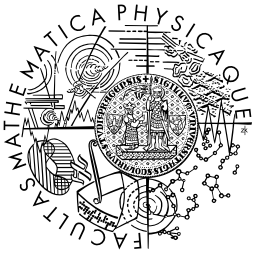

MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Tomáš Ondo

AKS test prvočíselnosti a jeho varianty

Katedra algebry

Vedoucí bakalářské práce: doc. Mgr. et Mgr. Jan Žemlička, Ph.D.

Studijní program: Matematika pro informační technologie

Studijní obor: Matematika pro informační technologie

Praha 2023

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Chcel by som sa poďakovať vedúcemu práce pánovi docentovi Jánovi Žemličkovi za ústretový prístup, dobré poznámky a ochotu pri tejto práci. Taktiež by som sa chcel poďakovať svojej rodine a priateľke za neustálu podporu.

Název práce: AKS test prvočíselnosti a jeho varianty

Autor: Tomáš Ondo

Katedra: Katedra algebry

Vedoucí bakalářské práce: doc. Mgr. et Mgr. Jan Žemlička, Ph.D., Katedra algebry

Abstrakt: V tejto práci je opísaný prvý polynomiálny deterministický test prvočíselnosti s názvom AKS algoritmus. Dôraz je kladený hlavne na jeho časovú náročnosť. Sú opísané jeho nedostatky, ktoré neumožňujú jeho použitie pri generovaní veľkých prvočísel. Sú tu zhrnuté navrhnuté zrýchlenia, ktoré pochádzajú z empirických výsledkov. Tieto zrýchlenia nie sú dokázané, a preto nedávajú deterministický test. Práca pokračuje porovnaním reálneho času výpočtu na konkrétnych implementáciách. Práca ďalej obsahuje variant tohto algoritmu, a to Bernsteinov variant, ktorý má lepšiu asymptotickú časovú náročnosť. Chod týchto algoritmov je ukázaný na príkladoch.

Klíčová slova: AKS algoritmus, test prvočíselnosti, časová náročnosť

Title: AKS primality test and its variants

Author: Tomáš Ondo

Department: Department of Algebra

Supervisor: doc. Mgr. et Mgr. Jan Žemlička, Ph.D., Department of Algebra

Abstract: In this thesis, the first polynomial deterministic primality test named AKS algorithm is described. The thesis is focused on the time complexity of the algorithm. Several drawbacks which make this algorithm unsuitable for generating large prime numbers are described. Improvements derived from empirical data are summed up. These improvements are not proven, so they do not yield a deterministic test. The thesis continues with a comparison of the runtime of concrete implementations. The thesis also contains a variant of the AKS test, the Bernstein variant, which has a better time complexity. The execution of these algorithms is shown in examples.

Keywords: AKS algorithm, primality test, time complexity

Obsah

Úvod

Dôležitou otázkou v teórií prvočísel je ako efektívne zistiť, či dané číslo je prvočíslo. Napriek jednoduchosti zadania sa ukazuje, že tento problém je dosť náročný.
V 80. rokoch 20. storočia sa začali objavovať prvé polynomiálne pravdepodobnostné testy prvočíselnosti. Tie sú založené na tom, že nám vedia určiť prvočíselnosť s určitou pravdepodobnosťou chyby. Tieto testy opakujeme, a tým dostávame menšiu pravdepodobnosť na chybu. V praxi sa používajú práve takéto testy. Hlavným dôvodom je ich rýchlosť.
Ak chceme vedieť s istotou, že dané číslo je prvočíslo, tieto testy nám nepomôžu. V tomto čase sa vyvíjali aj prvé deterministické polynomiálne testy, ale tie sú založené na platnosti nejakého predpokladu, ktorý zatiaľ nie je dokázaný.
V roku 2002 bol predstavený prvý deterministický polynomiálny test prvočíselnosti nazvaný AKS po svojich autoroch Agrawal, Kayal, Saxena, ktorý nevyžaduje žiaden predpoklad. Tomuto testu sa venuje táto práca.
Následne sa objavili ďalšie algoritmy, ktoré naďalej asymptoticky zlepšovali časovú náročnosť. Jeden z nich taktiež predstavím v tejto práci. Zameriam sa hlavne na časové náročnosti týchto algoritmov. Predstavím teoretickú podstatu algoritmov, ale iba odkážem na dôkazy týchto poznatkov. Ukážem, ako použiteľné sú tieto algoritmy v praxi, aké výhody a nevýhody majú, a urobím porovnanie svojich implementácií týchto algoritmov.

Značenie a základné pojmy

$\mathbf{N}$	Prirodzené čísla
$\mathbf{Z}$	Celé čísla
$\mathbf{Z}_n$	Okruh s operáciami modulo n
$\mathbf{Z}_p[x]$	Množina polynómov nad $\mathbf{Z}_p$, kde p je prvočíslo
$\mathbf{F}_p$	Konečné teleso s p prvkami, kde p je prvočíslo
$\varphi(r)$	Eulerova funkcia
$ord_r(n)$	Rád čísla n modulo r
$NSD(a,b)$	Najväčší spoločný deliteľ pre $a,b \in \mathbf{N}$
log	Logaritmus pri základe 2

Definice 1. *Definujeme*

- $f \in \mathcal{O}(g(n)) \Leftrightarrow \exists\, c > 0, n_0 : \forall n > n_0$ *platí* $|f(n)| \leq c\,|g(n)|$
- $f \in \mathcal{O}^{\sim}(g(n)) \Leftrightarrow \exists\, c > 0, n_0, k \geq 0 : \forall n > n_0$ *platí*

$$|f(n)| \leq c\,|g(n)| \log^k(|g(n)|)$$

- $f \in o(g) \Leftrightarrow \lim_{x\to\infty} \frac{f(x)}{g(x)} = 0$

Definice 2. *Nech* $r \in \mathbf{N}, a \in \mathbf{Z}, NSD(a,r) = 1$. *Povieme, že* $k \in \mathbf{N}$ *je rád čísla* a *modulo* r, *ak je najmenšie také, že* $a^k \equiv 1\ (mod\ r)$.

Definice 3. *Pre polynómy* $g_1, g_2 \in \mathbf{Z}[x]$ *definujeme*

$$g_1 \equiv g_2\ (mod\ f,n)$$

ak $g_1 - g_2 \in \mathbf{Z}_n[x]/f$.

Definice 4. *Majme komutatívny okruh* R. *Prvok* $a \in R$ *nazveme jednotka, ak existuje* $b \in R$, *pre ktoré platí* $ab = 1$. *Množinu všetkých jednotiek v* R *značíme* R^*.

