# Algoritmi i strukture podataka
# 2. Čas, Eratostenovo sito

Aleksandar Veljković

2017/2018

## 1 Definicija problema i naivni algoritam

Za zadatu vrednost $n$ potrebno je ispisati sve proste brojeve $\leq n$. Broj se smatra prostim ako je veći od 1 i ako su mu jedini delioci broj 1 i on sam. Naivnim algoritam podrazumeva proveru deljivosti svakog broja sa svim brojevima koji su manji ili jednaki od korena broja. Dovoljno je proveriti deljivost broja $n$ sa brojevima koji su manji ili jednaki korenu broja $n$ jer u slučaju da je $n$ složen, da ima pravog delioca, on se može zapisati u obliku $i \cdot j$, pri čemu za $i$ ili $j$ važi da mu je najveća moguća vrednost $\sqrt{n}$.

**Naivni algoritam:**

```
#include <iostream>
#include <cmath>

using namespace std;

bool prost(int n)
{
    double koren = sqrt(n);
    for(int j = 2; j <= koren; j++)
        if(n % j == 0)
            return false;

    return true;
}

int main()
{
    int n;
    cin >> n;

    for(int i = 2; i <= n; i++)
        if(prost(i))
            cout << i << " ";

    return 0;
}
```

Vremenska složenos naivnog algoritma je $O(n\sqrt{n})$, dok je prostorna složenost $O(1)$.

# 2 Eratostenovo sito

Starogrčki matematičar Eratosten pronašao je efikasan postupak za pronalaženje prostih brojeva manjih ili jednakih vrednosti $n$. Postupak zahteva niz dužine $n$ (u implementaciji dužine $n+1$ jer je niz indeksiran od 0) tako da $i$-ti element odgovara broju $i$. Inicijalno su svi brojevi lažno označeni kao prosti i u toku postupka se eliminišu svi oni koji to nisu (brojevi se "prosejavaju kroz sito"). Pronalaženje prostih brojeva počinje od najmanjeg prostog broja, broja 2.

Ako je broj $x$ prost, onda brojevi koji su umošci broja $x$ $(2x, 3x, ...)$ nisu prosti brojevi i mogu se eliminisati. Kada su svi umnošci broja $x$ označeni kao složeni, prelazi se na sledeći neoznačeni broj u nizu i eliminišu se njegovi umnošci. Postupak se ponavlja za sve neoznačene brojeve u toku prolaska kroz niz. Navedeni postupak se naziva Eratostenovo sito (*Sieve of Eratosthenese*).

**Eratostenovo sito:**

```
vector<bool> niz(n+1);

/* Inicijalizacija niza tako da su svi elementi oznaceni kao prosti */

for(int i = 0; i <= n; i++)
    niz[i] = true;

double koren = sqrt(n);

for(int i = 2; i <= koren; i++)
    if(niz[i] == true) /*Eliminisanje umnozaka prostog broja*/
    {
        for(int j = 2; i * j <= n; j++)
            niz[i*j] = false;
    }

for(int i = 2; i <= n; i++)
    if(niz[i])
        cout << i << " ";
```

## 2.1 Analiza složenosti algoritma

Prostorna složenost algoritma je u osnovi $O(n)$. Ipak, vektor **bool** vrednosti može, u zavisnosti od implementacije, efikasnije koristiti memoriju tako da su elementi niza označeni pojedinačnim bitovima, umesto bajtovima.

Vremensku složenost algoritma nije trivijalno izračunati. Složenost dolazi od unutražnje petlje koja pronalazi i eliminiše umnoške prostih brojeva. Petlja se izvršava za svaki prost broj do $\sqrt{n}$ pa je broj koraka:

$$\frac{n}{2} + \frac{n}{3} + ... + \frac{n}{\sqrt{n}} = n \cdot (\frac{1}{2} + \frac{1}{3} + ... + \frac{1}{\sqrt{n}})$$

Kako je:

$$\sum_{p \leq n, pprost} \frac{1}{p} = O(\log \log n)$$

to je:

$$\frac{1}{2} + \frac{1}{3} + ... + \frac{1}{\sqrt{n}} = O(\log \log \sqrt{n})$$

Pošto je $\log \sqrt{n} = \log n^{\frac{1}{2}} = \frac{1}{2} \cdot \log n$ može se reći da gornja suma pripada $O(\log \log n)$, pa je ukupna vremenska složenost algoritma:

$$n \cdot (\frac{1}{2} + \frac{1}{3} + ... + \frac{1}{\sqrt{n}}) = n \cdot O(\log \log n) = O(n \log \log n)$$