

KATEDRA INFORMATIKY
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY
UNIVERZITA KOMENSKÉHO, BRATISLAVA

TESTOVANIE PRVOČÍSELNOSTI
ALGORITMUS AKS
(diplomová práca)

ANDREJ GATIAL

Vedúci: RNDr. Jaroslav Guričan, CSc.

Bratislava, 2005

Čestne prehlasujem, že som túto diplomovú prácu vypracoval samostatne s použitím citovaných zdrojov.

.....

Ďakujem svojmu vedúcemu diplomovej práce, RNDr. Jaroslavovi Guričanovi, CSc. za cenné rady a pripomienky pri písaní tejto práce.
Taktiež by som chcel poďakovať rodine a priateľom za podporu pri štúdiu na FMFI UK.

Obsah

1	Úvod	6
1.1	História testovacích algoritmov	7
1.1.1	Jednoduché testy	7
1.1.2	Novodobé testy	7
1.2	Výpočtová zložitosť	9
1.2.1	Notácie a zložitosť triedy	9
1.2.2	Zložitosť testovacích algoritmov	10
1.3	Teoretické základy	10
1.3.1	Označenia	10
1.3.2	Definície	11
1.3.3	Pomocné lemy a tvrdenia	11
1.3.4	Cyklotomické polynómy	13
1.4	Zložitosť niektorých operácií	16
2	Algoritmus AKS	19
2.1	Myšlienka algoritmu	19
2.2	Algoritmus	20
2.3	Zložitosť algoritmu	26
3	Implementácia	27
3.1	Prostredie PARI	27
3.1.1	Konzolové riešenie	28
3.1.2	Práca s knižnicami PARI	28
3.1.3	Implementácia	29
3.1.4	Výkon	33
3.1.5	Zlepšenia v PARI	36
3.2	Prostredie NTL	38
3.2.1	Implementácia	38

3.2.2	Výkon	42
4	Praktické využitie a budúcnosť	44
4.1	Porovnanie s RSA - Miller-Rabinov test	44
4.2	Možné zlepšenia	45
5	Zhrnutie	47

Kapitola 1

Úvod

Rozdelenie čísel na zložené čísla a prvočísla je známe už od počiatkov matematiky. Z tohto rozdelenia plyní prirodzená otázka, ako určiť, či je číslo prvočíslo alebo nie. Ako prví sa týmto problémom zrejme serióznejšie zaoberali antickí Gréci, najznámejším je Eratostenes a po ňom pomenované Eratostenovo sito. Z novodobejších dejín to boli Marin Mersenne a Pierre de Fermat.

S nástupom moderných šifrovacích techník ako je RSA, sa tento problém stal znova zaujímavým nielen z hľadiska teoretickej matematiky a teórie veľkých čísel, ale aj z praktického použitia dosiahnutých výsledkov. Z toho dôvodu sa stále hľadajú rýchlejšie algoritmy na rozhodnutie, či je dané číslo prvočíslo alebo nie. Vzhľadom na veľkosť používaných prvočísel je dnes celý tento proces nemysliteľný bez zapojenia počítačov.

Cieľom tejto práce je predstaviť najnovší objav z oblasti určovania prvočíselnosti, algoritmus AKS [7], [6]. Algoritmus sa pokúsime implementovať a výsledky porovnáme s existujúcimi a v praxi používanými algoritmami.

Práca je rozdelená na nasledujúce časti:

- Prvá časť obsahuje úvod do problematiky prvočísel, stručnú históriu používaných testov
- V druhej časti predstavíme samotný algoritmus, uvedieme dôkaz správnosti a časový odhad
- V tretej časti sa nachádza naša implementácia s vysvetleniami a zdôvodnením

- V štvrtej časti prezentujeme získané výsledky, porovnáme ich s používaným algoritmom Miller-Rabin.

Prílohou k práci je CD, na ktorom sú zdrojové kódy jednotlivých implementácií, použité knižnice a prostredia spolu s dokumentáciou a návodom na použitie. CD taktiež obsahuje elektronickú verziu tohoto dokumentu.

1.1 História testovacích algoritmov

1.1.1 Jednoduché testy

Veľmi jednoduchý a priamočiary test nám ponúka už samotná definícia prvočísla. Stačí overiť, či n má práve dvoch rôznych deliteľov, jednotku a seba samého. Aby sme to overili, stačí deliť n postupne každým číslom $m \leq \sqrt{n}$. Ak niektoré m delí n , tak testované n je zložené prvočíslo, v opačnom prípade je to prvočíslo.

Mierne upravená verzia tohto testu je známa ako *Eratostenovo sito*, ktoré generuje všetky prvočísla $\leq n$. Princíp je jednoduchý, zoberieme všetky nepárne čísla $2 \dots n$ (aj 2) a postupne budeme vyškrtávať čísla podľa nasledujúcich pravidiel:

1. najmenšie nevyškrtnuté číslo m označíme ako prvočíslo
2. vyškrtáme všetky násobky m , opakujeme krok 2

Čísla, ktoré ostali, sú prvočísla.

1.1.2 Novodobé testy

Nakoľko vyššie uvedené testy sa dajú prakticky použiť len pre malé čísla (v prípade, ak počítame ručne, len s kalkulačkou a na papier, tak len pre veľmi malé čísla, rádovo do tisíc), bolo potrebné vymyslieť lepšie postupy.

Základom týchto testov sa stalo rozhodovanie podľa známych vlastností prvočísla a nie podľa definície. Jednu z týchto vlastností popisuje **malá Fermatova veta**:

Veta 1.1.1 Pre prvočíslo p a ľubovoľné číslo a také, že $(p, a) = 1$ platí $a^{p-1} = 1 \pmod{p}$.

To nám poskytuje jednoduchý a rýchly test, pre dané p zvolíme a a overíme podmienku. Tento test nie je však korektný, pretože niektoré zložené čísla pre niektoré hodnoty a týmto testom prejdú, dokonca v prípade *Carmichaelových čísel* pre ľubovoľné a . Aj napriek tomuto nedostatku sa táto vlastnosť stala základom mnohých efektívnych testov.

Ak číslo prejde testom pre nejaké a , označíme ho za možného kandidáta na prvočíslo pri základe a (probable prime base a , **a-PRP**). Zložené PRP je pseudoprvočíslo. Príkladom najmenších pseudoprvočísel sú tieto (pre rôzne základy) [1]:

341 = 11 · 31 je 2-PRP
 91 = 7 · 13 je 3-PRP
 271 = 7 · 31 je 5-PRP
 25 = 5 · 5 je 7-PRP

Zlepšenie Fermatovho testu dosiahneme ak si uvedomíme, že ak n nepárne je prvočíslo, potom 1 má práve dve odmocniny modulo n , tie sú 1 a -1 . Teda odmocnina z a^{n-1} bude 1 alebo -1 . Celkový test potom vyzerá nasledovne [1]:

$n - 1 = 2^s \cdot d$, d je nepárne, s je nezáporné
 ak $a^d = 1 \pmod{n}$ alebo $(a^d)^{2^r} = -1 \pmod{n}$, nezáporné $r < s$
 potom n je silné PRP pri základe a (strong PRP, a-SPRP)

Znovu príklad najmenších SPRP podľa [1]:

2047 = 23 · 89 je 2-SPRP
 121 = 11 · 11 je 3-SPRP
 781 = 11 · 71 je 5-SPRP
 25 = 5 · 5 je 7-SPRP

Skombinovaním viacerých báz sa znižuje pravdepodobnosť omylu. Dá sa ukázať, že ak n je SPRP pre určitú kombináciu báz a n je menšie ako určená hranica (podľa báz) potom n je prvočíslo.

Najznámejšie dnes používané testy sú Miller-Rabinov test a Solovay-Strassenov test. Oba sú pravdepodobnostné "Monte Carlo" testy.

V RSA sa používa Miller-Rabinov test. Má tú vlastnosť, že si vieme určiť pravdepodobnosť omylu pozitívneho výsledku. Ak test označí n ako zložené číslo, sme si istí, že je to správne, ak označí n ako prvočíslo, vieme s akou pravdepodobnosťou má pravdu. Pravdepodobnosť správnej odpovede vieme

ľubovoľne priblížiť k jednej, avšak nikdy to nebude 100%. Viac o tomto teste je v kapitole 4.

Z iných algoritmov spomenieme ešte algoritmus založený na eliptických krivkách, alebo algoritmy, ktoré využívajú k rozhodovaniu informácie o vlastnostiach $p + 1$ alebo $p - 1$, ak sa tieto vlastnosti dajú ľahšie určiť.

1.2 Výpočtová zložitosť

1.2.1 Notácie a zložitosť triedy

Aby bol algoritmus považovaný za rýchly, vyžadujeme, aby požadový počet krokov bol polynóm od veľkosti vstupu. Teda, ak je vstup veľkosti n , chceme, aby algoritmus potreboval vždy

$$a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$$

alebo menej krokov. Aby sme nemuseli vypisovať takýto polynóm, zaviedla sa O notácia, vyjadrujúca asymptotické ohraničenie zhora:

$$O(g(n)) = \{f(n) : \exists \text{ kladné konštanty } c \text{ a } n_0 \mid 0 \leq f(n) \leq cg(n) \forall n \geq n_0\}$$

Potom, ak funkcia $f(n)$ patrí do množiny $O(g(n))$, zapisujeme to $f(n) = O(g(n))$.

Podobne sa zaviedla Ω notácia, vyjadrujúca asymptotické ohraničenie zdola:

$$\Omega(g(n)) = \{f(n) : \exists \text{ kladné konštanty } c \text{ a } n_0 \mid 0 \leq cg(n) \leq f(n) \forall n \geq n_0\}$$

Požiadavku na rýchly algoritmus môžeme potom vyjadriť tak, že algoritmus potrebuje $O(n^k)$ krokov, kde k je nezávislé od n .

V teórii výpočtovej zložitosti sa zaviedli triedy zložitosti **P** a **NP**. Algoritmy v triede **P** sú deterministické a polynomiálne, tj. vždy vrátia presnú správnu odpoveď (deterministický) a čas behu je polynóm $O(n^k)$, kde n je veľkosť vstupu.

Veľkosťou vstupu rozumieme zápis čísla, pre potreby algoritmov sa pracuje s binárnym zápisom čísla n , tj. veľkosť vstupu n je $\log_2 n$. V nasledujúcom $\log n = \log_2 n$.

1.2.2 Zložitosť testovacích algoritmov

Priame delenie, resp. Eratostenovo sito potrebuje vykonať $\Omega(\sqrt{n})$ krokov, kde n predstavuje hodnotu vstupu, nie veľkosť, čím nespĺňa požiadavku polynomiálnosti.

Miller-Rabinov test je polynomiálny ($O(r \cdot m^3)$, kde m je počet bitov n a r je konštanta súvisiaca so zvolenou presnosťou), ale nie je deterministický. Rovnako Solovay-Strassenov test je polynomiálny, ale nie je deterministický.

Algoritmom, ktorý je deterministický je APR (Adleman, Pomerance, Rumely), ktorý však nie je polynomiálny, jeho zložitosť je $(\log n)^{O(\log \log \log n)}$, je teda skoro polynomiálny.

Jeden z algoritmov založených na eliptických krivkách, Goldwasser-Kilian, je deterministický, ale na niektorých vstupoch nekončí v polynomiálnom čase.

V roku 2002 sa podarilo trom indickým matematikom Agrawal, Kayal, Saxena vytvoriť algoritmus, ktorý beží v čase $O(\log^{10.5} n)$, teda polynomiálny, a je tiež deterministický. Týmto definitívne ukázali, že problém prvočíselnosti patrí do zložitosťnej triedy **P**. Algoritmus je v porovnaní s inými deterministickými algoritmami veľmi jednoduchý a na jeho dôkaz postačujú ďaleko jednoduchšie nástroje algebry.

1.3 Teoretické základy

1.3.1 Označenia

Znakom (a, n) označujem najväčší spoločný deliteľ čísel a, n . $\log n$ znamená $\log_2 n$ ak nie je povedané inak.

\mathcal{N} a \mathcal{Z} označujú množinu prirodzených a celých čísel. Ak máme $r \in \mathcal{N}$ a $a \in \mathcal{Z}$ také, že $(a, r) = 1$, potom *rádom a modulo r* nazveme také najmenšie k , že $a^k = 1 \pmod{r}$. Označíme to $o_r(a)$.