Lemma 1 ([1])**.** *Nech* $a,b \in \mathbf{N}$, $|a|,|b| \leq n$. *Potom platí:*

- *násobenie* a,b *má časovú náročnosť* $\mathcal{O}^{\sim}(log(n))$
- *násobenie dvoch polynómov stupňa* d *s koeficientmi, ktoré sú obmedzené v absolútnej hodnote hodnotou* n, *má časovú náročnosť* $\mathcal{O}^{\sim}(d \cdot log(n))$
- a *modulo* b *má časovú náročnosť* $\mathcal{O}^{\sim}(log(n))$
- *Euklidov algoritmus má časovú náročnosť* $\mathcal{O}(log^2(n))$

1. AKS algoritmus

1.1 Predstavenie algoritmu

Základnou myšlienkou algoritmu je nasledujúca lema.

Lemma 2 ([2], Lemma 2.1)**.** *Označme* $a \in \mathbf{Z}$, $n \in \mathbf{N}$, $n \geq 2$, $NSD(a,n) = 1$. *Potom n je prvočíslo práve vtedy, keď*

$$(X + a)^n \equiv X^n + a \ \ (mod\ n)$$

Dá sa ukázať, že stačí testovať kongruenciu v tvare

$$(X + a)^n \equiv X^n + a \ \ (mod\ X^r - 1, n) \tag{1.1}$$

pre vhodné, dostatočne malé r, ktorému sa budeme venovať v ďalšej sekcii. Na rozdiel od kongruencie v leme 2 budeme musieť testovať viacero kongruencií 1.1 pre rôzne čísla a. Modulenie polynómom $X^r - 1$ nám pomôže výrazne znížiť počet členov vo výpočtoch, a tým docieliť požadovanú polynomiálnu časovú náročnosť vzhľadom na dlžku vstupu.

1.2 Opis algoritmu

Definice 5. *Nech* $n \in \mathbf{N}, n > 1$. *Povieme, že n je perfektnou mocninou, ak existujú* $a,b \in \mathbf{N}, b > 1$, *že* $a^b = n$.

Prvým krokom je určiť, či naše n nie je perfektná mocnina. Tento špeciálny prípad je potrebný pre platnosť zvyšných krokov.

Následne potrebujeme nájsť vhodné a dostatočne malé r, ktoré budeme používať v kongruencii 1.1 na testovanie prvočíselnosti. Takéto vhodné r je také, pre ktoré platí $ord_r(n) > log^2(n)$.

Potom otestujeme $NSD(n,a)$ pre čísla a menšie ako r a podmienku $n \leq r$. Ak zistíme, že $1 < NSD(n,a) < n$ pre nejaké a, tak n je zložené číslo. Ak je n menšie ako r, tak n je prvočíslo.

V poslednom kroku budeme testovať kongruenciu 1.1 pre rôzne a od 1 do $\lfloor\sqrt{\varphi(r)}\log(n)\rfloor$. Ak nejaká z kongruencií 1.1 neplatí, tak n je zložené. Naopak, ak to platí pre všetky a, tak n je prvočíslo.

1.3 Kroky algoritmu podrobne

Teraz si opíšeme, ako budeme realizovať jednotlivé kroky algoritmu, a ukážeme ich časovú náročnosť.

1.3.1 Perfektná mocnina

Ukážeme algoritmus, ktorý zistí, či je n tvaru a^b, pre $a,b \in \mathbf{N}, b > 1$.
Pri výpočte budeme používať rýchle modulárne umocňovanie, ktoré si najskôr ukážeme.

Algorithm 1 Rýchle modulárne umocňovanie

Require: $a, n\, m \in \mathbf{N}, n \geq 1$
Ensure: $a^n \ mod \ m$

```
1: k ← n; l ← 1, z ← a mod m
2: while k ≥ 1 do
3:     if k mod 2 = 1 then
4:         l = (l · z) mod m
5:     z = (z · z) mod m
6:     k = k div 2
7: return l
```

Lemma 3. *Algoritmus 1 má časovú náročnosť* $\mathcal{O}^{\sim}(log(n)\, log(m))$.

Důkaz. While cyklus má $log(n)$ iterácií. V každej vykonáme dve násobenia modulo m. Činitele v násobení sú obmedzené veľkosťou m. Podľa lemy 1 má násobenie aj modulenie náročnosť $\mathcal{O}^{\sim}(log(m))$, a preto aj výpočet jednej iterácie má časovú náročnosť $\mathcal{O}^{\sim}(log(m))$. Dostávame finálnu časovú náročnosť $\mathcal{O}^{\sim}(log(n)\, log(m))$.

□

Algorithm 2 Perfektná mocnina

Require: $n \in \mathbf{N}, n \geq 2$
Ensure: "Je" alebo "Nie je" n perfektná mocnina

```
1: b ← 2
2: while 2^b ≤ n do
3:     a ← 1; c ← n
4:     while c − a ≥ 2 do
5:         m ← (a + c) div 2
6:         p ← min{m^b, n + 1}
7:         if p = n then return Je perfektná mocnina
8:         if p < n then
9:             a ← m
10:        else c ← m
11:    b ← b + 1
12: return Nie je perfektná mocnina
```

Lemma 4. *Algoritmus 2 funguje správne a má časovú náročnosť* $\mathcal{O}^{\sim}(log^3(n))$.

Důkaz. Vonkajší while cyklus prebehne $log(n)$-krát. V každom while cykle hľadáme číslo l, pre ktoré platí $l^b = n$ pre špecifické b. Hľadáme ho pomocou bisekcie intervalu, takže ho nájdeme v čase $log(n)$, ak existuje. Pri jeho vyhľadávaní počítame m^b. Na tento výpočet použijeme algoritmus 1 bez modulenia, ale hneď ako medzivýsledok bude väčší ako n, prestaneme s výpočtom, lebo to znamená, že m^b sa nemôže rovnať n. Výpočet m^b má časovú náročnosť $\mathcal{O}^{\sim}(log(b)\, log(n)) = \mathcal{O}^{\sim}(log(log(n))\, log(n)) = \mathcal{O}^{\sim}(log(n))$. Výsledná časová náročnosť je $\mathcal{O}^{\sim}(log^3(n))$.

Poznamenajme ešte raz, že tento algoritmus vyskúša všetký možné exponenty a bisekciou intervalu hľadá správny základ pre konkrétny exponent, ak taký existuje. Ak taký základ nájde, tak vráti, že číslo je perfektná mocnina. Naopak ak ho nenájde pre žiaden exponent, tak vráti, že číslo nie je perfektná mocnina.

□

1.3.2 Hľadanie r

Lemma 5 ([2], Lemma 4.3)**.** *Existuje $r \in \mathbf{N}$, ktoré spĺňa $ord_r(n) > log^2(n)$ a platí $r \leq \max\{3, \lceil log^5(n) \rceil\}$.*

Táto lema je dôležité obmedzenie veľkosti r, ktorá nám umožňuje jeho nájdenie v polynomiálnom čase.

