Ciurul lui Eratostene. Aplicații.

O alternativă la determinarea proprietății de număr prim este ciurul lui Eratostene. Acesta este un vector, de obicei caracteristic, în care componenta de pe poziția **i** are semnificația: dacă **ciur[i]** este 0 atunci numărul **i** este prim, iar dacă **ciur[i]** este 1, numărul **i** nu este prim.

Principiul de completare este următorul: inițial vectorul este completat cu 0 (excepție componentele de pe pozițiile 0 și 1) presupunând că toate numerele sunt prime. Pentru fiecare număr prim (**ciur[i]=0**), se parcurg toți multiplii acestuia, din intervalul considerat, și se setează **ciur[j]=1,** deoarece **j** nu mai este prim.

Implementarea în C++ a unui subprogram care realizează această construcție este prezentat mai jos:

```
const int maxx=1000;

int ciur[maxx];

void ciur_eratostene()
{
    int i,j;
    ciur[0]=ciur[1]=1;
    //se parcurg multiplii lui 2
    for(i=4;i<=maxx;i=i+2)
        ciur[i]=1;
    //se parcurg numerele impare
    for(i=3;i*i<=maxx;i=i+2)
        if(ciur[i]==0)
     //daca numarul i este prim, multiplii sai nu sunt numere prime
            for(j=i*i;j<=maxx;j=j+2*i)
          //primul multiplu, nevizitat anterior, al lui i este i*i
                ciur[j]=1;
}
```

Utilizarea ciurului lui Eratostene se pretează în probleme în care se solicită, în mod repetat, folosirea proprietății de număr prim. Exemplu: se citesc **n** numere naturale cu cel mult 3 cifre. Afișați numerele prime.

```
int main()
{
    //Utilizare ciur
    ciur_eratostene();
    int n,i,x;
    cin>>n;
    for(i=1;i<=n;i++)
    {
        cin>>x;
        if(ciur[x]==0)
            cout<<x<<" ";
    }
    return 0;
}
```

Aplicații care utilizează ciurul lui Eratostene

Numărul de factori primi

Problemă: Se citesc **n** numere naturale de cel mult 3 cifre. Afișați, pentru fiecare număr citit, numărul de factori primi din descompunerea sa.

Rezolvare: Se realizează construirea vectorului **ciur**. De data aceasta, el nu va mai fi un vector caracteristic. Principiul de completare: pentru fiecare număr prim (**ciur[i]=0), se** adună un 1 el și la toți multiplii acestuia deoarece fiecare dintre ei îl va avea în descompunerea în factori primi pe **i.**

```
const int maxx=1000;

int ciur[maxx];

void ciur_eratostene()
{
    int i,j;
    ciur[0]=ciur[1]=1;
    //se parcurg multiplii lui 2
    for(i=2;i<=maxx;i=i+2)
        ciur[i]=1;
    //se parcurg numerele impare
    for(i=3;i<=maxx;i=i+2)
        if(ciur[i]==0)
```

```
            for(j=i;j<=maxx;j=j+i)
                ciur[j]++;
}

int main()
{
    //Utilizare ciur
    ciur_eratostene();
    int n,i,x;
    cin>>n;
    for(i=1;i<=n;i++)
    {
        cin>>x;
        cout<<x<<" "<<ciur[x]<<"\n";
    }
    return 0;
}
```

Numărul de divizori

Problemă: Se citesc **n** numere naturale nenule de cel mult 3 cifre. Afișați, pentru fiecare număr citit, numărul de divizori.

Rezolvare: Principiul de completare: pentru fiecare număr **i,** se parcurg toți multiplii, **j**, inclusiv el însuși și se adună un 1 la fiecare **ciur[j].**

```
using namespace std;

const int maxx=1000;

int ciur[maxx];

void ciur_eratostene()
{
    int i,j;
    for(i=2;i<=maxx;i++)
        for(j=i;j<=maxx;j=j+i)
            ciur[j]++;
}

int main()
{
    //Utilizare ciur
    ciur_eratostene();
```

```
    int n,i,x;
    cin>>n;
    for(i=1;i<=n;i++)
    {
        cin>>x;
        cout<<x<<" "<<ciur[x]+1<<"\n";
        //fiecare numar il are ca divizor si pe 1
    }
    return 0;
}
```

Suma divizorilor

Problemă: Se citesc **n** numere naturale nenule de cel mult 3 cifre. Afișați, pentru fiecare număr citit, suma divizorilor.

Rezolvare: Principiul de completare: pentru fiecare număr **i,** se parcurg toți multiplii, **j**, inclusiv el însuși și se adună **i** la fiecare **ciur[j]** (multiplii lui **i** îl vor avea ca divizor pe acesta).

```
using namespace std;

const int maxx=1000;

int ciur[maxx];

void ciur_eratostene()
{
    int i,j;
    for(i=2;i<=maxx;i++)
        for(j=i;j<=maxx;j=j+i)
            ciur[j]=ciur[j]+i;
}

int main()
{
    //Utilizare ciur
    ciur_eratostene();
    int n,i,x;
    cin>>n;
    for(i=1;i<=n;i++)
    {
        cin>>x;
        cout<<x<<" "<<ciur[x]+1<<"\n";
        //fiecare numar il are ca divizor si pe 1
```

```
    }
    return 0;
}
```

Indicatorul lui Euler

Problemă: Se citesc **n** numere naturale nenule de cel mult 3 cifre. Afișați, pentru fiecare număr citit, numărul de numere prime cu el, mai mici decât el.