F_p označuje konečné pole s p prvkami, kde p je prvočíslo. Ak je p prvočíslo a $h(X)$ je ireducibilný polynóm v F_p stupňa d , potom $F_p[X]/(h(X))$ je konečné pole rádu p^d .

$\phi(r)$, pre $r \in \mathcal{N}$ je Eulerova funkcia určujúca počet čísel menších ako r , ktoré sú nesúdeliteľné s r . Pre $a, (a, r) = 1$ platí $o_r(r) \mid \phi(r)$.

Symbol $O^\sim(f(n))$ označuje $O(f(n) \cdot \text{poly}(\log f(n)))$ kde $f(n)$ je funkcia n . V našom prípade potom $O^\sim(\log^k n) = O(\log^k n \cdot \text{poly}(\log \log n)) = O(\log^{k+\epsilon})$ pre $\epsilon > 0$.

1.3.2 Definície

Definícia 1.3.1 *Prirodzené číslo $n > 1$ je **prvočíslo** práve vtedy, keď jeho jediní prirodzení delitelia sú 1 a n .*

Definícia 1.3.2 *Zložené prirodzené číslo n sa nazýva **Carmichaelovo číslo** ak $a^{n-1} = 1 \pmod{n}$ pre všetky a , $(a, n) = 1$.*

Definícia 1.3.3 *Číslo spĺňajúce malú Fermatovu vetu pre nejaké a nazývame **pravdepodobné prvočíslo pri základe a (a-PRP)**.*

Definícia 1.3.4 *Zložené PRP číslo sa nazýva **pseudoprvočíslo**.*

Definícia 1.3.5 *Číslo n , zapísané tak, že $n-1 = 2^s \cdot d$, kde d je nepárne, s je nezáporné, ak $a^d = 1 \pmod{n}$ alebo $(a^d)^{2^r} = -1 \pmod{n}$ pre nezáporné $r < s$ sa nazýva **silné PRP pri základe a (a-SPRP)**.*

1.3.3 Pomocné lemy a tvrdenia

Malá Fermatova veta

Veta 1.3.1 ([1]) *Pre prvočíslo p a ľubovoľné číslo a také, že $(p, a) = 1$ platí $a^{p-1} = 1 \pmod{p}$.*

Dôkaz.

Zoberme násobky a : $a, 2a, 3a, \dots, (p-1)a$. Nech $r \cdot a$ a $s \cdot a$ sú rovnaké modulo p . Máme $r = s \pmod{a}$, teda uvedených $(p-1)$ násobkov a je rôznych a nenulových a teda sú kongruentné s $1, 2, 3, \dots, p-1$ v nejakom poradí. Ak vynásobíme tieto kongruencie dostaneme

$$a \cdot 2a \cdot 3a \dots (p-1)a = 1 \cdot 2 \cdot 3 \dots (p-1) \pmod{p}$$

po vyňatí a na ľavej strane a úprave dostaneme

$$a^{p-1}(p-1)! = (p-1)! \pmod{p}$$

Po vydelení oboch strán výrazom $(p-1)!$ dostávame požadované tvrdenie. \square

Algebra

Lema 1.3.1 ([7]) Označme najmenší spoločný násobok prvých m čísel $LCM(m)$.
Pre $m \geq 7$ platí

$$LCM(m) \geq 2^m$$

Lema 1.3.2 (Cvičenie 5 v [4]) Polynóm $X - 1$ delí $X^r - 1$.

Dôkaz

$$\begin{aligned} X^r - 1 &= (X^r - X^{r-1}) + (X^{r-1} - X^{r-2}) + (X^{r-2} - X^{r-3}) + \dots + (X - 1) \\ &= X^{r-1}(X - 1) + X^{r-2}(X - 1) + \dots + (X - 1) \end{aligned}$$

Každý člen je deliteľný $X - 1$. □

$(Z_p, +, \cdot)$ je pole práve vtedy keď p je prvočíslo. $Z_p^* = (Z_p - \{0\}, \cdot)$ je grupa, $|Z_p^*| = p - 1$, je to multiplikatívna grupa.

Veta 1.3.2 Ak F je konečné pole, tak jeho multiplikatívna grupa (F^*, \cdot) je cyklická.

Lema 1.3.3 Pre $p \geq 2$

$$p^k - 1 \mid p^s - 1 \Leftrightarrow k \mid s$$

Dôkaz

Nech k delí s so zvyškom, tj. $s = k \cdot l + r$ (r je zvyšok po delení, $r < k$).
Potom

$$p^s - 1 = p^{k \cdot l + r} - 1 = p^r \cdot p^{k \cdot l} - p^r + p^r - 1 = p^r(p^{k \cdot l} - 1) + p^r - 1$$

Keďže $p^k - 1 \mid p^s - 1$ a $p^k - 1 \mid p^{k \cdot l} - 1$, tak musí aj $p^k - 1 \mid p^r - 1$. To však nie je možné, pretože $p^r - 1$ je malé ($r < k$), možné je to len ak $r = 0$. Ak $r = 0$ potom $k \mid s$.

Opačná implikácia vyplýva z lemy 1.3.2. □

Veta 1.3.3 Nech $(O, +, \cdot)$ je komutatívny okruh s jednotkou charakteristiky p . Takýto okruh je zároveň vektorový priestor nad Z_p . Nech $T : O \rightarrow O$ je zobrazenie dané $T(a) = a^p$. Potom

1. T je okruhový homomorfizmus

2. $T : O(Z_p) \rightarrow O(Z_p)$ je lineárne zobrazenie ($O(Z_p)$ je tu vektorový priestor).

Dôkaz

1. pre sčítanie: $(a + b)^p = \sum_{i=0}^p \binom{p}{i} (a^{p-i} b^i) = a^p + b^p$, pretože koeficienty $\binom{p}{1}, \dots, \binom{p}{p-1}$ sú deliteľné p . Pretože charakteristika okruhu je p , je každý koeficient $\binom{p}{i} (a^{p-i} b^i)$ nulový pre $1 \leq i \leq p-1$. Pre násobenie to vyplýva z komutatívnosti

2. linearita: treba ukázať, že pre $a \in Z_p$, $u \in O$ je $(a \cdot u)^p = a \cdot u^p$. Tvrdenie vyplýva z malej Fermatovej vety (1.3.1), ktorá hovorí, že pre $a \in Z_p$ platí $a^p = a$. \square

Dôsledok 1 $(Z_p[X], +, \cdot)$ je tiež okruh spĺňajúci predpoklady vety. Potom pre ľubovoľný polynóm $u(X) \in Z_p[X]$ platí rovnosť $u(X)^p = u(X^p)$.

1.3.4 Cyklotomické polynómy

Definícia 1.3.6 [11] ζ je n -tá odmocnina z jednotky ak $\zeta^n = 1$. ζ je n -tá primitívna odmocnina z jednotky ak n je najmenšie také, že $\zeta^n = 1$.

Definícia 1.3.7 Pre ľubovoľné n , je n -tý cyklotomický polynóm Q_n nad poľom $(F, +, \cdot)$ definovaný:

$$Q_n(X) = (X - \zeta_1)(X - \zeta_2) \dots (X - \zeta_k)$$

kde ζ_1, \dots, ζ_k sú všetky n -té primitívne odmocniny z jednotky nad poľom $(F, +, \cdot)$. Špeciálne pre n prvočíslo, n -tý cyklotomický polynóm Q_n je potom:

$$Q_n(X) = \frac{X^n - 1}{X - 1} = X^{n-2} + X^{n-3} + \dots + X + 1$$

Keď zoberieme n také, že $(p, n) = 1$, tak rovnica $X^n - 1$ má v Z_p n koreňov, ktoré sú jednoduché, lebo $(D(X^n - 1), X^n - 1) = (n \cdot X^{n-1}, X^n - 1) = 1$, kde D je derivácia, lebo X^{n-1} nedelí $X^n - 1$.

Tieto korene tvoria grupu s n prvkami a ak ζ je n -tá primitívna odmocnina z jednotky, tak $1, \zeta, \zeta^2, \dots, \zeta^{n-1}$ sú všetky prvky tejto grupy, tj. medzi nimi sa nájdu všetky n -té primitívne odmocniny z jednotky. Pre prvok ζ^i platí, že je primitívna odmocnina z jednotky $\Leftrightarrow (i, n) = 1$, teda počet primitívnych odmocnín z jednotky je $\phi(n)$.

Potom stupeň $st(Q_n(X)) = \phi(n)$ a nezávisí od p pre $(n, p) = 1$. Špeciálne ak r je prvočíslo, $\phi(r) = r - 1$ a $st(Q_r(X)) = r - 1$, a preto $Q_r(X) = \frac{X^r - 1}{X - 1}$,

lebo jednotka je jediný koreň $X^r - 1$, ktorý nemá rád r ; preto sme $X^r - 1$ vydělili koreňovým činiteľom $X - 1$.

Ďalej nás bude zaujímať r -tý cyklotomický polynóm $Q_r(X)$ ak r je prvočíslo. V nasledujúcom sú $r \neq p$ prvočísla.

Veta 1.3.4 *Nech $h(X)$ je ireducibilný deliteľ $\frac{X^r-1}{X-1}$ v $Z_p[X]$, nech je jeho stupeň k . Potom $k = o_r(p)$ a $X \in Z_p[X]/(h(X))$ je primitívna r -tá odmocnina z jednotky v tomto poli.*

Dôkaz

Ak $h(X)$ je ireducibilný deliteľ $\frac{X^r-1}{X-1}$ stupňa k , potom $Z_p[X]/(h(X))$ je pole, ktoré má p^k prvkov, jeho multiplikatívna grupa má $p^k - 1$ prvkov. Ak ζ je koreň $h(X)$, je to jedna z r -tých primitívnych odmocnín jednotky a teda rád ζ je r . Pretože rád prvku delí rád grupy do ktorej patrí, máme $r \mid p^k - 1$, čo je $p^k \equiv 1 \pmod{r}$.

Treba ukázať, že k je najmenšie, ktoré spĺňa túto kongruenciu, tj. ak iné s ju spĺňa tiež ($p^s \equiv 1 \pmod{r}$) potom $k \mid s$.

$p^s \equiv 1 \pmod{r} \Leftrightarrow p^s = 1 + a \cdot r$, nech $X^r - 1 = h(X) \cdot g(X) \Rightarrow X^r = 1 + h(X) \cdot g(X)$. Po dosadení a úprave

$$\begin{aligned} X^{p^s} &= X \cdot X^{a \cdot r} \\ &= X \cdot (1 + h(X) \cdot g(X))^a \\ &= X \cdot \left(1 + \binom{a}{1} h(X) \cdot g(X) + \binom{a}{2} h^2(X) \cdot g^2(X) + \dots\right) \\ &= X \cdot (1 + h(X) \cdot f(X)) \\ &= X + X \cdot h(X) \cdot f(X) \end{aligned}$$

vidieť, že $X^{p^s} - X = X \cdot h(X) \cdot f(X) \Rightarrow h(X) \mid X^{p^s} - X$, teda $X^{p^s} \equiv X \pmod{h(X)}$.

Grupa $(Z_p[X]/(h(X)))^*$ je cyklická, nech jej generátor je $a(X)$. Podľa dôsledku 1 vieme, že $a(X)^p = a(X^p)$. Opakovaným aplikovaním tohto dôsledku ($Y = X^p, a(X)^{p^2} = (a(X)^p)^p = (a(X^p))^p = a(Y)^p = a(Y^p) = a(X^{p^2})$) dostaneme $a(X)^{p^s} = a(X^{p^s})$. Pretože $X^{p^s} \equiv X \pmod{h(X)}$ potom $a(X)^{p^s} = a(X^{p^s}) = a(X)$. Po vydelení $a(X)$ (pretože $a(X)$ má v $(Z_p[X]/(h(X)))^*$ inverzný prvok) dostaneme $a(X)^{p^s-1} = 1$, čo hovorí, že rád $a(X)$ delí $p^s - 1$. Pretože $a(X)$ je generátor, jeho rád je $p^k - 1$. Teda $p^k - 1 \mid p^s - 1$, z toho podľa lemy 1.3.3 dostaneme $k \mid s$.