Algorithm 3 Nájdenie vhodného r

Require: $n \in \mathbf{N}, n \geq 2$
Ensure: $r \in \mathbf{N}$;

```
1: r ← 2; t ← 1
2: while r ≤ ⌈log^5(n)⌉ do
3:     for k = 1 to ⌊log^2(n)⌋ do
4:         t = (t · n) mod r
5:         if t = 1 then
6:             r ← r + 1
7:             t ← 1
8:             break
9: return r
```

Lemma 6. *Algoritmus 3 má časovú náročnosť $\mathcal{O}^{\sim}(log^7(n))$.*

Dôkaz. Prechádzame všetky možné r od 2 do $log^5(n)$. Pre každé r potrebujeme $log^2(n)$ násobení modulo r. Jedno násobenie má podľa lemy 1 časovú náročnosť $\mathcal{O}^{\sim}(log(r))$. Preto overiť jedno r má náročnosť $\mathcal{O}^{\sim}(log(r)\, log^2(n))$. Tým dostávame finálnu časovú náročnosť

$$\mathcal{O}^{\sim}(log(r)\, log^2(n)\, log^5(n)) = \mathcal{O}^{\sim}(log(log^5(n))\, log^7(n)) = \mathcal{O}^{\sim}(log^7(n))$$

□

1.3.3 NSD(a,n)

Na nájdenie jednotlivých $NSD(a,n)$ použijeme Euklidov algoritmus.

1.3.4 Počítanie kongruencií

V tomto kroku testujeme $\lfloor \sqrt{\varphi(r)}\, \log(n) \rfloor$ kongruencií.

Lemma 7. *Výpočet jednej kongruencie 1.1 má časovú náročnosť* $\mathcal{O}^{\sim}(r\,log^2(n))$.

Důkaz. Na výpočet $(X + a)^n$ použijeme podobný algoritmus ako algoritmus 1, ale pre polynómy. Podľa lemy 1 má vynásobenie dvoch polynómov stupňa d, ktoré majú koeficienty obmedzené číslom n, časovú zložitosť $\mathcal{O}^{\sim}(d\,log(n))$. V tomto prípade máme polynómy stupňa r a koeficienty obmedzené číslom n. Preto časová náročnosť tohto násobenia bude $\mathcal{O}^{\sim}(r\,log(n))$. Takýchto násobení potrebujeme $log(n)$, rovnako ako v algoritme 1. Preto výsledná časová náročnosť je $\mathcal{O}^{\sim}(r\,log^2(n))$.

□

1.4 Algoritmus v pseudokóde

Teraz, keď už chápeme realizáciu jednotlivých krokov algoritmu, si ho môžeme zapísať do pseudokódu.

Algorithm 4 AKS algoritmus

Require: $n > 1$
Ensure: Prvočíslo alebo Zložené číslo
1: **if** $n = a^b$ pre $a \in \mathbf{N}$ a $b > 1$ **then**
2: **return** Zložené číslo
3: Nájdeme r také, že $ord_r(n) > log^2(n)$
4: **if** $1 < NSD(a,n) < n$ pre nejaké $a \leq r$ **then**
5: **return** Zložené číslo
6: **if** $n \leq r$ **then**
7: **return** Prvočíslo
8: **for** $a = 1$ to $\lfloor\sqrt{\varphi(r)}\log(n)\rfloor$ **do**
9: **if** $(X + a)^n \not\equiv X^n + a \pmod{X^r - 1, n}$ **then**
10: **return** Zložené číslo
11: **return** Prvočíslo

Všetky kroky algoritmu majú polynomiálnu časovú náročnosť vzhľadom na dĺžku vstupu. Najvyššiu náročnosť má testovanie kongruencií, čo vidíme v nasledujúcej vete.

Věta 8. *Algoritmus 4 má časovú náročnosť* $\mathcal{O}^{\sim}(log^{21/2}(n))$.

Důkaz. Z lemy 4 vieme, že určiť, či n je mocnina prvočísla trvá, $\mathcal{O}^{\sim}(log^3(n))$.
Lema 6 nám hovorí, že vhodné r nájdeme za $\mathcal{O}^{\sim}(log^7(n))$.
Z lemy 5 vieme, že r je obmedzené hodnotou $log^5(n)$. Potrebujeme vypočítať toľko $NSD(a,n)$, koľko je r. Jeden výpočet $NSD(a,n)$ trvá podľa lemy 1 $\mathcal{O}(log^2(n))$. Z toho vyplýva, že náročnosť tohto kroku je $\mathcal{O}(log^7(n))$. Výpočet jednej kongruencie podľa lemy 7 trvá $\mathcal{O}^{\sim}(r\,log^2(n))$. Týchto výpočtov prebehne $\lfloor\sqrt{\varphi(r)}\log(n)\rfloor$, a preto výsledná časová náročnosť je $\mathcal{O}^{\sim}(r^{3/2}\,log^3(n)) = \mathcal{O}^{\sim}(log^{21/2}(n))$.
Vidíme, že posledný krok je časovo najnáročnejší, a preto to je aj časová náročnosť celého algoritmu.

□

1.5 Správnosť algoritmu

V tejto časti ukážeme náznak dôkazu správnosti AKS algoritmu. Najskôr si ukážeme, že ak n je prvočíslo, tak algoritmus nám vráti správny výsledok. Algoritmus nám v takom prípade nemôže vrátiť, že n je perfektná mocnina, rovnako ako nenastane situácia, že by sme našli netriviálneho deliteľa pomocou výpočtu $NSD(a,n)$. Ďalej vieme z lemy 2, že kongruencie platia, takže algoritmus vráti správny výsledok.

Na dôkaz opačnej implikácie musíme vybudovať sériu lem.
Predpokladáme, že nám algoritmus vrátil Prvočíslo, a ukážeme, že to je skutočne prvočíslo.

Máme $r \leq \max\{3, \lceil log^5(n) \rceil\}$, že $ord_r(n) > log^2(n)$. Z toho, že $ord_r(n) > 1$, musí existovať prvočíselný deliteľ p čísla n, pre ktorý platí $ord_r(p) > 1$. Ak by pre každý prvočíselný deliteľ platilo $ord_r(p) = 1$, tak by sme dostali, že $ord_r(n) = 1$, čo je spor s veľkosťou $ord_r(n)$. Pre každé n zafixujeme takéto p. Pre p ešte platí $p > r$, lebo inak by algoritmus vrátil Zložené číslo najneskôr v kroku počítania NSD. Toto p využijeme v nasledujúcej definícii.

Definice 6. *Povieme, že $m \in \mathbf{N}$ je introspektívne pre polynóm $f(X)$, ak*

$$(f(X))^m \equiv f(X^m)) \ \ (mod\ X^r - 1, p)$$

Teraz si uvedieme tri lemy, ktoré hovoria o vlastnostiach introspektívnych číslach a ich polynómoch. Ich dôkazy sú nad rámec tejto práce a sú uvedené v [2, Lemma 4.5,4.6].

