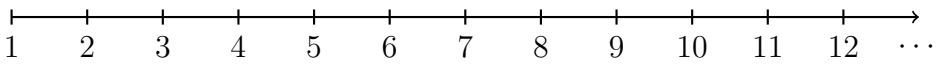
Eratostenovo rešeto

Aleksandar Jurišić in Matjaž Urlep

1 Uvod

Doma (v točki ena) nam postane dolgčas in podamo se na sprehod po številski premici.



Slika 1: Naravna števila, predstavljena na številski premici.

Hitro opazimo, da se večkratniki števil pojavljajo enakomerno. Poglejmo si dva primera.

- Vsako *sodo* število se pojavi na vsakem drugem koraku (pri tem seveda privzamemo, da je dolžina našega koraka enaka ravno enoti), ali če želite, ob desni nogi (če smo začeli naš sprehod z levo).
- Vsak večkratnik števila *tri* se pojavlja izmenično enkrat na levi, enkrat na desni nogi, vsakokrat pa na dveh vmesnih korakih manjka.

Kaj pa če opazujemo cela števila, ki niso večkratniki nobenega drugega števila, kakor le samega sebe in enice (s slednjo bi tako ali tako pokrili vsa števila)? Tem številom pravimo *praštevila* oziroma nerazcepna števila, saj se jih ne da razcepiti v produkt dveh od ena različnih števil.

Ali se tudi praštevila pojavljajo po kakšnem pravilu?

V tem sestavku bomo spoznali, kako bi se s tem lahko ukvarjal F. Gauss, če bi imel na voljo računalnik. Prav on je bil namreč tisti, ki je postavil domnevo o gostoti praštevil.

2 Rešeto

Gotovo ste v šoli že slišali za Eratostenovo rešeto. To je eden od najstarejših algoritmov, s pomočjo katerega lahko najdemo praštevila. Sestavimo tabelo števil od 1 do npr. 400. Za začetek prečrtamo vsa soda števila, z izjemo prvega, nato vse večkratnike števila 3, zopet z izjemo prvega. Ker smo število 4 že prečrtali, nadaljujemo z večkratniki števila 5. Po tem postopku pridejo na vrsto še večkratniki števil 7, 11, 13, 17 in 19.

Gotovo veste, zakaj nam v tabeli ostanejo samo še praštevila skupaj z enico[1]?

Ni kaj, to je kar učinkovit algoritem. Ko enkrat že poznamo praštevila do 20, je morda bolj zabavno opravljati izločanje sestavljenih števil v obratnem vrstnem redu, od večkratnikov števil 19, 17, . . ., pa vse do večkratnikov števila 2. V pomoč ti je lahko tabela na naslednji

[1]Naj vam zaupava, da najmanjši izmed faktorjev sestavljenega števila ni nikoli večji od korena tega števila.

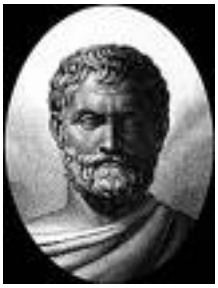
strani, lahko pa skočiš na spletno stran `http://www.hbmeyer.de/eratosiv.htm`, kjer lahko opazuješ kakšne vzorce naredijo izginjajoča sestavljena števila. Zabava gor ali dol, a ne zanimajo nas praštevila samo do 400. Če hočemo še naprej, si bomo seveda pomagali z računalnikom. Zapišimo kodo, ki opravi to delo namesto nas. To bo psevdo koda, tj. poenostavljena koda, ki ni pisana za noben konkreten programski jezik, temveč skuša predstaviti samo idejo algoritma na čim enostavnejši način.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120
121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140
141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180
181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200
201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220
221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260
261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280
281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300
301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320
321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340
341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360
361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380
381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400

Slika 2: Tabela prvih 400 naravnih števil za tiste, ki bodo poskusili izvajati postopek Eratostenovega rešeta na roke.

Eratosthenes se je rodil leta 276 p.n.š. v severni Afriki (ki je danes del Libije) in študiral v Atenah (bil je tudi tretji glavni knjižničar v Aleksandriji). Poleg svojega rešeta je izračunal tudi obseg Zemlje, pa oddaljenost Sonca in Lune. Gre za briljantnega znanstvenika, ki je bil prijatelj Arhimeda, enega največjih matematikov v zgodovini. Zavistneži so ga morda prav zaradi tega klicali tudi Beta – bil naj bi drugi najboljši med svojimi sodobniki v vsem. Toda zgodovina je pokazala, da je bil prvak na številnih področjih, ki vključujejo astronomijo, geografijo, literaturo, poezijo, filozofijo in matematiko. Celo 'biti drugi' na tolikih področjih in še v času neverjetnega napredka ga postavlja med največje genije vseh časov.

```
1  n ← 1000;
2  p ← array[1..n];
3  for i ← 1 to n do
4    p[i] ← 0
5  p[1] ← 1;
6  for i ← 2 to n do
7     if p[i] = 0 then
8        for j ← 2 to n / i do
9           p[j * i] ← 1;
```