Ukázali sme, že $p^k = 1 \pmod{r}$ a zároveň k je najmenšie s takou vlastnosťou. Spolu je to definícia $o_r(p)$.

Pretože $h(X) \mid X^r - 1$ potom rád prvku X v $(\mathbb{Z}_p[X]/(h(X)))^*$ delí r , ale keďže r je prvočíslo, rád X musí byť rovný r a preto je X r -tá primitívna odmocnina z jednotky. \square

Vzťahy kombinačných čísel

Lema 1.3.4 Pre $k \geq 3$ platí $\binom{2k-1}{k} \geq 2^k$.

Dôkaz

$$\begin{aligned} \binom{2k-1}{k} &= \frac{(2k-1)(2k-2)\dots(2k-1-(k+2))(2k-1-(k+1))}{k(k-1)(k-2)\dots 1} \\ &= \frac{2k-1}{k-1} \cdot \frac{2k-2}{k-2} \cdots \frac{k+1}{1} \cdot \frac{k}{k} \\ &\geq 2^{k-2} \cdot \frac{k+1}{1} \geq 2^k \end{aligned}$$

pretože pre $0 \leq x < 2k$ platí

$$2k - x \geq 2(k - x) \Rightarrow \frac{2k - x}{k - x} \geq 2$$

a pre $k \geq 3$ je $\frac{k+1}{1} \geq 4$. \square

Lema 1.3.5 Pre $k \geq 0$, $a \geq b$ platí $\binom{a+k}{b+k} \geq \binom{a}{b}$

Dôkaz

Tvrdenie dokážeme indukciou na k . Pre $k = 0$ tvrdenie platí, pre $k = 1$

$$\binom{a+1}{b+1} = \frac{(a+1)!}{(b+1)!(a+1-b-1)!} = \frac{(a+1) \cdot a!}{(b+1) \cdot b! \cdot (a-b)!} = \frac{a+1}{b+1} \cdot \binom{a}{b}$$

Pretože $a \geq b \Rightarrow \frac{a+1}{b+1} \geq 1$, tak celý výraz $\geq \binom{a}{b}$.

Prepokladajme teraz, že tvrdenie platí pre nejaké $k \geq 1$, $\binom{a+k}{b+k} \geq \binom{a}{b}$. Potom

$$\begin{aligned} \binom{a+(k+1)}{b+(k+1)} &= \frac{(a+k+1)!}{(b+k+1)!(a-b)!} \\ &= \frac{(a+k+1)(a+k)!}{(b+k+1)(b+k)!(a-b)!} \end{aligned}$$

$$\begin{aligned}
&= \frac{a+k+1}{b+k+1} \cdot \binom{a+k}{b+k} \\
&\geq \frac{a+k+1}{b+k+1} \cdot \binom{a}{b} \\
&\geq \binom{a}{b}
\end{aligned} \tag{1.1}$$

Nerovnosť 1.1 platí z indukčného predpokladu $\binom{a+k}{b+k} \geq \binom{a}{b}$, posledná nerovnosť platí, pretože uvedený zlomok je väčší ako 1, keď $a \geq b$. Tým sme ukončili indukčný krok. \square

Lema 1.3.6 Pre $k \geq 0$, $a \geq b$ platí $\binom{a+k}{b} \geq \binom{a}{b}$

Dôkaz

$$\begin{aligned}
\binom{a+k}{b} &= \frac{(a+k)!}{b!(a+k-b)!} \\
&= \frac{(a+k)(a+k-1)\dots(a+1)a!}{b!(a+k-b)(a+k-1-b)\dots(a+1-b)(a-b)!} \\
&= \frac{a+k}{(a-b)+k} \cdot \frac{a+k-1}{(a-b)+k-1} \cdots \frac{a+1}{(a-b)+1} \cdot \binom{a}{b}
\end{aligned}$$

Pretože $b \geq 0 \Rightarrow a \geq a-b \Rightarrow \frac{a+x}{(a-b)+x} \geq 1$ pre $x \geq 0$, tak celý výraz $\geq \binom{a}{b}$. \square

1.4 Zložitosť niektorých operácií

Ako sme už uviedli, všetky čísla sú reprezentované binárnym zápisom. V nasledujúcom uvažujeme čísla s binárnym zápisom dĺžky m .

Sčítanie, odčítanie

Je vykonávané po bitoch s prenosom do vyššieho rádu. Na sčítanie dvoch m -bitových čísel treba $O(m)$ krokov. Výsledok má veľkosť maximálne $m+1$ bitov, pretože jeho maximálna hodnota je $2 \cdot 2^m = 2^{m+1}$. Odčítanie sa realizuje podobne.

Násobenie

Je vykonávané po bitoch s posunom a pričítaním. Ilustračný príklad: $4 \cdot 6 = 24 = (100)_2 \cdot (110)_2$, čo je $0 \cdot 100 + 1 \cdot 1000 + 1 \cdot 10000 = 11000 = (24)_{10}$. Je vidieť, že na vynásobenie dvoch m -bitových čísel treba m sčítaní a m násobení jednotkou alebo nulou, na tieto násobenia je treba konštantný čas. Celkový čas je čas potrebný na m sčítaní m -bitových čísel, teda $O(m \cdot m) = O(m^2)$.

S použitím FFT (Fast Fourier Transformation - spôsob násobenia veľkých čísel) sa dá dosiahnuť čas $O^\sim(m)$ ako sa uvádza v [7].

Delenie

Je vykonávané po bitoch s posunom a odčítaním. Vyžaduje si $O(m^2)$ krokov.

Pododne ako násobenie, aj pre delenie je možné dosiahnuť pomocou FFT čas $O^\sim(m)$.

Modulárna aritmetika - $a^b \pmod n$

Výraz sa vypočíta opakovaným umocnením, podľa [10]. Binárny rozvoj b je $b_m b_{m-1} \dots b_0$, kde b je m -bitové číslo.

```
d = 1
for i = m to 0 do
  d = (d * d) mod n
  if b[i] = 1 then d = (d * a) mod n
return d
```

Na vynásobenie v $(d * d)$, resp. $(d * a)$ potrebujeme $O(m^2)$, cyklus sa opakuje m -krát, potom čas behu procedúry je $O(m^3)$ ak a , b sú najviac m -bitové čísla. Tento postup možno aplikovať aj na polynómy.

Ak sa použije FFT, čas bude $O^\sim(m^2)$.

Najväčší spoločný deliteľ - GCD

Z Euklidovho algoritmu vieme, že $\gcd(a, b) = \gcd(b, a \pmod b)$. Podľa [10] existuje implementácia Euklidovho algoritmu, ktorá vráti (d, x, y) také, že $d = \gcd(a, b) = ax + by$, $|x| \leq b$ a $|y| \leq a$.

```
procedure Euklid(a, b)
  if b = 0 then return (a, 1, 0)
```

```

    (d, z, x) = Euklid(b, a mod b)
    y = z - floor(a / b) * x
    return (d, x, y)
end

```

Dá sa ukázať, že počet rekurzívnych volaní je $O(m)$, lebo počas každých dvoch volaní sa hodnota prvého parametra zmenší aspoň na polovicu. Výpočty v rámci jedného volania sú násobenie a delenie, ktoré je možné vykonať v $O(m^2)$ a teda celkový čas výpočtu je $O(m^3)$ ak a, b sú najviac m -bitové čísla, $a > 0, b \geq 0$.

Určenie a^b

Majme $n = a^b$, $a, b, n \in \mathcal{N}$. Je zrejmé, že $c = \sqrt[b]{n}$ je tiež z \mathcal{N} a $c = a$ a teda $\lfloor \sqrt[b]{n} \rfloor^b = \sqrt[b]{n^b} = n$. Ďalej vidieť, že $b \leq \log_2 n$, inak by $a^b \geq 2^b > 2^{\log_2 n} = n$. Teda treba overiť, či n nie je nejaká i -ta mocnina pre $1 < i \leq \lfloor \log_2 n \rfloor$.

1. for $k = 1$ to $\log n$ do
2. if $\lfloor 2^{\frac{\log n}{k}} \rfloor^k = n$ then
3. return MOCNINA
4. return NIE MOCNINA

Tento test potrebuje $O(\log n \cdot f(n))$ krokov, $f(n)$ je zložitost výpočtu $\lfloor 2^{\frac{\log n}{k}} \rfloor^k$. Pri použití násobenia s FFT je celkový čas $O(\log^3 n)$, bez FFT je čas $O(\log^4 n)$.

Kapitola 2

Algoritmus AKS

2.1 Myšlienka algoritmu

Algoritmus AKS je založený na nasledovnej identite pre prvočísla, ktorá je zovšeobecneným malej Fermatovej vety (1.3.1). Táto identita už bola použitá ako základ niektorých pravdepodobnostných algoritmov.

Lema 2.1.1 *Nech $a \in \mathcal{Z}$, $n \in \mathcal{N}$, $n \geq 2$, $(a, n) = 1$. Potom n je prvočíslo práve vtedy, keď*

$$(X + a)^n = X^n + a \pmod{n} \quad (2.1)$$

Dôkaz

Pre $0 < i < n$ koeficient pri x^i vo výraze $((X + a)^n - (X^n + a))$ je rovný $\binom{n}{i}a^{n-i}$.

Prepokladajme, že n je prvočíslo. Potom

$$\binom{n}{i} = \frac{n!}{i!(n-i)!} = \frac{n(n-1)\dots(n-i+1)}{i(i-1)\dots 1} = 0 \pmod{n}$$

a preto sú všetky koeficienty rovné nule. Je to špeciálny prípad dôsledku 1 pre polynóm $u(X) = X + a$.

Predpokladajme, že n je zložené číslo. Zoberme prvočíslo q , ktoré je faktorom n a nech $q^k \mid n$ a q^{k+1} nedelí n . Potom q^k nedelí $\binom{n}{q}$, pretože potom by q^{k+1} delilo n . Ďalej máme $1 = (a, n) = (a, q) = (a^{n-q}, q^k)$. Potom koeficient pri X^q (tj. $(-1)^q \binom{n}{q} a^{n-q}$) nie je rovný nula $(\text{mod } n)$. Preto $((X + a)^n - (X^n + a))$ nie je identicky rovné nule nad Z_n . \square

Hore uvedená identita poskytuje jednoduchý test na prvočíselnosť: pre daný vstup n , vyber a a otestuj, či je splnená rovnosť 2.1. Avšak na vyhodnotenie rovnice potrebujeme čas $\Omega(n)$, pretože potrebujeme vyhodnotiť n koeficientov. Jednoduchý spôsob, ako zredukovať počet koeficientov je vyhodnocovať obe strany rovnice 2.1 modulo polynóm tvaru $X^r - 1$ pre vhodne zvolené malé r . Uvedená rovnosť bude mať potom tvar

$$(X + a)^n = X^n + a \pmod{X^r - 1, n} \quad (2.2)$$

Z lemy 2.1.1 vyplýva, že všetky prvočísla spĺňajú aj rovnicu 2.2 pre všetky hodnoty a a r . Objavuje sa však problém, že teraz aj niektoré zložené čísla spĺnia uvedenú rovnicu pre niekoľko málo hodnôt r a a . Ak pre vhodne zvolené r je rovnica 2.2 splnená pre niekoľko a , potom testované n je mocnina prvočísla.

Ukážeme, že počet hodnôt a a vhodné r sú obe polynomiálne ohraničené od $\log n$, čím dostaneme deterministický polynomiálny algoritmus.

2.2 Algoritmus

Vstup: $n > 1$

1. If $(n = a^b \text{ pre } a \in \mathcal{N}, b > 1)$, výsledok ZLOŽENÉ
2. Nájdi najmenšie r také, že $o_r(n) > 4 \log^2 n$
3. If $1 < (a, n) < n$ pre nejaké $a \leq r$, výsledok ZLOŽENÉ
4. If $n \leq r$, výsledok PRVOČÍSLO
5. For $a = 1$ to $\lfloor 2\sqrt{\phi(r)} \log n \rfloor$ do
 if $((X + a)^n \neq X^n + a \pmod{X^r - 1, n})$, výsledok ZLOŽENÉ
6. výsledok PRVOČÍSLO

Uvedieme dôkaz správnosti ako je podaný v [6], novšia verzia [7].