Lemma 9. *Introspektívne čísla pre polynóm $f(X)$ sú uzavreté na násobenie.*

Lemma 10. *Množina polynómov, pre ktorú je m introspektívne, je uzavretá na násobenie.*

Zavedieme dve množiny, pomocou ktorých definujeme dve dôležité grupy. Označíme $\ell = \lfloor \sqrt{\varphi(r)} \log(n) \rfloor$. Prvá z nich je $I = \{(\frac{n}{p})^i \ p^j \mid i,j \in \mathbf{N_0}\}$ a druhá $P = \{\prod_{a=0}^{\ell}(X+a)^{e_a} \mid e_a \in \mathbf{N_0}\}$.

Lemma 11. *Každý prvok z množiny I je introspektívny pre každý polynóm z P.*

Definice 7. *Definujeme množinu G ako zvyšky po delení číslom r v I. To znamená $\{(\frac{n}{p})^i \ p^j \ mod\ r \mid i,j \in \mathbf{N_0}\}$.*

Veľkosť tejto množiny budeme využívať v ďalšej časti, a preto ju označíme $t = |G|$ a v nasledujúcej leme ju odhadneme.

Lemma 12. *G je multiplikatívna podgrupa $\mathbf{Z_r^*}$ a platí $t > log^2(n)$.*

Důkaz. Pre n a p platí $NSD(n,r) = NSD(p,r) = 1$. G je generovaná n a p, a preto to je podmnožina $\mathbf{Z}_r^*$.
Majme $a,b \in G$, že $a = (\frac{n}{p})^{i_1} \ p^{j_1} \ (mod\ r)$ a $b = (\frac{n}{p})^{i_2} \ p^{j_2} \ (mod\ r)$. Potom

$$ab = (\frac{n}{p})^{i_1} \ p^{j_1} \ (mod\ r) \cdot (\frac{n}{p})^{i_2} \ p^{j_2} \ (mod\ r) = (\frac{n}{p})^{i_1+j_1} \ p^{j_1+j_2} \ (mod\ r)$$

Z toho vyplýva, že $ab \in G$. Pre n platí $ord_r(n) > log^2(n)$ a z toho dostávame $t > log^2(n)$.

□

Věta 13 ([3], Theorem 2.47)**.** *Buď $Q_r(X)$ cyklotomický polynóm nad telesom $\mathbf{F_p}$. Platí:*

- *$Q_r(X)$ delí $X^r - 1$*
- *$Q_r(X)$ sa rozkladá na ireducibilné polynómy stupňa $ord_r(p)$*

Teraz zadefinujeme druhú množinu, ktorá hrá kľúčovú úlohu v dôkaze správnosti. Podľa Vety 13 máme ireducibilný polynóm stupňa $ord_r(p)$ nad $\mathbf{F_p}$. Z voľby p vieme, že $ord_r(p) > 1$, takže máme polynóm stupňa aspoň 2. Nejaký takýto polynóm označíme $h(X)$. Definujeme množinu $\mathcal{G}$ ako zvyšky po delení polynómov z P polynómom $h(X)$ a číslom p.

Lemma 14 ([4], Poznámka 8)**.** *$\mathcal{G}$ je podgrupa multiplikatívnej grupy telesa $\mathbf{F} = \mathbf{F_p}[x]/h(X)$*

Keď už máme definované grupy, pozrieme sa na ich veľkosti.
Dá sa ukázať, že platí spodný odhad na veľkosť grupy $\mathcal{G}$,
a to $|\mathcal{G}| > n^{\sqrt{t}}$[2, Lemma 4.9]. Naopak, dá sa tiež ukázať horný odhad. Ten nám dáva do súvislosti veľkosť tejto grupy s prvočíselnosťou čísla n. Platí, že ak n nie je mocnina p, tak $|\mathcal{G}| \leq n^{\sqrt{t}}$[2, Lemma 4.8].

Z týchto pozorovaní vidíme, že n je mocnina p, t. j. $n = p^k$, ale ak $k > 1$, tak nám algoritmus skončí v prvom kroku. Z toho vyplýva, že $n = p$, takže n je prvočíslo.

1.6 Príklady AKS algoritmu

V tejto podkapitole si ukážeme 2 príklady, jeden na prvočíslo a druhý na zložené číslo. Zložené číslo sa skladá z dvoch prvočísel podobnej dĺžky. Príklady obsahujú aj časové porovnanie testované na implementácii v Sagemath na procesore Intel i5 s frekvenciou 4,2 GHz.

Příklad. Chceme zistiť, či je $n = 10593829 = 6113 \cdot 1733$ prvočíslo.

1. Je $n = 10593829$ tvaru a^b pre a,$b \in \mathbf{N}, b > 1$?
   While cyklus v algoritme končí, ak $2^b > n$ a b sa inkrementuje o 1 každým cyklom. V tomto prípade je to do $b = 23$.
   Algoritmus vráti, že n nie je perfektná mocnina.

2. Nájdenie vhodného r.
   Chceme najmenšie r, že $ord_r(n) > log^2(n)$. Podľa lemy 6 vieme, že takéto r bude menšie ako $log^5(n)$.
   For cyklom začneme hľadanie od 2 do $\lceil log^5(n) \rceil$.
   Pre každé r vyskúšame $n^k \ mod \ r$, pre každé k od 1 do $\lfloor log^2(n) \rfloor$.
   Ak pre žiadne k sa výraz nerovná 1, máme zaručené $ord_r(n) > log^2(n)$.
   V tomto prípade $r = 557$.

3. Skontrolujeme $1 < NSD(a,n) < n$ pre každé $a \leq r$. Pre naše n,r to nenastane.

4. Ak $n \leq r$, tak by to bolo prvočíslo. To nie je náš prípad.

5. $\lfloor\sqrt{\varphi(r)}\log(n)\rfloor = 550$, takže pre $a = 1$ do 550 skúšame

$$(X + a)^n \equiv X^n + a \ \ (mod\ X^r - 1, n)$$

Ak táto kongruencia neplatí pre nejaké a, číslo je zložené. Pre 10593829 to neplatí pre $a = 1$.
Ľavá strana: $(X + 1)^{10593829} \equiv 6843828\, X^{556} + \cdots + 3319309$
Pravá strana: $X^{246} + 1$
Takže toto číslo je zložené.

Výpočet prebehol v priemere za 106 ms. Treba poznamenať, že tento čas je na pomery algoritmu relatívne dobrý, keďže sme testovali iba jednu kongruenciu. Nájdenie vhodného r trvalo 98 ms. Ostatné kroky trvali výrazne menej.

Teraz sa pozrieme na výpočet, ak číslo je prvočíslo.

Příklad. Otestujeme $n = 830111$

1. $n = a^b$? Algoritmus vráti nie.
2. Nájdeme $r = 389$
3. Skúsime $1 < NSD(a,n) < n$ pre každé $a \leq r$. Všetky sa rovnajú 1.
4. Ak $n \leq r$, tak by n bolo prvočíslo. To neplatí.
5. $\lfloor\sqrt{\varphi(r)}\log(n)\rfloor = 387$, počítame

$$(X + a)^n \equiv X^n + a \ \ (mod\ X^r - 1, n)$$

Táto kongruencia platí pre všetkých 387 prípadov. Tu môžeme vidieť jasný výpočtový problém algoritmu, keďže vypočítať jednotlivé kongruencie nie je také rýchle.

$$(X + a)^{830111}\ (mod\ X^{389} - 1, 830111)$$

To v priemere trvalo 794 ms.

