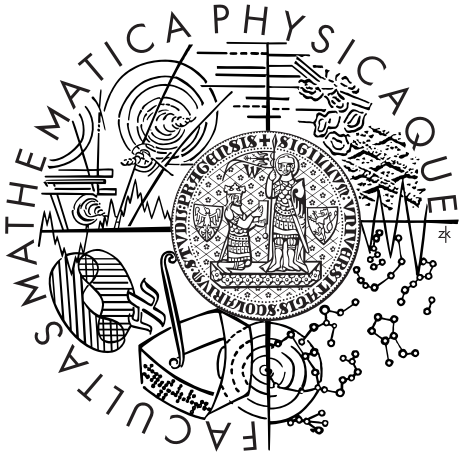
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

**BAKALÁŘSKÁ PRÁCE**




Martin Franců

# Test Rabina-Millera a volba báze

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: prof. RNDr. Petr Simon, DrSc.
Studijní program: Informatika, bakalářské studium
Studijní obor: Obecná informatika

Praha 2011

Děkuji svému vedoucím, panu profesoru Simonovi, za všechny rady a přípomínky.
Díky ale patří i všem ostatním, kteří mě všemožně pomoholi.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

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 v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle 60 odst. 1 autorského zákona.

V Praze dne 3. 8. 2011 Martin Franců

Název práce: Test Rabina-Millera a volba báze

Autor: Martin Franců

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: prof. RNDr. Petr Simon, DrSc., Katedra teoretické informatiky a matematické logiky

Abstrakt: Práce se zabývá různými způsoby volby báze v Rabinově-Millerově testu. V teoretické části je učiněn krátký přehled prvočíselných testů podobných Rabinovu-Millerovu testu a je dokázáno několik tvrzení o struktuře množiny silných lhářů v multiplikativní grupě. Vybrané netradiční volby báze jsou otestovány na množině lichých složených čísel od 100 do 200 000 000 a výsledky jsou porovnány s výsledky při obvyklé volbě báze. Je vyslovena domněnka o vylepšení testu prostřednictvím používání bází určitého tvaru vzhledem k testovanému číslu. Součástí práce je také program, který implementuje posuzované způsoby volby báze. Tento program umožňuje uživateli pohodlné srovnávání výsledků testů s různými způsoby volby báze. V druhé části práce je dokumentace programu.

Klíčová slova: prvočíselný test, Rabinův-Millerův test, volba báze, silný lhář.

Title: Rabin-Miller test and the choice of a basis

Author: Martin Franců

Department: Department of Theoretical Computer Science and Mathematical Logica

Supervisor: prof. RNDr. Petr Simon, DrSc., Department of Theoretical Computer Science and Mathematical Logic

Abstract: This thesis is dedicated to various choices of basis in Rabin-Miller test. Short overview of similar methods is shown and some properties of structure of the set of strong liars are proved in theoretical part. Selected innovative choices of basis are tested on the set of odd composite numbers in range of 100 and 200 000 000 and the results are compared to results of tests with usual choices of bases. Hypothesis about possible improvement of test through using basis of special form with regard to tested number is proposed. Program used for computations of these results is included. The program allows user to compare results of tests with various ways of choosing basis. The second part of the thesis contains documentation of the program.

Keywords: primality test, Rabin-Miller test, choice of basis, strong liar.

# Obsah

# Úvod

Prvočísla přitahovala pozornost matematiků od pradávna. Myšlenka jednoznačné faktorizace je rozebrána již v Eukleidových elementech, Eratosthenovo síto je pojmenováno po řeckém učenci, který žil před naším letopočtem. Později byly vytvářeny rozsáhlé tabulky prvočísel. Zmiňme například tabulku J.P. Kulíka, profesora matematiky na pražské univerzitě, která je výsledkem jeho dvacetiletého koníčku a obsahuje faktorizace všech čísel do 100 000 000. Tato fakta a mnoho dalších je možné najít v [6]. Využití prvočísel však dlouho zůstávalo převážně v akademické rovině.

To se změnilo s objevem šifry RSA. K použití této šifry je potřeba znát dvě poměrně velká prvočísla. Tato čísla musí být tajná, z čehož plyne potřeba uživatele mít možnost tato prvočísla samostatně nacházet. Nejjednodušším řešením tohoto úkolu je generovat náhodné čísla a ta potom testovat na prvočíselnost. Z toho plyne potřeba efektivních prvočíselných testů. Tímto způsobem se prvočísla stala zajímavá pro mnohem širší veřejnost než jen pro matematiky.