Veta 2.2.1 *Algoritmus vráti PRVOČÍSLO vtedy a len vtedy keď je n prvočíslo.*

Najprv dokážeme niekoľko pomocných tvrdení, ktoré použijeme v dôkaze vety.

Lema 2.2.1 *Ak n je prvočíslo, algoritmus vráti PRVOČÍSLO.*

Dôkaz.

Ak n je prvočíslo, tak kroky 1 a 3 nikdy nemôžu vrátiť ZLOŽENÉ. Podľa lemy 2.1.1 vieme, že **for** cyklus v kroku 5 tiež nemôže vrátiť ZLOŽENÉ. Preto algoritmus identifikuje n ako PRVOČÍSLO a to buď v kroku 4 alebo v kroku 6. \square

Opačným smerom je dôkaz zložitejší. Ak algoritmus vráti PRVOČÍSLO v kroku 4, potom n musí byť prvočíslo, inak by krok 3 našiel netriviálneho deliteľa n . Ostáva jediné miesto, kde algoritmus môže vrátiť PRVOČÍSLO, a to v kroku 6. Ďalej prepokladajme, že sa tak udeje.

Najprv ukážeme ohraničenie na vhodné r .

Lema 2.2.2 *Existuje $r \leq \lceil 16 \log^5 n \rceil$ také, že $o_r(n) > 4 \log^2 n$.*

Dôkaz.

Označme r_1, r_2, \dots, r_t všetky čísla také, že $o_{r_i} \leq 4 \log^2 n$. Podľa definície $o_r(n)$ máme $n^{k_i} - 1 = 0 \pmod{r_i}$, $k_i \leq 4 \log^2 n$, teda každé $r_i \mid n^{k_i} - 1$ a celkovo každé r_i delí

$$\prod_{i=1}^{\lfloor 4 \log^2 n \rfloor} (n^i - 1)$$

Označíme $H = \lfloor 4 \log^2 n \rfloor$

$$\prod_{i=1}^H (n^i - 1) < \prod_{i=1}^H n^i = n^{\sum_{i=1}^H i} = n^{\frac{H(H+1)}{2}} \leq n^{H^2} = n^{16 \log^4 n} \leq 2^{16 \log^5 n}$$

Kedže všetky r_i delia $\prod_{i=1}^H (n^i - 1)$, ten je aspoň taký ako ich najmenší spoločný násobok. Celý výraz je však $<$ ako $2^{16 \log^5 n}$ a to je podľa 1.3.1 $\leq LCM(\lceil 16 \log^5 n \rceil)$. Z toho vyplýva, že

$$\{r_1 \dots r_t\} \neq \{1 \dots \lceil 16 \log^5 n \rceil\} \quad (2.3)$$

inak by sme mali spor medzi uvedenými najmenšími násobkami. Z 2.3 vyplýva, že existuje číslo $r \leq \lceil 16 \log^5 n \rceil$ také, že $o_r(n) \geq 4 \log^2 n$. \square

Nech p je prvočíselný deliteľ čísla n . Vieme, že $p \geq r$, lebo inak by krok 3 alebo 4 určil prvočíselnosť n . Kedže $(n, r) = 1$ (inak by kroky 3 alebo 4 určili n), $p, n \in Z_r^*$. V ďalšom sú čísla p, n pevné, položíme $l = \lfloor 2\sqrt{\phi(r)} \log n \rfloor$.

Krok 5 overuje l rovníc. Kedže algoritmus nevráti ZLOŽENÉ v tomto kroku, dostávame

$$(X + a)^n = X^n + a \pmod{X^r - 1, n}$$

pre každé a , $1 \leq a \leq l$. Z toho vyplýva

$$(X + a)^n = X^n + a \pmod{X^r - 1, p} \quad (2.4)$$

pre $1 \leq a \leq l$. Podľa lemy 2.1.1

$$(X + a)^p = X^p + a \pmod{X^r - 1, p} \quad (2.5)$$

pre každé $1 \leq a \leq l$. Teda n sa v uvedenej rovnici správa ako prvočíslo p . Túto vlastnosť pomenujeme:

Definícia 2.2.1 *Pre daný polynóm $f(X)$ a číslo $m \in N$ hovoríme, že m je introspektívne pre $f(X)$ ak*

$$[f(X)]^m = f(X^m) \pmod{X^r - 1, p}$$

Z rovníc 2.4 a 2.5 vidieť, že obidve p aj n sú introspektívne pre $X + a$ pre $1 \leq a \leq l$. Introspektívne čísla sú uzavreté na násobenie, a pre dané m je aj množina polynómov, pre ktoré je introspektívne, uzavretá na násobenie. Tieto dve vlastnosti dokážeme v nasledujúcich tvrdeniach.

Lema 2.2.3 *Ak m a m' sú introspektívne čísla pre $f(X)$ potom aj $m \cdot m'$ je introspektívne.*

Dôkaz.

Z introspektivity m pre $f(X)$ máme:

$$[f(X)]^{m \cdot m'} = [f(X^m)]^{m'} \pmod{X^r - 1, p}$$

Položíme $X^m = Y$ a z introspektivity pre m' dostaneme:

$$[f(X^m)]^{m'} = [f(Y)]^{m'} = f(Y^{m'}) \pmod{Y^r - 1, p}$$

Spätným dosadením za Y

$$[f(Y)]^{m'} = [f(X^{m \cdot m'})] \pmod{X^{r \cdot m'}, p} = f(X^{m \cdot m'}) \pmod{X^r - 1, p}$$

Posledná rovnosť platí, pretože $X^r - 1$ delí $X^{rm} - 1$ (podľa lemy 1.3.2, kde za X dosadíme X^r). Spojením rovníc dostávame

$$[f(X)]^{m \cdot m'} = f(X^{m \cdot m'}) \pmod{X^r - 1, p}$$

□

Lema 2.2.4 *Ak m je introspektívne pre $f(X)$ a $g(X)$, potom je introspektívne aj pre $f(X)g(X)$.*

Dôkaz.

Máme:

$$\begin{aligned} [f(X)g(X)]^m &= [f(X)]^m [g(X)]^m \\ &= f(X^m)g(X^m) \pmod{X^r - 1, p} \end{aligned}$$

□

Z uvedených tvrdení vyplýva, že každé číslo v množine

$$I = \{n^i \cdot p^j \mid i, j \geq 0\}$$

je introspektívne pre každý polynóm v množine

$$P = \left\{ \prod_{a=1}^l (X + a)^{e_a} \mid e_a \geq 0 \right\}$$

Na základe týchto množín teraz definujme dve grupy, ktoré budú základom dôkazu.

Prvá grupa je množina všetkých zvyškov čísel z I modulo r . Toto je podgrupa Z_r^* pretože $(n, r) = (p, r) = 1$. Označme ju G a $|G| = t \leq \phi(r) = |Z_r^*|$. G je generovaná n a p modulo r a pretože $o_r(n) > 4 \log^2 n$, $t > 4 \log^2 n$.

Na definovanie druhej grupy použijeme vlastnosti cyklotomických polynómov nad konečnými poľami. Nech $Q_r(X)$ je r -tý cyklotomický polynóm nad F_p . Polynóm $Q_r(X)$ delí $X^r - 1$ a rozkladá ho na ireducibilné polynómy stupňa $o_r(p)$, podľa vlastnosti vo vete 1.3.4. Nech $h(X)$ je takýto deliteľ.

Druhou grupou bude množina všetkých nenulových zvyškov polynómov v P modulo $h(X)$ a p . Označme ju H , je generovaná $X + 1$, $X + 2$, \dots , $X + l$ v poli $F = F_p[X]/(h(X))$ (vyššie sme položili $l = \lfloor 2\sqrt{\phi(r)} \log n \rfloor$) a je to podgrupa multiplikatívnej grupy $F_p[X]/(h(X))$. O veľkosti tejto grupy hovoria nasledujúce tvrdenia.

Lema 2.2.5 $|H| \geq \binom{t+l-2}{t-1}$.

Dôkaz.

Pretože $h(X)$ je deliteľ cyklotomického polynómu $Q_r(X)$, X je r -tý primitívny koreň jednotky v F (veta 1.3.4).

Teraz ukážeme, že každé dva rôzne polynómy z P stupňa menšieho ako t sa zobrazia do rôznych prvkov v H . Nech $f(X)$ a $g(X)$ sú takéto dva polynómy z P . Predpokladajme, že $f(X) = g(X)$ v poli F . Nech $m \in I$. Vieme, že $[f(X)]^m = [g(X)]^m$ v F . Pretože m je introspektívne pre oba polynómy f a g a zároveň $h(X)$ delí $X^r - 1$, dostávame $f(X^m) = g(X^m)$ v F . Z toho vyplýva, že X^m je koreňom polynómu $Q(Y) = f(Y) - g(Y)$ pre každé $m \in G$. Pretože $(m, r) = 1$ (G je podgrupa Z_r^*), každé X^m je r -tým primitívnym koreňom jednotky. Odtiaľ máme $t = |G|$ rôznych koreňov $Q(Y)$ v F . Avšak stupeň $Q(Y)$ je menej ako t kvôli výberu f a g (každý je menší ako t). Tým dostávame spor, pretože polynóm stupňa t má najviac t koreňov, a preto v F $f(X) \neq g(X)$.

Vieme, že $i \neq j$ v F_p pre $1 \leq i \neq j \leq l$ pretože $l = \lfloor 2\sqrt{\phi(r)} \log n \rfloor \leq 2\sqrt{r} \log n < r$ a $p > r$. Takže prvky $X + 1, X + 2, \dots, X + l$ sú rôzne v F . Je možné, že by bolo $X + a = 0$ v F pre nejaké $a \leq l$ (to je vtedy, keď $h(X) = X + a$), a preto takéto $X + a$ nebude zahrnuté do grupy H . Potom v grupe H existuje aspoň $l - 1$ rôznych polynómov stupňa jeden. Z toho dostávame, že v grupe je aspoň $\binom{t+l-2}{t-1}$ rôznych polynómov stupňa menšieho ako t . \square

Lema 2.2.6 Ak n nie je mocnina p potom $|H| < \frac{1}{2}n^{2\sqrt{n}}$.

Dôkaz.

Uvažujme nasledujúcu podmnožinu I :

$$J = \{n^i \cdot p^j \mid 0 \leq i, j \leq \lfloor \sqrt{t} \rfloor\}$$

Ak n nie je mocnina p potom množina J má $\lfloor 1 + \sqrt{t} \rfloor^2 > t$ rôznych čísel. Pretože $|G| = t$, najmenej dve čísla z J musia byť rovnaké modulo r . Nech sú to m_1 a m_2 a nech $m_1 > m_2$. Dostávame

$$X^{m_1} = X^{m_2} \pmod{X^r - 1}$$

Nech $f(X) \in P$. Pretože m_1, m_2 sú introspektívne pre $f(X)$

$$[f(X)]^{m_1} = f(X^{m_1}) \pmod{X^r - 1, p}$$

$$\begin{aligned}
&= f(X^{m_2}) \pmod{X^r - 1, p} \\
&= [f(X)]^{m_2} \pmod{X^r - 1, p}
\end{aligned}$$

Z toho vyplýva, že

$$[f(X)]^{m_1} = [f(X)]^{m_2}$$

v poli F . Preto $f(X) \in H$ je koreňom polynómu $Q'(Y) = Y^{m_1} - Y^{m_2}$ v poli F . Keďže $f(X)$ je ľubovoľný prvok H , dostávame, že $Q'(Y)$ má najmenej $|H|$ rôznych koreňov v F . Stupeň $Q'(Y)$ je $m_1 \leq (np)^{\lfloor \sqrt{t} \rfloor} < \frac{1}{2}n^{2\sqrt{t}}$ (lebo p delí n ale $n \neq p$ z predpokladov). Z toho vyplýva, že $|H| < \frac{1}{2}n^{2\sqrt{t}}$. \square

Teraz môžeme dokázať správnosť algoritmu.

Lema 2.2.7 *Ak algoritmus vráti PRVOČÍSLO, potom n je prvočíslo.*

Dôkaz.