Celý algoritmus trval 875 ms a vrátil "Prvočíslo", čo je správny výstup.

2. Možné zlepšenia

V tejto kapitole si ukážeme reálne výpočtové problémy tohto algoritmu a ukážeme návrhy na zrýchlenie.
Návrhy na zlepšenie budeme čerpať z článku [5], v ktorom skúmali, ako sa správajú jednotlivé kroky pre veľký počet čísel. Ako sme videli v príkladoch, máme 2 veľké problémy algoritmu, a to hľadanie vhodného r a počet kongruencií, ktoré musíme testovať. V článku navrhujú tieto zlepšenia:

2.1 Zlepšiť hľadanie r

1. Empiricky sa ukazuje, že je možné nastaviť začiatok hľadania vhodného r na $(\lceil log(n) \rceil - 1)^2$.
   To by pre náš vzorový príklad $n = 10593829$, kde $r = 557$, znamenalo, že by sme začali od 529, čo by ušetrilo zhruba 95 percent výpočtov. Treba poznamenať, že tento odhad platí pre tie čísla, ktoré sa dostanú do časti algoritmu, kde počítame kongruencie. Pre čísla, ktoré sú vyhodnotené pri počítaní $NSD(n,a)$ by toto r malo byť menšie, ale to by nemalo vyvrátiť validnosť tohoto kroku. Zároveň sa ukazuje, že ak r nie je najmenšie možné, tak to tiež nevyvracia správnosť kroku.

2. Ukazuje sa, že väčšina r pre n, ktoré dôjdu do posledného kroku, sú prvočísla. Ak by sme urobili rýchly pravdepodobnostný prvočíselný test kandidáta na r, tak by sme sa vyhli počítaniu $ord_r(n)$ pre zložených kandidátov. Následne, ak nájdeme r prvočíslo, tak pre toto r testujeme jeho rád. Je pravda, že musíme navyše určovať prvočíselnosť r, ale to zaberie menej času ako výpočet rádu.

2.2 Znížiť počet počítaných kongruencií

Zaujímavé pozorovanie je, že ak zložené číslo vstúpi do posledného kroku, tak a, pri ktorom rovnosť neplatí, je extrémne malé. Čísla, ktoré zlyhajú až pre $a = 2$ alebo $a = 3$, majú približne 150 cifier a viac. Rozdiel rýchlosti algoritmu je preto pri prvočíslach a zložených číslach veľmi výrazný.
Podľa empirických výsledkov sa ukazuje, že ak nájdeme najmenšie číslo $\tilde{a}$ z intervalu $[1, \lfloor \sqrt{\varphi(r)} \log(n) \rfloor]$, pre ktoré kongruencia neplatí, tak už kongruencia neplatí pre všetky čísla väčšie ako $\tilde{a}$. Zároveň pre všetky čísla menšie ako $\tilde{a}$ táto kongruencia platí.
Preto stojí za uváženie nahradiť celý for cyklus jedným testom s hodnotou $a = \lfloor \sqrt{\varphi(r)} \log(n) \rfloor$, keďže ak kongruencia neplatí pre nejaké menšie a, tak nebude ani pre toto posledné a.
Treba ale poznamenať, že tento poznatok nie je dokázaný, a preto nám testovanie iba posledného a nedáva deterministický test prvočíselnosti.

Tieto poznatky môžeme integrovať do algoritmu a dostaneme nasledujúci pseudokód.

Algorithm 5 Zrýchlený AKS

Require: $n \in \mathbf{N}$
Ensure: Prvočíslo alebo Zložené číslo
1: **if** $n = a^b$ pre $a \in \mathbf{N}$ a $b > 1$ **then**
2: **return** Zložené číslo
3: Nájdi prvočíslo $r \in \left((\lceil log(n) \rceil - 1)^2, 2\,(\lceil log(n) \rceil)^2 \right)$ a $ord_r(n) \geq \lfloor log^2(n) \rfloor$
4: **if** $1 < NSD(a,n) < n$ pre nejaké $a \leq r$ **then**
5: **return** Zložené číslo
6: **if** $n \leq r$ **then**
7: **return** Prvočíslo
8: **if** $(X + \lfloor \sqrt{\varphi(r)} \log(n) \rfloor)^n \not\equiv X^n + \lfloor \sqrt{\varphi(r)} \log(n) \rfloor \pmod{X^r - 1, n}$ **then**
9: **return** Zložené číslo
10: **return** Prvočíslo

Zmeny môžeme vidieť pri výbere r a počte kongruencií.

3. Porovnanie rýchlosti algoritmov

V tejto časti sa pozrieme na porovnanie troch algoritmov pre čísla rôznej dĺžky. Budeme porovnávať pôvodný AKS, zrýchlený AKS a Trial division algoritmus, ktorý skúša deliteľnosť 2 a všetkými nepárnymi číslami od 3 do $\sqrt{n}$. Ten má časovú náročnosť $\mathcal{O}(\sqrt{n})$

Najskôr sa pozrieme na výsledky z merania, keď sme testovali prvočísla. Značkou — označíme prípad, keď výpočet trval veľmi krátko, prípadne príliš dlho.

Počet cifier	**AKS**	**Zrýchlený AKS**	**Trial division**
4	0,14s	0,006s	—
6	0,82s	0,018s	—
8	4,29s	0,059s	—
10	23,27s	0,226s	—
12	77,72s	0,51s	0,04s
16	—	2,36s	4,88s
18	—	4,38s	—

Tabuľka 3.1: Porovnanie rýchlosti algoritmov pre prvočísla

Teraz si ukážeme porovnanie, kde testujeme zložené čísla.

Počeť cifier	**AKS**	**Zrýchlený AKS**	**Trial division**
10	0,19s	0,024s	0,0002s
15	0,75s	0,14s	0,21s
20	1,46s	0,41s	97,49s
40	16,45s	9,87s	—

Tabuľka 3.2: Porovnanie rýchlosti algoritmov pre zložené čísla

V tabuľke 3.1 vidíme, že zrýchlený variant je oveľa rýchlejší, a to hlavne pre veľký rozdiel v počte testovaných konvergencií.
Zároveň pri číslach $n \leq 10^9$ je Trial algoritmus dokonca rýchlejší (pre menší počet cifier ako 12 to boli zlomky milisekúnd), ale tento algoritmus je pre čísla väčšie ako 10^9 nepoužiteľný.
Čo sa týka tabuľky 3.2, vidíme veľké zlepšenie vo výkone AKS algoritmu. Už medzi 15- až 20-cifernými číslami je rýchlejší ako Trial algoritmus. Zrýchlený algoritmus je stále rýchlejší. Aj keď vieme z predošlej kapitoly, že výpočet kongruencie prebehol iba jeden, rovnako ako v zrýchlenom, tak výpočet r trvá dlhšie. To je vďaka lepšie zvolenému intervalu hľadania vhodného r.
Preto môžeme poznamenať, že AKS nie je až taký zlý test, ak testované n je zložené. 100-ciferné číslo zložene z dvoch veľkých prvočísel vyhodnotil zhruba za 8 minút.