V této práci se budeme zabývat prvočíselnými testy. Největší pozornost bude věnována Rabinově-Millerově testu. Průběh tohoto testu je závislý na volbě používaných bází. Nejčastějšími způsoby je buď náhodná volba a nebo volba malých prvočísel. Zvláště druhá možnost je v literatuře často studována. My se budeme zabývat některými alternativními možnostmi volby báze. Vyzkoušíme tyto přístupy na vybraném souboru čísel a výsledky porovnáme s běžnými přístupy. Pro účely získávání těchto dat byl vytvořen program.
V první kapitole je uveden přehled většiny známých prvočíselných testů a je dokázáno několik vlastností Rabinova-Millerova testu. Druhá kapitola se zabývá různými přístupy k volbě báze v tomto testu a výsledky takto upravených testů. V třetí kapitole je popsáno ovládání programu, který byl použit k získání výsledků prezentovaných v druhé kapitole. Poslední kapitola obsahuje programátorskou dokumentaci k vytvořenému programu.

# 1. Prvočíselné testy

## 1.1 Elementární pojmy

Většinu používaných pojmů zavedeme v příslušných kapitolách. Je však mnoho definic a zákonitostí, které s tématem této práce přímo nesouvisí a přesto je budeme v některých důkazech potřebovat. Jejich přesné znění a důkazy by zabraly zbytečně mnoho místa, vzhledem k tomu, že většina čtenářů bude tyto znalosti ovládat. Čtenáře, kterého by tato látka zajímala hlouběji, můžeme odkázat například do knihy [1], případně skripta [2].

V této práci budou všechny číselné proměnné přirozená čísla.

**Definice 1.1** (Dělitelnost)**.** *Řekneme, že $a$ dělí $b$ pokud existuje $k$ takové, že*

$$b = a \cdot k.$$

**Poznámka 1.2.** *Někdy se skutečnost, že $a$ dělí $b$ zapisuje zkráceně jako $a|b$.*

**Definice 1.3** (Prvočíslo)**.** *Číslo $p \in \mathbb{N}$ nazveme prvočíslem, pokud jej nedělí žádné číslo kromě něho samotného a jedničky.*

Kolik je prvočísel? Odpověď dává následující věta.

**Věta 1.4.** *Prvočísel je nekonečně mnoho.*

*Důkaz.* Sporem. Kdyby bylo prvočísel konečně mnoho: $p_1, \dots p_n$. Vezměme jejich součin zvětšený o jedničku $s = p_1 \cdot \dots p_n + 1$. Toto číslo není dělitelné žádným prvočíslem z naší konečné množiny prvočísel. Musí tedy být prvočíslem. Podle předpokladu jej totiž nedělí žádné jiné prvočíslo. To je ale ve sporu s tím, že $s$ není prvkem množiny všech prvočísel $\{p_1, \dots p_n\}$. □

Není bez zajímavosti, kolik rozdílných přístupů k důkazu tohoto faktu existuje. Například v [3] jsou uvedeny hned čtyři.

**Označení 1.5.** *$\mathbb{Z}_n$ budeme značit cyklickou grupu o $n$ prvcích. Znakem $\mathbb{Z}_n^*$ je značena multiplikativní grupa okruhu $\mathbb{Z}/n\mathbb{Z}$.*

## 1.2 Prvočíselné testy

Testování zda dané číslo je prvočíslo přímo z definice je velmi pracné, museli bychom vyzkoušet dělit každým číslem, které je menší než číslo zadané. Snadno si uvědomíme, že stačí testovat pouze prvočísla, protože pokud složené číslo dělí $n$, pak i každý jeho faktor dělí $n$. Tento postup se dá dále zlepšit pomocí jednoduchého lemmatu, které nyní uvedeme.

**Lemma 1.6.** *Nejmenší netriviální dělitel složeného čísla $n$ je vždy menší nebo roven $\sqrt{n}$.*

*Důkaz.* Kdyby byli všichni dělitelé větší než $\sqrt{n}$ a $d$ by byl nejmenší z nich, potom $n = d \cdot e$, přičemž $d \leq e$. Pak dostáváme

$$n = d \cdot e > \sqrt{n} \cdot \sqrt{n} = n,$$

což je spor. □

Díky předešlému lemmatu stačí testovat dělitelnost pouze čísly menšími, nebo rovnými $\sqrt{n}$. Nicméně i takto upravený postup je pořád velmi pracný a v praxi nepoužitelný. Z toho tedy vyplývá potřeba testů prvočíselnosti založených na něčem jiném, než samotné definici prvočísel.

