИСЛА) Дадени са целите положителни числа *N* (5≤*N*≤40000) и *K* (1≤*K*<*N*). Напишете програма **SMPK**, която намира *K*-тото по големина просто число, ако то не надминава *N*. По дефиниция 1 не се счита просто число, затова най-малкото просто число е 2, второто по големина е 3 и т.н. Единственият ред на стандартния вход ще съдържа числата *N* и *K*, разделени с един интервал. Изходът трябва да се състои също от един ред, в който е записано намереното *K*-то по големина просто число, ако то не е по-голямо от *N*. Ако *K*-то по големина просто число е по-голямо от *N*, тогава в единствения ред на стандартния изход трябва да е записано числото 0. (Pleven 2002, зад.1, C)

ПРИМЕРИ:	Вход	Изход
	100 10	29
	11 5	11
	12 10	0

```
#include <iostream>
using namespace std;
int main()
{
int N,K,erato[40001];
 int i,sim,numsim;
 cin >>N>>K;
   for(i=1;i<=N;i++) erato[i]=1;
   sim=2;numsim=1;
   while(sim<N && numsim<K)
   {
      for(i=sim;i<=N;i=i+sim) erato[i]=0;
      i=sim+1;while(i<=N && erato[i]==0) i++;
      if(i>N) break;
      sim=i;numsim++;
   }
   if(numsim==K) cout<<sim;
   else cout<<"0";
 }
```

Задача 4. (Делимост) Напишете програма **DIV11**, която въвежда от клавиатурата трицифрено число и определя, дали като се задраска първата цифра и се напише накрая, се получава число, което се дели на 11. Програмата извежда на екрана съответно "Yes" или "No".

Пример1:		***Пример 2:***	
Вход:	*Изход:*	*Вход:*	*Изход:*
314	Yes	315	No

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
  int a, b, c, s=0;
  cin >>a;
  c=a/100;
  b=a%100;
  b=b*10+c;
   if (b%11==0) cout <<"YES"<<endl;
  else cout <<"NO"<<endl;
     }
```

Задача 5. (Съвършено число) Да се определи дали дадено естествено число е съвършено, т.е. дали е равно на сумата на всичките си делители, освен самото то. (например, числото 6 е съвършено, защото 6=1+2+3, които са всичките му делители). Името на програмата е **PERF**. Програмата да извежда YES, ако числото е съвършено и NO, ако не е.

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
  int a, b, c, s=0;
  cin >>a;
  c=a/2;
  for (b=1; b<=c; b++)
     if (a%b==0)s+=b;
  if (a==s) cout <<"YES"<<endl;
  else cout <<"NO"<<endl;
 }
```

Задача 6. Напишете програма **NOK**, която въвежда две цели положителни числа a и b и отпечатва тяхното най-малко общо кратно. (От математиката е известно, че най-малкото общо кратно е равно на частното на a*b с най-големия общ делител на a и b.)