4. Bernsteinov variant AKS

Deterministické testy nám s istotou určia prvočíselnosť čísla. Tento variant nám najskôr nie deterministicky vyberie, za určitých kritérií, kandidáta na certifikát, a následne deterministicky overíme jeho pravosť. Pravosť certifikátu nám už garantuje prvočíselnosť daného čísla. Časová náročnosť tohto algoritmu je $\mathcal{O}(log^{4+o(1)}(n))$.

4.1 Hlavná myšlienka

Hlavnou myšlienkou tohto variantu je nasledujúca veta z článku [6].

Věta 15. *Majme n,d,$e \in \mathbf{N}$, pre ktoré platí:*

- $2^e - 1 \geq n^{2d\lfloor\sqrt{e}\rfloor}$ *a* $e| \; n^d - 1$.
- f *monický polynóm v* $\mathbf{Z}_n[x]$ *stupňa* d.
- r *prvok* $R = \mathbf{Z}_n[x]/f$, *že*

$$r^{n^d-1} \equiv 1 \pmod{f, n}$$

- $r^{(n^d-1)/q} - 1 \in R^*$ *pre každé prvočíslo* q, *že* $q \mid e$
- $r - 1 \in R^*$
- $(y-1)^{n^d} \equiv r^{(n^d-1)/e}\, y - 1 \pmod{y^e - r}$ *v okruhu* $R[y]$

Potom n je mocnina prvočísla.

Vidíme podobnú štruktúru ako pri AKS. Najskôr otestujeme, či n nie je mocnina prvočísla algoritmom 2. Potom využijeme vetu, ktorá nám za určitých podmienok povie, že n je mocnina prvočísla, a tým dostaneme, že n je prvočíslo.

Následne si definujeme, čo je to certifikát pre prvočíslo.

Definice 8. *Buď n,d,$e \in \mathbf{N}$, c,$c' \in \mathbf{Z}$. Nech je f monický polynóm v $\mathbf{Z}_n[x]$ stupňa d. Buď r prvok $R = \mathbf{Z}_n[x]/f$ a S podmnožina R.*
Nech platí:

- $e|n^d - 1$
- $e > c \geq c' \geq 0$
- $r^{n^d-1} \equiv 1 \pmod{f, n}$
- $r^{(n^d-1)/q} - 1 \in R^*$ *pre každé prvočíslo* q, *že* $q|e$
- $s \in R^*$ *pre každé* $s \in S$
- $s^e - (s')^e \in R^*$ *pre rôzne* s,$s' \in S$
- $s^e - r \in R^*$ *pre každé* $s \in S$

- $\binom{e|S|}{c'}\binom{c}{c'}\binom{e|S|-c'+e-1-c}{e-1-c} \geq n^{d\lceil\sqrt{e/3}\rceil}$
- $(y-s)^{n^d} \equiv r^{(n^d-1)/e}\, y - s$ *v okruhu* $R[y]/(y^e - r)$ *pre každé* $s \in S$.

Potom (d,e,c,c',f,r,S) *nazveme certifikát pre* n.

Teraz uvedieme dôležitú vetu, ktorá nám hovorí, ako prepojiť certifikát a prvočíselnosť n.

Věta 16 ([6], Theorem 3.2)**.** *Majme certifikát z Definície 8* (d,e,c,c',f,r,S) *pre* n. *Potom* n *je mocnina prvočísla.*

4.2 Vytvorenie certifikátu

Dá sa dokázať[6, Theorem 5.3], že pre každé n prvočíslo existuje certifikát špecifického tvaru, a to $(d,e,0,0,f,r,\{1\})$. Rovnako ako v sekcii 1.5 vety uvedieme a odkážeme na ich dôkazy, ktorými sa nebudeme zaoberať.
Túto znalosť využijeme tak, že ľahko spočítame kandidátov na d,e,f,r a následne budeme testovať, či pre nich platia podmienky z definície 8.

4.2.1 Nájdenie vhodných kandidátov na d a e

V [6, Theorem 5.1] je dokázané, že pre $n > 1$ existuje d, že $n^d - 1$ má deliteľa $e \geq 6$, pre ktorý platí $e \in (\ d^2\lceil log^2(n)\rceil,\ d^2(d+1)\lceil log^2(n)\rceil\ )$.

Algorithm 6 Vhodné d a e

Require: $n \in \mathbf{N}, n \geq 1$
Ensure: $d,e \in \mathbf{N} : e \mid n^d - 1$

```
1: for d = 1,2,... do
2:     for e = d^2⌈log^2(n)⌉ to d^2(d+1)⌈log^2(n)⌉ do
3:         if n^d − 1 mod e = 0 then
4:             return d,e
```

Lemma 17. *Číslo* d *má hodnotu* $\mathcal{O}(log^{o(1)}(n))$ *a algoritmus 6 má časovú náročnosť* $\mathcal{O}(log^{3+o(1)}(n))$.

Důkaz. Podľa [[6], Theorem 5.2] nájdeme číslo d po $\mathcal{O}(log^{o(1)}(n))$ pokusoch. Pri výpočte d inkrementujeme o 1, a preto má číslo d hodnotu $\mathcal{O}(log^{o(1)}(n))$. Pre každé d máme $d^3\lceil log^2(n)\rceil$ čísel e, ktoré potrebujeme vyskúšať. To je celkovo $\mathcal{O}(log^{2+o(1)}(n))$ operácií mod. Použitím lemy 1 dostávame finálnu náročnosť $\mathcal{O}(log^{3+o(1)}(n))$.

□

Lepšia časová náročnosť sa dá docieliť pomocou výpočtu *zvyškového stromu* [[7], Section 18]. To je algoritmus, ktorý dostane $n,k_1,k_2,\ldots,k_l$ a vráti $n \bmod k_i$ pre $i \in \{1,\ldots,l\}$. Jeho hlavná myšlienka je, že najskôr spočítame $n \bmod k_1 \cdot k_2 \cdots k_l$. To je náš koreň stromu. Následne rozdelíme $\{k_1,\ldots k_l\}$ na $\{k_1,\ldots,\lfloor l/2\rfloor\}$

a $\{\lfloor l/2 \rfloor + 1, \ldots, l\}$. Napríklad $n \ mod \ k_1 \cdots k_{\lfloor l/2 \rfloor}$ spočítame ako $(n \ mod \ k_1 \cdot k_2 \cdots k_l) \ mod \ k_1 \cdots k_{\lfloor l/2 \rfloor}$. Takto postupujeme rekurzívne. Listy tohto stromu budú $n \ mod \ k_1, \ldots, n \ mod \ k_l$. Týmto algoritmom spočítame $n^d - 1$ modulo všetky e v čase $\mathcal{O}(log^{2+o(1)}(n))$.