Prepokladajme, že algoritmus vrátil PRVOČÍSLO. Podľa lemy 2.2.5 máme $t = |G|$ a $l = \lfloor 2\sqrt{\phi(r)} \log n \rfloor$:

$$\begin{aligned}
|H| &\geq \binom{t+l-2}{t-1} \\
&\geq \binom{l-1 + \lfloor 2\sqrt{t} \log n \rfloor}{\lfloor 2\sqrt{t} \log n \rfloor} \tag{2.6}
\end{aligned}$$

$$\geq \binom{2\lfloor 2\sqrt{t} \log n \rfloor - 1}{\lfloor 2\sqrt{t} \log n \rfloor} \tag{2.7}$$

$$\begin{aligned}
&\geq 2^{\lfloor 2\sqrt{t} \log n \rfloor} \tag{2.8} \\
&\geq \frac{1}{2}n^{2\sqrt{t}}
\end{aligned}$$

Podľa lemy 1.3.5 platí nerovnosť 2.6 lebo $t > 2\sqrt{t} \log n$, $t > 4 \log^2 n$. Podľa lemy 1.3.6 platí nerovnosť 2.7 lebo $l = \lfloor 2\sqrt{\phi(r)} \log n \rfloor \geq \lfloor 2\sqrt{t} \log n \rfloor$. Odvodenie 2.8 dostaneme podľa lemy 1.3.4, podmienka lemy je splnená, pretože $2\sqrt{t} \log n \geq 3$.

Ak by n nebolo mocninou p , potom podľa lemy 2.2.6 $|H| < \frac{1}{2}n^{2\sqrt{t}}$ čo by bol spor s $|H| \geq \frac{1}{2}n^{2\sqrt{t}}$. Preto $n = p^k$ pre $k > 0$. Ak by $k > 1$ potom algoritmus vrátil ZLOŽENÉ už v kroku 1. Ostáva len možnosť $k = 1$ tj. $n = p$. \square

2.3 Zložitosť algoritmu

Pre ďalšom využívame fakt, že operácie sčítania, násobenia a delenia dvoch m bitových čísel vieme vykonať v čase $O^{\sim}(m)$ použitím FFT. Podobne, rovnaké operácie na dvoch polynómoch stupňa d s najviac m bitovými koeficientami vieme vykonať v čase $O^{\sim}(d \cdot m)$. Podľa [7]

Veta 2.3.1 *Asymptotická časová zložitosť algoritmu je $O^{\sim}(\log^{10.5} n)$.*

Dôkaz.

Prvý krok algoritmu potrebuje čas $O^{\sim}(\log^3 n)$.

V kroku 2 potrebujeme nájsť vhodné r také, že $o_r(n) > \log^4 n$. To je možné uskutočniť postupným skúšaním hodnôt r , začínajúc od 2, a testujúc či $n^k \not\equiv 1 \pmod{r}$ pre každé $k \leq 4 \log^2 n$. Pre každé r je potrebné najviac $O^{\sim}(\log^2 n)$ násobení modulo r , čo vyžaduje čas $O^{\sim}(\log^2 n \log r)$. Z lemy 2.2.2 vieme, že takéto r nájdeme na $O(\log^5 n)$ pokusov. Celkový čas kroku 2 je potom $O(\log^5 n) O^{\sim}(\log^2 n \log r) = O^{\sim}(\log^7 n)$.

Tretí krok vyžaduje vypočítanie GCD r čísel. Výpočet jedného GCD sa dá urobiť v čase $O(\log n)$. Celková zložitosť kroku je $O(r \log n) = O(\log^6 n)$.

Krok štyri je jedno porovnanie v čase $O(\log n)$.

V poslednom, najdôležitejšom kroku potrebujeme overiť $\lfloor 2\sqrt{\phi(r)} \log n \rfloor$ rovníc. Každá rovnica vyžaduje $O(\log n)$ násobení polynómov stupňa r s koeficientami veľkosti $O(\log n)$. Každú rovnicu vieme overiť v čase $O^{\sim}(r \log^2 n)$. Celková zložitosť kroku 5 je $O^{\sim}(r \sqrt{r} \log^3 n) = O^{\sim}(r^{\frac{3}{2}} \log^3 n) = O^{\sim}(\log^{10.5} n)$.

Celému algoritmu dominuje čas kroku 5, a preto je to celkový čas algoritmu. \square

Podľa [7] sa dá časová zložitosť zlepšiť za predpokladu rôznych, zatiaľ otvorených teórií. Potom by bolo možné nájsť lepšie ohraničenia na r . Jednou z nich je napríklad hypotéza o hustote Sophie-Germain prvočísel, v takom prípade by sa dalo nájsť $r = O^{\sim}(\log^2 n)$, čím by sa dosiahla celková zložitosť $O^{\sim}(\log^6 n)$.

Hypotéza o hustote Sophie-Germain prvočísel

Počet prvočísel $q \leq m$ takých, že $2q + 1$ je tiež prvočíslo je asymptoticky $\frac{2C_2 m}{\ln^2 m}$, kde C_2 je konštanta prvočíselných dvojčiat (približne sa odhaduje na 0.66), \ln je prirodzený logaritmus. Prvočísla q majúce túto vlastnosť sa tiež volajú Sophie-Germain prvočísla.

Kapitola 3

Implementácia

Algoritmus sme implementovali v jazyku C, štandard GNU, kompilovateľnom kompilátorom dodávaným v distribúciach LINUXu gcc. Na prácu s veľkými číslami a matematickými objektami sme použil matematické knižnice/prostredie PARI GP [9].

Po meraní výkonu prvej implementácie v PARI sme sa rozhodli implementovať AKS aj v inom prostredí, ktoré by eliminovalo nevýhody zistené v PARI. Rozhodli sme sa pre jazyk C++ a pre knižnice NTL [3] s podporou GMP [2]. Použili sme kompilátor g++ rovnako dodávaný v distribúcii LINUXu. Na simuláciu systému LINUX v prostredí WINDOWS XP sme použili emulátor CYGWIN.

3.1 Prostredie PARI

PARI/GP je súbor matematických knižníc implementujúcich široké spektrum matematických funkcií a objektov z mnohých oborov matematiky. Na implementáciu AKS z nich postačuje aritmetika veľkých čísel a práca s polynómami.

PARI okrem samotných knižníc v jazyku C, obsahuje aj tzv. PARI kalkulačku. To je konzolová aplikácia na riešenie matematických výpočtov. Všetky funkcie sú tu prezentované ako príkazy prostredia. Konzola je riešená pre užívateľov tak, aby nemuseli programovať, stačí im zadávať jednoduché príkazy.

3.1.1 Konzolové riešenie

Ručné riešenie algoritmu AKS v konzole je jednoduché, uvedieme len riešenie kroku 5, ktorý je z ďalšieho hľadiska zaujímavý, v príklade predpokladáme, že už máme vypočítané r . Zadefinujeme si hodnoty premenných $n=1009$, $r=401$, $a=5$. Príkazom `Mod(x+Mod(a,n), x^r-1)^n` vykonáme presne ľavú stranu rovnice 2.2. Výstupom je objekt modulo polynómu `Mod(Mod(1,1009)* x^207 + Mod(5,1009), x^401-1)`. Čitateľnejšiu formu dosiahneme príkazom `lift()`, ktorý odstraňuje najvyššie modulo. Po aplikovaní dvoch príkazov `lift()` dostaneme výsledok x^{207+5} čo je hodnota pravej strany.

Tento postup treba zopakovať pre všetky potrebné hodnoty a , PARI konzola umožňuje konštrukt tvaru štandardného príkazu cyklu `for`.

3.1.2 Práca s knižnicami PARI

Na začiatku práce s PARI je potrebné inicializovať pracovné prostredie, v ktorom sa budú vykonávať všetky operácie. Prostredie je charakterizované veľkosťou pamäte a počtom predpočítaných prvočísel, ktoré sú uložené interne v tabuľke a používajú sa pri volaniach funkcií, ktoré potrebujú na svoj výpočet prvočíslo.

Knižnice PARI obsahujú jeden generický typ GEN, ktorý predstavuje ľubovoľný matematický objekt, napríklad číslo (celé, reálne, zlomok, ...), polynóm, maticu, vektor, modulovú triedu apod.

Z nášho pohľadu sú ešte zaujímavé konštanty `gzero`, `gun`, `gdeux`, ktoré predstavujú GEN s hodnotou 0, 1 resp 2.

Všetky operácie (sčítanie, odčítanie, ...) sú realizované samostatnými funkciami, napríklad operáciu sčítania dvoch ľubovoľných objektov realizuje funkcia `GEN gadd(GEN x, GEN y)`, sčítanie sa samozrejme vykoná len v prípade ak je možné objekty sčítať, resp. jeden konvertovať na druhý a potom sčítať (napr. polynóm a celé číslo, celé číslo sa konvertuje na polynóm nultého stupňa a sčítajú sa dva polynómy).

Všetky premenné typu GEN sa alokujú v pamäti pridenej prostrediu, rovnako všetky funkcie PARI počítajú v tejto pamäti. Táto pamäť sa správa ako zásobník a je na programátorovi, aby si spravoval jej využitie. Na toto slúžia funkcie tvary `gerepile*`, ktoré upravujú obsah zásobníka. Pri úprave zásobníka sa určuje dvoma pamäťovými adresami úsek, ktorý sa má preusporiadať, a ďalej je možné určiť, ktoré premenné GEN, špecifikované poľom ukazovateľov, majú byť ponechané v zásobníku po jeho úprave. Na získania

aktuálnej adresy vrchu zásobníka slúži premenná `avma`.

Príklad: chceme v cykle vypočítavať nejakú hodnotu, po skončení cyklu nás zaujíma iba výsledok; medzivýsledky chceme "zahodiť".

Povedzme, že vo výpočte používame funkciu `GEN gadd(GEN x, GEN y)`. Z jej definície vidieť, že výsledok sčítania vráti v novom objekte, ktorý sa vytvorí na vrchu zásobníka. Ak máme premenné `z`, `x`, `y`, v zásobníku sú uložené v poradí `[z,x,y]` (vrch je vpravo), a zavoláme funkciu `z = gadd(x,y)`, situácia sa zmení na `[G,x,y,z]`, kde `G` označuje "garbage" v zásobníku (stará hodnota `z`).

Riešení, ako sa zbaviť balastu je niekoľko. Ak nepotrebujeme z výpočtu uchovať žiadnu hodnotu (stačí nám ju vypočítať a hneď sa podľa nej rozhodnúť), môžeme celý úsek zásobníka zrušiť tak, že si pred výpočtom zapamätáme vrch zásobníka `avma` a po skončení výpočtu vrch zásobníka vrátime na pôvodnú hodnotu. Toto riešenie je najrýchlejšie, avšak nedá sa často použiť.

Ostatné riešenia spočívajú vo volaní niektorej z funkcií `gerepile*`, ktoré zabezpečia uchovanie jednej alebo viacerých premenných. Ich zložitosť závisí od počtu premenných, ktoré musia preusporiadať a od veľkosti úseku zásobníka, v ktorom majú pracovať.

Časté volanie týchto funkcií môže ovplyvniť efektívnosť programu.

3.1.3 Implementácia

Prvý krok algoritmu AKS

Účelom tohto kroku je odstrániť zložené čísla tvaru a^b , ktoré ako jediné môžu splniť rovnicu 2.2. Tento test sme implementovali jedným cyklom. V cykle sa postupne skúša či sa dá n napísať ako mocnina niektorého základu a .

Pseudokód testujúci, či $n = a^b$.

```
1. for  $k = 1$  to  $\log n$  do
2.   if  $[2^{\frac{\log n}{k}}]^k = n$  then
3.     return MOCNINA
4. return NIE MOCNINA
```

Implementácia:

```
ok = 0; i = gdeux;
while(!(gcmp(i, logn) == 1) && (ok != 1)) {
```

```

tmp = gpow(gffloor(
    gadd(half, gpow(gdeux, gdiv(logn, i), 5))), i, 5);
if (gcmp(n, tmp) == 0) { ok = 1; }
avma = ltop;
i = gadd(i, gun);
}

```

Premenná `logn` obsahuje $\log_2 n$, `half` je konštanta $\frac{1}{2}$. Funkcia `int gcmp(x, y)` vracia znamienko (-1, 0, 1) rozdielu x-y. Tretí argument funkcií `gpow`, `gffloor` určuje presnosť počítača, hodnota 5 predstavuje nastavenie pre 32-bit procesory.