Psevdo koda Eratostenovega rešeta

Poglejmo si, kako koda deluje. V prvi vrstici povemo, da bomo iskali vsa praštevila do 1000. To število lahko pri pravilno napisani kodi nadomestimo s poljubnim drugim naravnim številom, ki je večje od 1. Nato si v drugi vrstici pripravimo tabelo `p` prvih 1000 naravnih števil. V tretji in četrti vrstici se s *for zanko* sprehodimo po vseh naravnih številih od 1 do `n` in postavimo vse vrednosti v tabeli na 0, kar naj pomeni, da so še neprečrtane[2]. Črtanje števila i v tabeli bomo označili tako, da bomo vrednost `p[i]` postavili na 1. V peti vrstici tako prečrtamo število 1, ki ni ne praštevilo ne sestavljeno število. Nato pa sledimo postopku, ki smo ga opisali zgoraj. Tako se v šesti vrstici s *for zanko* ponovno sprehodimo po vseh naravnih številih od 2 do `n`, saj vnaprej ne vemo, katera števila bodo že prečrtana in katera še ne. To ugotovimo v sedmi vrstici, ko z *if stavkom* preverimo, ali je trenutno število `p[i]` še neprečrtano (`p[i] = 0`), tj. ali je praštevilo. Samo v tem primeru namreč črtamo njegove večkratnike. To naredi *for zanka* v osmi vrstici, ki nas popelje od drugega pa do $\lfloor n/i \rfloor$-tega večkratnika. Samo črtanje večkratnikov opravimo v deveti vrstici, ko postavimo ustrezne elemente tabele na 1. Ko se postopek konča, imamo v tabeli `p` spravljene podatke o tem katera števila so sestavljena (imajo vrednost 1) in katera ne (vrednost 0). Zgornjo kodo lahko seveda zapišemo v poljubnem programskem jeziku in popravimo tako, da uporabnik sam vpiše zgornjo mejo `n`.

Pozoren bralec bo hitro opazil, da se da zgornjo kodo še bistveno izboljšati. V zadnji *for zanki* smo črtali kar vse večkratnike, ne glede na to, ali so bili že črtani ali ne. Ker je branje precej hitrejše od pisanja, lahko z dodatnim *if stavkom*, ki preveri ali je večkratnik že prečrtan, ali ga še moramo, naš program precej pohitrimo (za okoli 27%). Nadalje smo že omenili, da je v *for zanki* v šesti vrstici dovolj gledati le do korena števila `n`. Na ta način prihranimo veliko klicev *if stavka* v sedmi vrstici in s tem pohitrimo naš program še za okoli 37%.

Vsak izkušen programer bi nam tudi povedal, da je deljenje števil v računalniku dosti počasnejše od množenja, ki je tudi počasnejše od seštevanja. Zato bi lahko osmo in deveto vrstico zgornjega programa spremenili tako, da bi začeli z dvakratnikom števila i in na vsakem koraku prištevali i namesto 1. S tem se znebimo tako deljenja kot množenja in pridobimo še 25%.

Nadalje lahko opazimo, da nam v *for zanki* iz osme vrstice ni potrebno začeti pri dvakratniku števila i, ampak lahko začnemo šele pri i-kratniku števila i, torej z i^2. Vsi večkratniki števila i, ki so manjši od i^2, so namreč ravno i-ti večkratniki števil $2, \ldots, i-1$.

[2]Za postavljanje vrednosti v tabeli na 0 ponavadi poskrbi že operacijski sistem.

S tem pridobimo še dodatne 4%. Skupno je naš program sedaj trikrat hitrejši.[3]

Poglejmo si, kako bi izgledal naš program, vključno z vsemi izboljšavami, v programskem jeziku C. Iskal bo vsa praštevila do 100.000.000.

```c
#include <stdio.h>

#define MAX 100000000
#define SQRT 10000

char p[MAX + 1];

int main(void) {
   int i, j;

   for (i = 2; i <= SQRT; i++)
      if (p[i] == 0)
         for (j = i * i; j <= MAX; j += i)
            if (p[j] == 0)
               p[j] = 1;

   return 0;
}
```

Eratostenovo rešeto v programskem jeziku C

Označimo s $p(n)$ število praštevil, ki so manjša ali enaka n. To je ravno število polj v tabeli `p`, katerih vrednost je enaka 1. Pri različnih vrednostih konstante `MAX` dobimo naslednje rezultate:

`MAX`	$p(n)$	$\frac{p(n)\ln(n)}{n}$
100	25	1.151292546
1.000	168	1.160502887
10.000	1.229	1.131950832
100.000	9.592	1.104319811
1.000.000	78.498	1.084489948
10.000.000	664.579	1.071174789
100.000.000	5.761.455	1.061299232

Gotovo ste opazili, da se v tabeli nahaja tudi logaritemska funkcija $\ln n$. V srednji šoli pride ta snov na vrsto v drugem letniku[4]. Si predstavljate, da je Gauss prišel do podobnih rezultatov peš (brez računalnika) in pri šestnajstih letih postavil naslednjo domnevo:

[3]Možne so seveda še nadaljnje izboljšave. Če recimo upoštevamo, da je 2 edino sodo praštevilo, bo program kar štirikrat hitrejši od začetnega.

[4]Ali kdo ve, kako bi iz vrednosti $\log_a b$ dobili dolžino zapisa števila b v številskem sistemu z osnovo a?

za dovolj veliko število n

je število praštevil do števila n

približno $\dfrac{n}{\ln n}$.

Pri tem imejte v mislih, da je bilo v tistih časih že računanje naravnega logaritma povsem netrivialno (oziroma zelo zamudno) opravilo, saj še ni bilo kalkulatorjev. V nekem trenutku so bile na voljo Vegove tablice, kot najmodernejši pripomoček za računanje (kateremu bi danes rekli računalnik) tistega obdobja.

Iz zgornje ocene lahko dobite z osnovnim znanjem o limitah tudi oceno za velikost n-tega praštevila p_n:

$$p_n \approx n \ln n,$$

ki vpelje nekakšen red med dovolj velika praštevila.