4.2.2 Nájdenie f

Potrebujeme nájsť vhodný monický polynóm stupňa d v $\mathbf{Z}_n[x]$.

Algorithm 7 Vhodný polynóm f

Require: n,$d \in \mathbf{N}, n \geq 1$
Ensure: monický polynóm stupňa d v $\mathbf{Z}_n[x]$
1: Vygeneruj náhodný monický polynóm f stupňa d
2: **for** $i = 1$ to $d - 1$ **do**
3: $\quad g = NSD(f, x^{n^i} - x)$
4: $\quad$ **if** $\deg(g) > 0$ **then**
5: $\quad\quad$ **Go to 1**
6: **return** f

Lemma 18. *Ak je n prvočíslo, tak algoritmus 7 vráti polynóm f, ktorý je ireducibilný.*

Dôkaz. Ireducibilný polynóm stupňa $k < d$ nad $\mathbf{Z}_n$, kde n je prvočíslo, delí polynóm $x^{n^k} - x$, a preto ak by f nebol ireducibilný, tak by mal netriviálneho NSD s nejakým $x^{n^k} - x$.

□

Lemma 19. *Algoritmus 7 má časovú náročnosť $\mathcal{O}(log^{2+o(1)}(n))$.*

Dôkaz. Podľa lemy 17 má d veľkosť $\mathcal{O}(log^{o(1)}(n))$. Euklidovým algoritmom pre polynómy potrebujeme vypočítať $NSD(f,x^{n^i} - x)$ pre $i \in \{1, \ldots, d-1\}$. Na výpočet prvého kroku $x^{n^i} - x \ mod \ f$ využijeme variant algoritmu 1, kde umocňujeme v $\mathbf{Z}_n[x]/f$. Tento krok má časovú náročnosť $\mathcal{O}(log^{2+o(1)})(n))$ podľa lemy 3. Následne pokračujeme Euklidovým algoritmom a podľa lemy 1 dostávame časovú náročnosť $\mathcal{O}(log^{2+o(1)})(n))$. Počet výpočtov NSD je $\mathcal{O}(log^{o(1)}(n))$, a preto finálna časová náročnosť je $\mathcal{O}(log^{2+o(1)})(n))$.

□

4.2.3 Nájdenie r

Na dokončenie certifikátu potrebujeme nájsť r, pre ktoré platí, že $r^{(n^d-1)/e}$ má rád e v okruhu $\mathbf{Z}_n[x]/f$.

Algorithm 8 Vhodné r

Require: n,d,e,$f \in \mathbf{N}, n \geq 1$
Ensure: $r \in \mathbf{Z}_n[x]/f$: $r^{(n^d-1)/e}$ rádu e
1: Vygeneruj náhodný prvok $r \in \mathbf{Z}_n[x]/f - \{0\}$
2: **if** $r^{n^d-1} \not\equiv 1 \pmod{f, n}$ **then**
3: **Go to 1**
4: **if** $r^{(n^d-1)/q} \equiv 1 \pmod{f, n}$ pre nejaké $q|e$ prvočíslo **then**
5: **Go to 1**
6: **return** r

Lemma 20. *Algoritmus 8 má časovú náročnosť* $\mathcal{O}(log^{2+o(1)}(n))$.

Dôkaz. Použijeme algoritmus 1 pre polynómy, kde stupeň polynómov je obmedzený číslom d a členy obmedzené číslom n. Číslo d má veľkosť $\mathcal{O}(log^{o(1)}(n))$ podľa lemy 17. Podľa lemy 3 a 1 dostávame časovú náročnosť jedného umocňovania $\mathcal{O}(log^{2+o(1)}(n))$. Číslo e je veľkosti $\mathcal{O}(log^{2+o(1)}(n))$, a preto počet bitov je $\mathcal{O}(log^{o(1)}(n))$. Uvážme prvočíselný rozklad $e = \prod_{i=1}^{k} q_i^{s_i}$, potom $2^k \leq e$, takže $k \leq log(e) = log^{o(1)}(n)$. Z toho vyplýva, že je maximálne $log^{o(1)}(n)$ rôznych prvočísel, ktoré delia e. Očakávame, že r volíme náhodne v cyklickej grupe rádu n^d-1, čo je izomorfné $\mathbf{Z}_{n^d-1}$. Vhodné voľby sú čísla, ktoré nie sú násobky prvočísel q, ktoré delia e. Pomer vhodných volieb ku všetkým voľbám je v najhoršom prípade

$$\prod_{i=1}^{k} \frac{q_i}{q_i - 1} = \prod_{i=1}^{k} (1 + \frac{1}{q_i - 1}) \leq \prod_{i=1}^{k} (1 + \frac{1}{i}) = k + 1 \leq 2k$$

Z toho dostávame, že

$$\prod_{i=1}^{k} \frac{q_i}{q_i - 1} \leq 2k = 2(log^{o(1)}(n)) = \mathcal{O}(log^{o(1)}(n)).$$

Výsledná časová náročnosť je $\mathcal{O}(log^{2+o(1)}(n))$.

□

4.3 Overenie certifikátu

Nasledujúca sekcia sa zaoberá prípadom, ak máme certifikát pre dané n a chceme overiť jeho platnosť. Tento postup sa dá zapísať do nasledujúceho algoritmu.

Algorithm 9 Overenie certifikátu

Require: Certifikát (d,e,c,c',f,r,S)
Ensure: Platný alebo neplatný certifikát

1: **if** $n^d - 1 \; mod \; e \; \neq 0$ **then**
2: **return** Neplatný
3: **if not** $e > c \geq c' \geq 0$ **then**
4: **return** Neplatný
5: **if** $r^{n^d-1} \not\equiv 1 \pmod{f, n}$ **then**
6: **return** Neplatný
7: **if** $r^{(n^d-1)/q}$ nie je v R^* pre $q \mid e$ prvočíslo **then**
8: **return** Neplatný
9: **if** s,$s^e - (s')^e$,$s^e - r$ nie sú v R^* pre každé $s \in S$ **then**
10: **return** Neplatný
11: **if** $\binom{e|S|}{c'}\binom{c}{c'}\binom{e|S|-c'+e-1-c}{e-1-c} < n^{d\lceil\sqrt{e/3}\rceil}$ **then**
12: **return** Neplatný
13: **if** $(y-s)^{n^d} \not\equiv r^{(n^d-s)/e}\, y - 1 \pmod{y^e - r}$ v okruhu $R[y]$ pre káždé $s \in S$ **then**
14: **return** Neplatný
15: **return** Platný

Lemma 21. *Algoritmus 9 má časovú náročnosť* $\mathcal{O}(log^{4+o(1)}(n))$

Dôkaz. Najnáročnejší krok výpočtu je výpočet $(y-s)^{n^d} \; mod \; (y^e - r)$ v okruhu $R[y]$. Číslo e má veľkosť $\mathcal{O}(log^{2+o(1)}(n))$. Podľa lemy 1 dostávame, že jedno násobenie v $R[y]/(y^e - r)$ má časovú náročnosť $\mathcal{O}(log^{3+o(1)}(n))$. Potrebujeme vypočítať $(y-s)^{n^d}$. To použitím verzie algoritmu 1 vyžaduje $\mathcal{O}(log^{1+o(1)}(n))$ násobení. To platí, lebo potrebujeme $log(n^d) = d\,log(n)$ násobení, čo je podľa lemy 17 $\mathcal{O}(log^{1+o(1)}(n))$. Preto finálna časová náročnosť je $\mathcal{O}(log^{4+o(1)}(n))$.