Druhý a tretí krok algoritmu AKS

Účelom tohto kroku je nájsť vhodné r , ktoré ovplyvňuje celý nasledujúci beh algoritmu (počet samotných testov v kroku 5).

Samotná požiadavka nájdenia najmenšieho r , spĺňajúceho danú podmienku poskytuje priamočiary spôsob implementácie a to postupne overovať pre $r = 1, 2, \dots$ či spĺňa danú podmienku. Na výpočet rádu $o_r(n)$ sme použili funkciu `order` obsiahnutú v knižnici PARI. To však vyžaduje, aby platilo $(n, r) = 1$. Toto je zároveň aj podmienka, ktorá sa testuje v kroku 3, takže sme uvedený test ponechali v kroku 2 a krok 3 sme vypustili.

V cykle postupne testujeme r , pre každé vypočítame rád n modulo r a ten porovnáme s požadovanou hodnotou.

```

l = gtrunc(gsqr(gmul(gdeux, logn)));
r = gdeux; ltop = avma;
while(1) {
    if (gcmp(ggcd(n, r), gun) == 1) {
        if (gcmp(ggcd(n, r), n) == 0) {
            printf("je to prvocislo\n");
        } else {
            printf("je sudelitelne s ");
            output(r);
        }
    }
    exit(0);
}
tmp = Mod0(n, r, 0);
o = order(tmp);

```

```

    if (gcmp(o, 1) == 1) break;
    avma = ltop;
    r = gadd(r, gun);
}

```

Funkcia `ggcd` počíta GCD dvoch čísel, `Mod0` počíta modulo, jej výsledkom je zvyšková trieda a nie konkrétne hodnota (napríklad výsledok operácie `Mod0(5, 3)` je objekt `Mod(2, 3)` a nie číslo 2). Tretí argument `Mod0` je systémový.

Takto získané r sa ukazujú byť prvočísla, čo je vhodné, pretože v kroku 5 nemusíme počítať $\phi(r)$, ktoré vyžaduje faktorizáciu r , ale vieme hneď, že $\phi(r) = r - 1$.

Štvrtý krok algoritmu AKS

Tento test je potrebný pre malé n . Implementovali sme ho ako jedno porovnanie.

```

if (!(gcmp(n,r) == 1)) {
    printf(" je to prvocislo");
    exit(0);
}

```

Piaty krok algoritmu AKS

Toto je kľúčový krok algoritmu, v ktorom sa overujú rovnosti 2.2. Počet kontrol závisí od vypočítaného r . Keďže v kroku 2 vždy získame r , ktoré je prvočíslo, uľahčuje to výpočet hodnoty $\phi(r)$ (Eulerova funkcia), pretože $\phi(r) = r - 1$ ak r je prvočíslo. Na výpočet hodnoty $\phi(r)$ je potrebné faktorizovať číslo r , na čo zatiaľ nie je efektívny postup.

Jadrom cyklu je výpočet a porovnanie dvoch polynómov. Z implementačného hľadiska sme tento krok rozdelili do viacerých podkrokov, reprezentujúcich postup výpočtu z praktického hľadiska. Toto rozdelenie bolo nutné aj z dôvodu práce s objektami v PARI. Objekt polynóm nie je možné modulovať prirodzeným číslom, možno ho modulovať iba ďalším polynómom. Aby sme dosiahli požadované $(\text{mod } n)$, je nutné modulovať samotné koeficienty ešte predtým, ako sa z nich vytvorí polynóm.

1. vypočítanie hodnôt, ktoré nezávisia od behu cyklu, tj. $X^r - 1$ (objekt, ktorým sa modulujú obe strany rovnice), $X^n \pmod{X^r - 1}$ (modulovaná časť pravej strany)
2. výpočet ľavej strany
3. výpočet pravej strany
4. porovnanie oboch strán

```

i = gun;
tmp = gtrunc(gmul(gdeux, gmul(gsqrt(gsub(r, gun), 5), logn)));
x = flisexpr("x");
tmp_pol3 = gpow(x, r, 5);
tmp_pol2 = gsub(tmp_pol3, gun);
p10 = gpow(Mod0(x, tmp_pol2, 0), n, 5);
ltop=avma;
while(!(gcmp(i, tmp) == 1)) {
    tmp_pol1 = Mod0(i, n, 0);
    p1 = gadd(x, tmp_pol1);
    p2 = Mod0(p1, tmp_pol2, 0);
    p3 = gpow(p2, n, 5);
    p11 = gadd(p10, i);
    if (gegal(p3, p11) != 1) {
        printf(" nie je to prvocislo");
        exit(0);
    }
    avma = ltop;
    i = gadd(i, gun);
}

```

Funkciou `flisexpr("x")` získame GEN objekt typu polynóm. Do premennej `tmp_pol3` si uložíme hodnotu X^r a do `tmp_pol2` uložíme $X^r - 1$, čo bude objekt, ktorým budeme modulovať obe strany rovnice. Premenná `p10` obsahuje konštantnú časť pravej strany $X^n \pmod{X^r - 1}$.

Jednotlivé kroky cyklu sú nasledovné: do `tmp_pol3` uložíme $i \pmod{n}$, do `p1` vytvoríme polynóm $X + a \pmod{n}$, následne tento polynóm zmodulujeme $X^r - 1$ a výsledok uložíme do `p2`. Nakoniec ľavú stranu umocníme

na n a výsledok ľavej strany uložíme do `p3`. Ku konštante pravej strany $X^n \pmod{X^r - 1}$ pripočítame i a výsledok pravej strany uložíme do `p11`.

Funkcia `gegal(x,y)` porovná dve premenné či predstavujú rovnakú hodnotu, aj keď sú rôzneho typu (napríklad "nulový polynóm" = "nula" = "Mod(0, n)" = ...).

Ostatné časti kódu

Kompletný zdrojový kód algoritmu ďalej obsahuje definície a inicializácie premenných, inicializáciu prostredia PARI a riešenie vstupu n . Vstup sa zadáva ako reťazec, preto je možné zadať aj n v tvare $10^{20} + 39$, čo prirodzenejšie ako také číslo vypísať celé.

Uvedieme len, že kvôli vysokému stupňu použitých polynómov, inicializujeme prostredie PARI príkazom `pari_init(40000000, 10000)` s cca 40MB pamäte na zásobník a predpočítanými 10000 prvočíslami. Vstup je riešený štandardne cez parametre príkazového riadku, program očakáva ako vstup hodnotu n .

3.1.4 Výkon

Ako už teória naznačila, najpomalšou časťou je cyklus v kroku 5, konkrétne umocnenie ľavej časti na n . Toto sa ukázalo aj v praxi. Jednou z možných praktických príčin môže byť veľmi všeobecné umocňovanie a modulovanie polynómov implementované funkciami `gpow` a `Mod0`. Celkovo je čas výpočtu porovnateľný spusteniu ekvivalentných príkazov v konzole PARI/GP.

Ďalším nedostatkom, nesúvisiacim so zložitou algoritmu, sa ukázalo byť obmedzenie PARI na stupeň polynómu. Maximálny povolený stupeň je 65535. Táto hodnota sa nedá zmeniť, vyplýva totiž z vnútornej implementácie typu polynómu PARI, kde sa stupeň indexuje premenou fyzicky definovanou ako `WORD`. Toto obmedzenie postačuje pre bežné výpočty, ale pre algoritmus AKS je táto hodnota rýchlo prekročená. Už pre $n = 10^{20} + 39$ je nájdené $r = 17683$ a počas výpočtu dosiahneme polynóm veľkosti stupňa $2r$.

Spracovanie prvočísel

Postupne sme testovali 3, 4, 5, 6, 7 a viac miestne prvočísla. Získané časy sú uvedené v tabuľke. Hodnoty boli namerané pomocou C funkcie `gettimeofday` pred a po skončení daného kroku, uvedené hodnoty sú v milisekundách.

Všetky testy sme vykonali na počítači AMD Athlon XP 2000+ s 512MB RAM, operačný systém Win XP.

n	r	l	st01	st02	st05	st05 p
271	269	264	0	0	665	2
1009	401	399	0	4	6379	15
10007	719	712	0	8	71405	100
100109	1109	1105	9	30	390933	353
1000003	1601	1594	5	23	1361055	853
5500003	2027	2015	11	51	2939885	1459
10000019	2179	2170	11	58		1774

Hodnota "l" je počet opakovaní cyklu v kroku 5. Stĺpce "st01", "st02", "st05" obsahujú hodnoty celkového behu uvedeného kroku, stĺpec "st05 p" uvádza priemernú hodnotu jedného opakovania cyklu v kroku 5. Prázdné pole "st05" pre $10^7 + 19$ znamená, že táto hodnota nebola odmeraná, rovnako hodnota "st05 p" pre bola určená po 100 opakovaniach cyklu.

V nasledujúcej tabuľke uvádzame časy jednotlivých podkrokov kroku 5, PS znamená pravá strana, LS ľavá strana rovnice 2.2.

	konštanty	LS modulo	LS umocnenie	PS
271	0	0	2	0
1009	0	0	15	0
10007	0	0	100	0
100109	0	0	353	0
1000003	0	0	852	0
5500003	0	0	1458	0
10000019	0	0		0

Ako vidieť z tabuľky, najnáročnejším výpočtom je umocnenie ľavej strany. V tomto kroku sa umocňuje modulovaný objekt $(X + a) \pmod{X^r - 1, n}$ na n . Ostatné výpočty sú zanedbateľné, v súhrne predstavujú jednu milisekundu.

Pre viac ako 7 ciferné čísla je beh algoritmu neúnosný, jeden prechod cyklu v kroku 5 trvá rádovo sekundy a tento čas rýchlo rastie. Spolu s tým rastie aj nájdené r a od neho závisiaci počet opakovaní v cykle. Pre $10^7 + 19$ je $r = 2179$, čo dáva pri čase jedného prechodu cyklu 1,5 sekundy celkový čas skoro jednu hodinu.

Pre zaujímavosť uvedieme ešte hodnoty r pre niektoré n spolu s časom

potrebným na ich nájdenie.

n	r	čas	n	r	čas
$10^{300} + 331$	3972697	653s	$10^{200} + 31$	1765661	255s
$10^{100} + 267$	441421	48s	$10^{50} + 151$	110477	8.2s
$10^{40} + 121$	70627	4.5s	$10^{30} + 57$	39733	2.1s
$10^{17} + 3$	12791	0.4s	$10^{14} + 31$	8699	0.2s

Ako vidieť, pre čísla 10^{200} sa r blíži k hodnote dvoch miliónov. Prvočísla takejto veľkosti sa používajú v praxi v RSA. Pri overovaní sa bude pracovať s polynómom stupňa približne štyri milióny (max $2r$). Toto je neúnosné ako časovo, tak aj pamäťovo.

Zložené čísla

Testovali sme, koľko času potrebuje algoritmus, aby zistil, že číslo nie je prvočíslo. Testované čísla boli náhodné čísla a čísla niekoľkých špeciálnych tvarov ako Carmichaelove čísla, pseudoprvočísla, súčin prvočísel, násobok prvočísel.

n	r	l	st01	st02	st05 p	prechod
$1009 \cdot 10007 =$ 1001790763	3581	3578	90	96	3254	1
$100109 \cdot 687223$ $= 68797207307$	5189	5186	0	130	10231	1
$8059 \cdot 11833 \cdot 12547$ $= 1196508858409$	6449	6443	94	182	17050	1
$61979 \cdot$ $371869 \cdot 433847 =$ 9999335483415097	11311	11305	32	346	65215	1

Stĺpce reprezentujú rovnaké ukazovatele ako v predchádzajúcej tabuľke, posledný stĺpec "prechod" hovorí, v ktorom teste v kroku 5 algoritmus označil číslo ako zložené.

Prvé dve testované čísla sú súčinom dvoch prvočísel, druhé dve sú Carmichaelove čísla (získané z databázy [8]).