Rezolvare: Cerința face referire la indicatorul lui Euler.

Indicatorul lui Euler. Totientul sau indicatorul lui Euler determină, folosind descompunerea în factori primi, numărul de numere prime cu **n,** mai mici decât **n.** Din acest motiv, lecția este prezentată ca o aplicație directă a descompunerii în factori primi ai unui număr natural nenul.

Se scrie **n** sub forma:

$$n = {f_1}^{p_1} * {f_2}^{p_2} * \dots * {f_k}^{k}$$

Indicatorul lui Euler se poate utiliza cu una din cele două forme:

$$\varphi(n) = n * \left(1 - \frac{1}{f_1}\right) * \left(1 - \frac{1}{f_2}\right) * \dots * \left(1 - \frac{1}{f_k}\right), \qquad forma\ (1)$$

$$\varphi(n) = (f_1 - 1) * {f_1}^{p_1 - 1} * (f_2 - 1) * {f_2}^{p_2 - 1} * \dots * (f_k - 1) * {f_k}^{p_k - 1}, \qquad forma\ (2)$$

În rezolvarea problemei enunțate mai sus, se va utiliza forma (1) a indicatorului lui Euler.

Principiul de completare al ciurului: fiecare poziție din ciur se completează cu **i,** apoi, pentru fiecare multiplu **j** al unui număr prim se completează **ciur[j]** cu **(1-1/i)** adică **(i-1)/i .**

Numerele prime, **x,** au valoarea totientului **x-1.**

```
#include <iostream>
using namespace std;
const int maxx=1000;
int ciur[maxx];
void ciur_eratostene()
{
    int i,j;
    for(i=1;i<=maxx;i++)
        ciur[i]=i;
    //un numar este prim daca, ramane ciur[i] ramane i
    for(i=2;i<=maxx;i++)
```

```
        if(ciur[i]==i)
            for(j=i;j<=maxx;j=j+i)
                ciur[j]=ciur[j]*(i-1)/i;
}

int main()
{
    //Utilizare ciur
    ciur_eratostene();
    int n,i,x;
    cin>>n;
    for(i=1;i<=n;i++)
    {
        cin>>x;
        cout<<x<<" "<<ciur[x]<<"\n";
        //se afiseaza fiecare numar si indicatorul lui
    }
    return 0;
}
```

Problema LIBER DE PĂTRATE

#3315, Eratostene4 – PBINFO.RO

Problemă: Se dau **n** numere naturale. Pentru fiecare număr aflaţi câţi divizori liberi de pătrate are acesta. Un număr natural se numeşte **liber de pătrate** dacă nu se divide cu pătratul unui număr prim.

Exemplu:
eratostene4.in
3
20 8 5

eratostene4.out
4 2 2

Explicație
Divizorii lui 20, liberi de pătrate, sunt: 1, 2, 5, 10.
Divizorii lui 8, liberi de pătrate, sunt: 1, 2.
Divizorii lui 5, liberi de pătrate, sunt: 1, 5.

Rezolvare: Pentru a explica ideea de rezolvare, vom considera exemplul lui n=20.

$20 = 2^2 * 5^1$. Divizorii liberi de pătrate vor fi printre divizorii numărului 20, însă vor fi de forma $2^a * 5^b$, unde a și b pot lua valorile 0 sau 1. În consecință aceste valori vor fi: $2^0 * 5^0 = 1$, $2^0 * 5^1 = 5$, $2^1 * 5^0 = 2$ și $2^1 * 5^1 = 10$. Aceștia vor fi în număr de 4 = 2* 2.

Pentru un număr oarecare **n**, care are **k** factori primi în descompunerea sa, indiferent de puterea la care aceștia apar, numărul de divizori liberi de pătrate este produsul cartezian a **k** mulțimi de forma {0, 1}, deci 2^k.

Având în vedere această observație, problema se rezolvă ușor. Vom construi termenii unui ciur al lui Eratostene astfel încât **ciur[i]** să rețină numărul de factori primi din descompunerea lui **i**.

```
#include <bits/stdc++.h>

using namespace std;

ifstream f("eratostene4.in");
ofstream g("eratostene4.out");

const int maxx=1000;

int ciur[maxx];

void ciur_eratostene()
{
    int i,j;
    ciur[0]=ciur[1]=1;
    //se parcurg multiplii lui 2
    for(i=2;i<=maxx;i=i+2)
        ciur[i]=1;
    //se parcurg numerele impare
    for(i=3;i<=maxx;i=i+2)
        if(ciur[i]==0)
            for(j=i;j<=maxx;j=j+i)
                ciur[j]++;
}


int main()
{
    //Utilizare ciur
    ciur_eratostene();
    int n,i,x;
    f>>n;
    for(i=1;i<=n;i++)
```

```
    {
        f>>x;
        g<<(1ull<<ciur[x])<<" ";
        //1ull - conversie la unsigned long long
        //1<<x echivalent cu 2^x pe biti
    }

    return 0;
}
```