□

4.4 Príklad Bernsteinového variantu

Miller-Rabinovým pravdepodobnostným testom sme určili, že číslo $n = 1839755599$ je prvočíslo s pravdepodobnosťou $1 - 1/2^{20}$. Teraz, keď máme dobrého kandidáta, chceme určiť, či to je určite prvočíslo. Použijeme na to Bernsteinov AKS algoritmus.
Prvým krokom je zistiť, či naše n nie je mocninou nejakého prvočísla. Ukáže sa, že nie je.
Následne potrebujeme vytvoriť kandidáta na certifikát pre n.

4.4.1 Vytvorenie certifikátu

- Potrebujeme nájsť d a e, pre ktoré platí $e \mid n^d - 1$. Skúsime to s $d = 1$. Vhodné e hľadáme v intervale [784,1569]. Nájdeme $e = 866$, ktoré splňuje našu podmienku pre $d = 1$.

- V ďalšom kroku hľadáme ireducibilný polynóm stupňa 1 v $\mathbf{Z}_n[x]$. Ten môžeme zvoliť ako náhodný monický polynóm. Vybrali sme $x + 76464621$.
- Následne hľadáme r, pre ktoré platí, že $r^{(n^d-1)/e}$ má rád e v $R = \mathbf{Z}_n[x]/f$. Budeme generovať náhodné prvky z $R - \{0\}$ a testovať rád.
  Rozložíme $e = 866$ na prvočísla, t. j. $866 = 2 \cdot 433$. Po dvoch neúspešných pokusoch na r dostávame $r = 14318755$. Pre toto r platí:
  - $r^{n^d-1} \equiv 1$
  - $r^{(n^d-1)/2} \not\equiv 1$
  - $r^{(n^d-1)/433} \not\equiv 1$

  Jedno z neúspešných r bolo 45160192, pre ktoré dostaneme $r^{n^d-1} \equiv 1$, ale $r^{(n^d-1)/2} \equiv 1$.

Tým dostávame kandidáta na certifikát, a to

$$(1{,}866{,}0{,}0{,}x + 76464621{,}14318755{,}\{1\})$$

Teraz tento certifikát musíme overiť.

4.4.2 Overenie certifikátu

Overujeme certifikát $(1{,}866{,}0{,}0{,}x + 76464621{,}14318755{,}\{1\})$
pre $n = 183642229$. Budeme postupne overovať podmienky z definície 8.

- $e \mid n^d - 1$ platí z výberu d a e.
- $e \leq 0$
- $r^{n^d-1} \equiv 1$
- $r^{(n^d-1)/q} - 1$ pre prvočíselné delitele q čísla e je v R^*.
  Dostaneme 183642227 a 73417853
- $s = 1$ je v R^*.
- c' je v našom certifikáte 0, takže stačí porovnať $\binom{2e-1}{e-1} \geq n^{d\lceil e/3 \rceil}$.
  Ľavá strana má 520 cifier, pravá 141, takže nerovnosť platí.
- Počítame $(y-1)^{n^d}$ a $r^{(n^d-1)/e}$ a testujeme kongruenciu

  $$(y-1)^{n^d} \equiv r^{(n^d-1)/e}\, y - 1$$

  Vychádza nám $(y-1)^{n^d} = 125673755\, y - 1$ a $r^{(n^d-1)/e} = 125673755$.

Týmto výpočtom sme ukázali, že číslo je s určitosťou prvočíslo.
Výpočet prebehol za 409 s, kde výpočty okrem posledného kroku trvali zhruba desatinu sekundy.

Závěr

Cieľom tejto práce bolo opísať časovú náročnosť prvého deterministického polynomiálneho prvočíselného testu AKS a jeho varianty a ukázať ich použiteľnosť. V kapitole 1 som opísal kroky tohto algoritmu, ako aj uviedol časovú náročnosť jednotlivých krokov. Tiež som pomocou svojej implementácie predviedol dva ilustratívne príklady na použitie algoritmu, jeden na prvočíslo a jeden na zložené číslo. Na týchto príkladoch som ilustroval hlavné problémy použiteľnosti v praxi. V kapitole 2 som upozornil na tieto nedostatky a podľa článku [5] som zhrnul možné zlepšenia.
V kapitole 3 som porovnal svoje implementácie AKS algoritmu, zrýchlenej verzie podľa kapitoly 2 a Trial division algoritmu. Tu sa ukázala reálna nepoužiteľnosť algoritmu AKS. Na druhú stranu zlepšenia sa ukazáli dosť efektívne. Ak by sa dokázali empirické výsledky aj teoreticky, dostali by sme relatívne rýchly deterministký test. To by mohlo pomocť priviesť túto variantu do praxe v určitých prípadoch.
V kapitole 4 som predstavil jeden z variantov AKS algoritmu, a to Bernsteinov variant. Tento variant navrhne kandidáta na certifikát a následne testuje jeho platnosť pre dané prvočíslo. Napriek asymptoticky nižšej časovej náročnosti som ukázal, že v realite je tento variant ešte nepoužiteľnejší ako AKS algoritmus. Taktiež som uviedol ilustratívny príklad behu tohto variantu.

Seznam použité literatury

[1] Libor Barto David Stanovský. *Počítačová algebra*. Matfyzpress, 2017.

[2] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, 2004.

[3] R. Lidl. *Introduction to Finite Fields and their Applications*. CAMBRIDGE University Press, 1986.

[4] Alžběta BEDNAŘÍKOVÁ. *Testování prvočíselnosti v polynomiálním čase*. Bakalářská práce. Univerzita Karlova, Matematicko-fyzikální fakulta, Katedra algebry, Praha, 2020.

[5] Lalitha Kiran Nemana and Vadlamudi Ch. Venkaiah. An Empirical Study towards Refining the AKS Primality Testing Algorithm. *IACR Cryptol. ePrint Arch.*, 2016:362, 2016.

[6] Daniel J. Bernstein. Proving Primality in Essentially Quartic Random Time. *Mathematics of Computation*, 76(257):389–403, 2007.

[7] Daniel J. Bernstein. Fast multiplication and its applications. 44, 02 2003. URL: `https://cr.yp.to/lineartime/multapps-20080515.pdf`. Navštívené 8.5.2023.