Pre väčšinu zložených čísel algoritmus už v druhom kroku popri hľadaní r nájde deliteľa čísla n . Uvedené čísla sme sa snažil vybrať tak, aby ich najmenší deliteľ bol väčší ako možné r . Ťažké bolo nájsť napríklad vhodné Carmichaelove čísla, všetky ≤ 100000 sa totiž vylúčia hneď v druhom kroku. Rovnako pri pseudoprvočíslach a násobkoch prvočísel sa ukázalo, že sa rýchlo

nájde ich prvočíselný deliteľ, ktorý je menší ako vhodné r . Pre väčšie faktory sa čísla správajú rovnako ako súčin prvočísel.

Zaujímavým pozorovaním je, že ak sa test dostane až do kroku 5, tak všetky testované čísla vylúči už po prvom teste rovnosti 2.2.

Algoritmus má pomerne rýchlu odozvu na zložené čísla. Buď ich odhalí v prvom kroku (mocniny), alebo ak sú to dosť malé čísla (rádovo 10^8), tak nájde pri hľadaní r nejakého deliteľa. Posledným miestom je samozrejme krok 5, avšak tu sa ukazuje, že pre zložené čísla už nebýva splnená ani prvá rovnosť $(X + 1)^n \pmod{X^r - 1, n}$, čo sa overí stále dosť rýchlo. Hodnoty r a čas na jeden prechod cyklu sú síce vyššie, ale oproti prvočíslam rovnakej veľkosti, nie je potrebné overiť všetky rovnosti. Napríklad $n = 10^7 + 19$ má čas overenia cca. jednu hodinu, avšak zloženému číslu to bude trvať do dvoch sekúnd.

3.1.5 Zlepšenia v PARI

Snažili sme sa urýchliť krok 5, implementovaním vlastného umocňovania modulo $X^r - 1$. Algoritmus z teoretického hľadiska silne využíva práve tvar modulujúceho polynómu $X^r - 1$. Naše úvahy vychádzajú z faktu, že mi potrebujeme počítať veľmi špeciálne modulo, ale všeobecná knižnica typu PARI musí vedieť počítať modulo ľubovoľný tvar. Modulovanie takýmto polynómom predstavuje len posun a sčítanie mocnín, tj. koeficient X^r sa pripočíta k absolútnemu členu X^0 , X^{r+1} k X^1 atď., výsledný polynóm bude stupňa $r - 1$.

Kedže potrebujeme výsledok umocniť na n , nestačí vykonať operáciu modulo raz, treba ju vykonávať priebežne počas umocňovania. Z toho dôvodu bolo potrebné napísať aj vlastnú funkciu na umocňovanie, resp. násobenie polynómov. Na umocnenie sme použili klasické umocnenie podľa binárneho rozvoja n .

Táto implementácia však stále nerieši problém veľkého stupňa polynómu v PARI. (Poznámka: ďalšie verzie knižníc PARI majú mať polynómy implementované iným spôsob, už bez obmedzenia na maximálny stupeň.)

Pseudokód: umocnenie a^n

```
bin = binary(n);
result = 1;
tmp = x + 1;
modulus = x^r - 1;
```

```

for (i = 1; i <= size(bin); i++) do
  if (bin[i] == 1) then
    result = (result * tmp) (mod modulus);
  fi;
  tmp = (tmp * tmp) (mod modulus);
od;

```

Pseudokód: modulo $X^r - 1$

```

for(i = r; i <= deg(pol); i++) do
  pol[i - r] = pol[i - r] + pol[i];
  pol[i] = 0;
od;

```

Výkon

Nepodarilo sa nám pôvodnú implementáciu algoritmu urýchliť, dosiahli sme zhruba dvoj až dva a pol násobok pôvodných hodnôt. Dôvodom je náročná a nepohodlná práca s polynómom v prostredí PARI. Kedže polynóm je implementovaný ako pole koeficientov typu GEN, pričom každý koeficient má rôznu veľkosť, pri práci s ním je nutné vytvoriť si novú kópiu. Nie je vo všeobecnosti možné urobiť priradenia $p[i] = p[j]$, kde $p[i]$ je koeficient pri člene X^i , pretože jednotlivé koeficienty môžu mať vnútorne rôznu dĺžku a tým aj štruktúru (v rámci typu GEN).

V tabuľke sú uvedené porovnania kroku 5 oproti pôvodnej verzii implementácie.

n	verzia 1	verzia 2
271	2	6
1009	15	34
10007	100	182
100109	353	491
1000003	853	1393
5500003	1459	2865
10000019	1774	netestované

3.2 Prostredie NTL

Podobne ako PARI, aj NTL je súbor matematických knižníc. NTL je písané v jazyku C++. Poskytuje dátové štruktúry a základné algoritmy aritmetických operácií pre veľké čísla, vektory, matice, polynómy nad číslami a aj konečnými poľami, a reálne čísla.

Podľa NTL dokumentácie [3], všetky operácie sú implementované pomocou najlepších a najrýchlejších algoritmov známych pre daný problém. Z nášho pohľadu je dôležitá polynomiálna aritmetika, autori o nej prehlasujú, že je jedna z najrýchlejších a pomocou nej bolo dosiahnutých niekoľko rekordov pri faktorizácii polynómov. Pre vyšší výkon odporúčajú autori použiť NTL v spolupráci s knižnicami GMP [2]. S použitím GMP bol rovnaký výpočet zhruba o päťnásobne rýchlejší.

NTL je tvorený niekoľkými nezávislými modulmi, každý z nich predstavuje jednu C++ triedu, ktorá je použitá na reprezentáciu daného objektu, napr. `ZZ` je veľké číslo, `ZZ_p` je veľké číslo modulo p , `ZZ_pX` je polynóm s koeficientami `ZZ_p`.

V porovnaní s PARI, NTL neposkytuje žiadny užívateľský výstup. NTL je určené na priame použitie v zdrojovom kóde a je už úlohou aplikácie ako bude prezentovať výsledky. Z tohto dôvodu je práca s každým objektom priamočiara.

Pri implementácii AKS sme použili dátové štruktúry na reprezentáciu polynómu s koeficientami (mod p), umocnenie polynómu na n a modulo $X^r - 1$ sme použili vlastné.

3.2.1 Implementácia

Prvý krok algoritmu AKS

Prvý krok sme implementovali nasledovne ako cyklus počítajúci odmocniny pokým odmocnina je väčšia ako 2. Oproti predchádzajúcej implementácii používame len jednu operáciu s reálnymi číslami ($\sqrt{}$), v pôvodnej to boli dve ($\log n$ a umocnenie). Na realizáciu sme použili typ `RR`, ktorý reprezentuje reálne číslo v ľubovoľnej zvolenej presnosti.

```
cnt = 2; baseRR = to_RR(2);
while(cnt <= binSize) {
    expRR = to_RR(1)/baseRR;
    tmpRR = pow(floatN, expRR);
```

```

powRes = FloorToZZ(tmpRR);
powRes = power(powRes, cnt);
if (powRes == N) {
    cout << "je to mocnina " << powRes << "^" << cnt << "\n";
    exit(0);
}
cnt++;
baseRR++;
}

```

Cyklus postupne skúša vypočítať rôzne odmocniny, zoberie z nich celú časť a tú napäť umocní. Výsledok porovná s pôvodným n .

Druhý a tretí krok algoritmu AKS

Tento krok sme implementovali rovnakým spôsobom ako v PARI. Postupne sa skúšajú vhodné r a pre každé sa testuje, či $o_r(n) \leq 4 \log^2 n$. Keďže NTL nemá funkciu podobnú funkcii `order` v PARI, ktorá by vypočítala rád, tento výpočet sme implementovali ako vnorený cyklus, kde sa testuje pre $k \leq 4 \log^2 n$ nerovnosť $n^k \not\equiv 1 \pmod{r}$.

```

R = 2; R_zz = 2; ok = false;
while(!ok) {
    gcd_tmp = GCD(R_zz, N);
    if (gcd_tmp > 1) {
        if (gcd_tmp == N)
            cout << N << " je prvocislo\n";
        else
            cout << N << " je delitelne " << R << "\n";
        exit(0);
    }
    expo = 1; ok = true;
    while(ok && expo <= bound) {
        copyN = N % R;
        pow = PowerMod(copyN, expo, R_zz);
        if (pow == 1)
            ok = false;
        expo++;
    }
}

```

```

    }
    R++; R_zz++;
}
R--;
cout << "Vyhovujuce R: " << R << "\n";

```

Funckia `GCD(ZZ, ZZ)` vráti najväčšieho spoločného deliteľa dvoch čísel, tento výpočet bol v systéme PARI potrebný ak sme volali funkciu `order`. Zároveň sa tým overuje aj krok 3, ktorý požaduje overiť $(a, n) \neq 1$ pre $a \leq r$. Z tohto dôvodu sme tento výpočet integrovali do kroku 2.

Štvrtý krok algoritmu AKS

Rovnako ako v implementácii PARI, krok 4 je triviálne porovnanie. Poznamenáme, že vzhľadom na možnosti jazyka C++ preťažovať operátory, nie je potrebné volať špeciálnu funkciu na porovnanie dvoch objektov, toto volanie je skryté v použití operátora \leq .

```

if (N <= R)
    cout << "Zlozene cislo\n";

```

Piaty krok algoritmu AKS

V implementácii najdôležitejšieho kroku sme sa rozhodli napísať vlastné umocnenie a priebežné modulovanie medzivýsledku. Dôvody sme uviedli v časti analyzujúcej vylepšenia v implementácii PARI.

Oproti PARI sa prejavila výhoda priameho prístupu k objektom a práce s nimi. Funkciou `SetCoeff(poly, exp, value)` sa v polynóme `poly` nastaví koeficient pri člene x^{exp} na hodnotu `value`. Tým sa priamo mení hodnota koeficientu a nie je nutné pracovať s novými dočasnými objektami na zásobníku ako v PARI.

```

ZZ_p::init(N);
ZZ_pX tmp, rs;
ZZ_p A;

NmodR = N % R;
A = 3;
FloorToZZ(bound, to_RR(2)*SqrRoot(to_RR(R - 1))*logN);

```



```

for(cnt = 1; cnt <= bound; cnt++) {
    set(rs);
    clear(tmp);
    SetCoeff(tmp, 0, A);
    SetCoeff(tmp, 1, 1);

    for(cntSqr = 0; cntSqr < binSize; cntSqr++) {
        if (bin[cntSqr] == 1) {
            rs *= tmp;
            if (deg(rs) >= R) {
                for(cntCoef = R; cntCoef <= deg(rs); cntCoef++) {
                    SetCoeff(rs, cntCoef - R,
                        coeff(rs, cntCoef - R) + coeff(rs, cntCoef));
                    SetCoeff(rs, cntCoef, 0);
                }
            }
        }
        tmp = sqr(tmp);
        if (deg(tmp) >= R) {
            for(cntCoef = R; cntCoef <= deg(tmp); cntCoef++) {
                SetCoeff(tmp, cntCoef - R,
                    coeff(tmp, cntCoef - R) + coeff(tmp, cntCoef));
                SetCoeff(tmp, cntCoef, 0);
            }
        }
    }

    if ((deg(rs) != NmodR) || (LeadCoeff(rs) != 1) ||
        (ConstTerm(rs) != A)) {
        cout << "Zlozene cislo\n";
        cout << " Krok cyklu: " << cnt << "\n";
        exit(0);
    }

    for(cntCoef = deg(rs) - 1; cntCoef > 1; cntCoef--)
        if (coeff(rs, cntCoef) != 0) {
            cout << "Zlozene cislo\n";
            cout << "Koeficient " << cntCoef << " nie je nula\n";
        }
}

```

```

        exit(0);
    }
}
A++;
}

```

Pred začatím výpočtu je nutné nastaviť Z_n , čím dosiahneme vlastnosť $(\text{mod } n)$ z lemy 2.1.1. Toto sa udeje príkazom `Z_p::init(N)`. Zároveň tým nastavíme aj prostredie, v ktorom budú vytvorené polynómy `rs` a `tmp`.

V samotnom cykle sa postupne overujú rovnosti 2.2 pre všetky potrebné hodnoty a . Na začiatku vždy vyprázdňujeme dočasné polynómy (`clear(tmp);`) a vytvoríme nové s aktuálnou hodnotou a (príkaz `SetCoeff`). Vo vnútornom cykle sa vykoná výpočet pravej strany. Realizovali sme tu umocnenie podľa binárneho rozvoja n , v `res` je dočasný výsledok, ktorý sa poskladá s potrebných mocnín `tmp`. Ak stupeň polynómov prekročí r (podmienka `deg(rs) >= R`, analogicky pre `tmp`), vykoná sa modulovanie polynómom $X^r - 1$. To prebehne len posunutím a sčítaním príslušných koeficientov a vynulovaním koeficientov pri $x^i, i \geq r$. Vynulovaním koeficientov udržíme správny stupeň polynómu.

Po umocnení sa porovná výsledok s očakávaným výsledkom predstavujúcim pravú stranu rovnice 2.2. Pravá strana je polynóm $X^n \pmod{r} - a$. Teda, overíme, či vedúci koeficient a absolútny člen polynómu majú rovnaké hodnoty ako pravá strana a či majú obe strany rovnaký stupeň. Nakoniec v cykle overíme, či všetky ostatné koeficienty sú rovné 0.

Ostatné časti kódu

Podobne ako pri implementácii v PARI, aj tu vo zvyšnom kóde iba inicializujeme prostredia resp. inicializujeme a predpočítame premenné.

Vstup n je riešený načítaním až po spustení programu, užívateľ je vyzvaný, aby zadal číslo. Rovnako tu nie je žiadna kontrola na správnosť vstupu. Vstup sa zadáva v dekadickej forme, prípadne v tzv. *scientific* forme "8.5e12". Nie je však možné zadať vstup vo forme " $10^7 + 19$ ", ale je potrebné číslo vypísať celé tj. "10000019".

3.2.2 Výkon

Implementáciu algoritmu v NTL sme testovali na rovnakých číslach ako implementáciu v PARI. Rovnako sme merali aj časy jednotlivých krokov po

mocou funkcie `gettimeofday` . Uvedené hodnoty sú v milisekundách.

Táto implementácia je viditeľne rýchlejšia ako prvá implementácia v PARI. Ako sme už viackrát uviedli, dôvodom bude lepšia práca s objektami a ich zložkami. Ďalej sa "odbúral" problém s maximálnym stupňom polynómu, NTL umožňuje polynómy stupňa $2^{31} - 1$, toto je dané typom `long`. Takáto veľkosť je zatiaľ postačujúca.

Spracovanie prvočísel

n	r	l	st01	st02	st05	st05 p
271	269	264	1	45	846	3
1009	401	399	1	82	4311	10
10007	719	712	3	196	28038	39
100109	1109	1105	4	436	118810	107
1000003	1601	1564	3	916	318516	199
5500003	2027	2015	7	1547	668646	331
10000019	2179	2170	6	1582	872278	401

Ako vidieť z uvedených čísel, výpočet je rapídne rýchlejší. Od hodnôt 10^6 sa ukazuje 3 až 4 násobné urýchlenie.

Spracovanie zložených čísel

n	r	l	st01	st02	st05 p	prechod
1009 · 10007 = 1001790763	3581	3578	5	4193	908	1
100109 · 687223 = 68797207307	5189	5186	5	9482	1881	1
8059 · 11833 · 12547 = 1196508858409	6449	6443	8	12348	3017	1
61979 · 371869 · 433847 = 9999335483415097	11311	11305	11	39346	10468	1

Znova je výpočet rapídne rýchlejší. Pre väčšie n však začína prevládať súčet krokov 1 a 2 nad samotným overením jednej rovnice v kroku 5.

Kapitola 4

Praktické využitie a budúcnosť

4.1 Porovnanie s RSA - Miller-Rabinov test

Dnes zrejme najznámejší a najviac používaný algoritmus na testovanie prvočíselnosti je pravdepodobnostný Miller-Rabinov test [10]

Miller – Rabin

Vstup: nepárne $n > 2$, $r \in \mathcal{N}$

$[2^t \mid (n-1)] \wedge \neg[2^{t+1} \mid (n-1)]$

```
for  $i = 1$  to  $r$  do
     $a = \text{random} \in (1 \leq a \leq n-1)$ 
1.    if  $(a^2 \equiv 1 \pmod{n})$  and  $(a \not\equiv \pm 1 \pmod{n})$ 
         $x \in \{a^{\frac{n-1}{2}}, a^{\frac{n-1}{4}}, a^{\frac{n-1}{8}}, \dots, a^{\frac{n-1}{2^t}}\}$ 
        then return "n COMPOSITE, 100%"
2.    if  $(a^{n-1} \not\equiv 1 \pmod{n})$ 
        then return "n COMPOSITE, 100%"
3.    return "n PRIME, possibly wrong at most  $2^{-r}$ "
```

Veta 4.1.1 Ak b nie je prvočíslo, potom náhodne vybrané číslo a ($1 \leq a \leq b-1$) spĺňa podmienku v príkaze 1 alebo 2 s pravdepodobnosťou aspoň $\frac{1}{2}$.

Dôsledok 2 Záverečné tvrdenie algoritmu je pravdivé.

Dôkaz

Pravdepodobnosť, že ani jedno z r náhodne vybratých čísel a nespĺňa podmienku v príkaze 1 a ani podmienku v príkaze 2 je najviac $(\frac{1}{2})^r$. \square

Existuje implementácia tohto testu využívajúca princípy modulárnej aritmetiky s časovou zložitou $O(r \cdot m^3)$ kde m je počet bitov čísla n .

Pozrime sa na algoritmus bližšie. Parametrom r určujeme počet náhodných výberov hodnoty a . V kroku 2 vidíme test Fermatovej vety. Keďže pravdepodobnosť omylu závisí od r , zvyšovaním jeho hodnoty máme stále väčšiu istotu, že výsledok je správny. Zoberme si použitie v praxi: majme 300-ciferné číslo, bitová dĺžka je približne 1000. Opakujme test $r = 50$ krát.

$$r \cdot m^3 = 50 \cdot 1000^3 \approx 5 \cdot 10^{10}$$

Teda na overenie potrebujeme 50 miliárd krokov/operácií. Pri dnešných procesoroch s frekvenciami 2-3GHz je výpočet otázkou niekoľkých sekúnd. Zároveň máme istotu správnosti výsledku $2^{-50} \approx 8 \cdot 10^{-16}$. Z praktického hľadiska je to nepodstatná chyba, je pravdepodobnejšie, že človek vyhrá v lotérii.

Prečo je algoritmus taký rýchly? Hlavným dôvodom je jeho nenáročnosť. Všetko čo robí, je umocňovanie na druhú modulo n . Nato existujú dostatočne rýchle postupy. Síce sa jedná o umocňovanie veľa ciferného čísla na veľa ciferné číslo, stále je to však rapidne jednoduchšia a ľahšia operácia ako práca s polynómami, ktoré majú koeficienty takej veľkosti. Pre ilustráciu, $n = 5 \cdot 10^{300} + 123$ má parameter $r = 3991237$, takže počas výpočtu algoritmus AKS dosiahne maximálny stupeň polynómu približne 8 miliónov. To je nepoužiteľná hodnota.

4.2 Možné zlepšenia

Objav deterministického polynomiálneho algoritmu riešiaceho problém prvočísel vyvolal novú vlnu záujmu a prvá publikácia AKS sa z teoretického hľadiska rýchlo dočkala mnohých vylepšení.

Hlavným miestom na zlepšenie je samozrejme krok 5, ktorý priamo závisí od veľkosti nájdeneho r . Podľa [5] sa podarilo obmedziť veľkosť grúp, z veľkosti ktorých vyplývajú ohraničenia na r .

Zároveň sa podarilo znížiť konštanty skryté v O -notácii. Oficiálne sa uvádza zlepšenie faktorom 2 milióny, avšak z formálneho hľadiska O -notácie je to málo významné a v praktickom použití dva milióny krát rýchlejšie stále nemusí byť rýchle.

Z praktického hľadiska je algoritmus veľmi vhodným kandidátom na paralelizovanie. Celý výpočet sa skladá z nezávislých overovaní rovníc 2.2. Ich počet je známy na začiatku cyklu, je možné ich rozvrhnúť rovnomerne medzi veľa počítačov. Ako sme ukázali, overenie jednej rovnice nie je nepoužiteľne dlhé, problémom je ich počet.

Algoritmus nie je ani náročný na sieťovú komunikáciu, účastníckemu počítaču sa musí preniesť vstup, hodnota r a hodnoty a , ktoré má overiť. Obidve sú veľkosti $O(\log n)$. Ak sa použije nejaké implicitné rozdelenie hodnôt a medzi počítačmi, napríklad prvý účastník počíta párne hodnoty, druhý nepárne, nie je nutné ani rozdeľovať hodnoty a , stačí pri inicializácii výpočtu vhodne označiť jednotlivých účastníkov. Komunikácia výsledku sa dá riešiť konštantnou veľkosťou, nutné sú dve hodnoty "úspech", "neúspech".

Kapitola 5

Zhrnutie

Ukázali sme, že algoritmus AKS nie je vhodný na praktické použitie. Je neúmerne pomalý už pre malé hodnoty n , už pri hodnotách okolo 1000 výrazne zaostáva za inými metódami. Je to ukážka toho, ako teoretický polynomiálny čas môže zlyhať v skutočnom živote.

Hlavnou brzdou algoritmu je jeho práca s polynómami. Operácie s polynómami sú oproti operáciám s číslami všeobecne pomalé, aj keď sa použijú efektívne postupy, ako napríklad lepšia implementácia násobenia metódou Karatsuba alebo FFT. Celková pomalosť polynómov závisí od ich veľkosti. Číslo rádu 1000 nie je veľké, dá sa s ním pracovať aj ručne, polynóm stupňa 1000 je už obrovský.

Napriek pomalosti pri overovaní skutočných prvočísel sa pri našich testoch ukázalo, že zložené čísla algoritmus rýchlo odhalí už v prvých iteráciách svojho cyklu.

Ďalším praktickým problémom je miesto (pamäť) potrebná na beh algoritmu. S týmto sa vo výpočtovej zložitosti pri určovaní času veľmi neuvažuje. Samozrejme, existujú aj zložitostné triedy kombinujúce pamäťovú a časovú zložitosť, napríklad v [10].

Algoritmus však stále znamená významný prelom v teórii čísel a v problematike prvočíselnosti. Jeho hlavným prínosom je odpoveď na 2000 rokov starú otázku, či je možné nájsť deterministický efektívny (z hľadiska teórie výpočtovej zložitosti) algoritmus na určenie prvočíselnosti. Tým posúva tento problém do triedy zložitosti **P**. Rovnako má algoritmus veľký edukačný potenciál, keďže sám o sebe nie je náročný a na dôkaz správnosti postačujú jednoduché nástroje algebry.

Literatúra

- [1] Prime Pages. available from <http://www.utm.edu/research/primes/>.
- [2] *GMP - GNU Multiple Precision Arithmetic Library, v4.1.4*, 2004. available from <http://www.swox.com/gmp/>.
- [3] *NTL, version 5.4*, 2004. available from <http://www.shoup.net/ntl/index.html>.
- [4] Tibor KAtrínák a kol. *Algebra a teoretická aritmetika*. Univerzita Komenského Bratislava, 1999.
- [5] Daniel J. Bernstein. Proving primality after agrawal-kayal-saxena, draft. available from <http://modular.fas.harvard.edu/edu/Spring2004/129/references/primes/Ber%20nstein-Proving%20primality%20after%20Agrawal-Kayal-Saxena.pdf>.
- [6] Nitin Saxena Manindra Agrawal, Neeraj Kayal. Primes is in p. available from <http://www.cse.iitk.ac.in/news/primality.pdf>.
- [7] Nitin Saxena Manindra Agrawal, Neeraj Kayal. Primes is in p, revised. available from http://www.cse.iitk.ac.in/news/primality_v3.pdf.
- [8] Richard Pinch. Richard Pinch's list of Carmichael Numbers. from <ftp://ftp.dpmms.cam.ac.uk/pub/Carmichael/>.
- [9] The PARI Group, Bordeaux. *PARI/GP, version 2.1.5*, 2004. available from <http://pari.math.u-bordeaux.fr/>.
- [10] Pavol Ďuriš. Výpočtová zložitosť, poznámky k prednáškam. 2005.

- [11] Eric W. Weisstein. Primitive Root of Unity, from MathWorld – A Wolfram Web Resource. from <http://mathworld.wolfram.com/PrimitiveRootofUnity.html>.