### 1.2.1 Eratosthenovo síto

Eratosthenovo síto slouží k tvorbě seznamu několika prvních prvočísel. Není tedy přímo pročíselným testem, ale dalo by se tak použít. Připomeňme si jeho algoritmus:

**Require:** Limit $L$, do kterého chci spočítat prvočísla.
**Ensure:** Seznam prvočísel, menších než limit.

1. Vytvoř pole $P$ hodnot typu boolean s prvkem za každé číslo menší než $L$.
2. Vlož do každého prvku pole $P$ hodnotu `TRUE`.
3. $x \leftarrow 2$.
4. **while** $x < L$ **do**
5. Vlož do prvků pole $P$ odpovídajících násobkům $x$ (větších než $x$) `FALSE`.
6. $x \leftarrow$ nejmenší číslo, jehož prvek v poli má stále hodnotu `TRUE`, pokud takové číslo neexistuje použij $L$.
7. **end while**
8. **return** Seznam čísel, hodnota jejichž prvku v poli $P$ je stále `TRUE`.

Eratosthenovo síto je poměrně efektivní algoritmus, potřebujeme-li všechna prvočísla. V případě rozhodování o prvočíselnosti je ale ještě méně efektivní než ověřování z definice. Pro generování databáze prvočísel menších než zadaný limit je však tento algoritmus užitečný. Dotaz do této databáze by potom mohl být použit jako prvočíselný test. Poznamenejme ještě, že podle věty 1.4 je prvočísel nekonečně mnoho, tedy ani nemůžeme mít úplný seznam prvočísel.

### 1.2.2 Wilsonova věta

Jiný prvočíselný test je postavený na charakterizaci prvočísel pomocí Wilsonovy věty. Její důkaz se dá najít například v [5].

**Věta 1.7.** *Číslo $n$ je prvočíslem právě tehdy, když platí*

$$(n-1)! \equiv -1 \mod n.$$

K ověření prvočíselnosti $n$ by tedy stačilo spočíst $(n-1)! \mod n$. Bohužel je počítání faktoriálu výpočetně natolik náročná operace, že je tento test v praxi nepoužitelný.

### 1.2.3 Malá Fermatova věta

Další možností, jak testovat prvočíselnost, je najít vlastnost, kterou splňují prvočísla a složená čísla ji splňovat nemusí. Pokud ji testované číslo nesplňuje, určitě nemůže být prvočíslem. Takovou vlastnost nám dá následující věta.

**Věta 1.8** (Malá Fermatova věta)**.** *Pokud $n$ je prvočíslo, potom pro každé $0 < a < n$ platí:*

$$a^{(n-1)} \equiv 1 \mod n.$$

*Důkaz.* Podle Lagrangeovy věty musí řád prvku v grupě dělit řád grupy. Multiplikativní grupa tělesa zbytkových tříd po dělení číslem $n$, tedy $\mathbb{Z}_n$, má ale v případě prvočísla mohutnost $n-1$. Tedy platí:

$$a^{(n-1)} \equiv a^{o \cdot k} \equiv (a^o)^k \equiv 1^k \equiv 1 \mod n,$$

kde $o$ značí řád prvku $a$ v multiplikativní podgrupě a tedy $a^o \equiv 1 \mod n$. □

Test tedy probíhá tak, že zvolíme číslo $a$ z $\{2, \ldots, (n-1)\}$ a spočítáme $a^{(n-1)}$ mod $n$. Pokud výsledek není roven jedné, máme důkaz složenosti testovaného čísla. V opačném případě může být testované číslo pořád i prvočíslo i složené. Mohli bychom ale test opakovat s jinou volbou čísla $a$. Jak si ale ukážeme v následující kapitole, ani to nám nemusí pomoci. Poznamenejme ještě, že číslům $a$, která neodhalí složenost testovaného čísla se říká „Fermatovi lháři“, či „slabí lháři“, nebo někdy jen „lháři“.

### 1.2.4 Carmichaelova čísla

Existují čísla která se chovají jako prvočísla vzhledem k Fermatově testu pro skoro každou volbu $a$.

**Definice 1.9.** *Složené číslo $n$ se nazývá Carmichaelovo, pokud pro každé $a$ nesoudělné s $n$ platí:*

$$a^{(n-1)} \equiv 1 \mod n.$$

Nejmenším takovým číslem je číslo 561. Dříve, než bylo známo první takové číslo, byla dokázána následující charakterizace Carmichaelových čísel.

**Věta 1.10** (Korseltova věta)**.** *Nechť $n \geq 2$ je složené číslo. Následující tvrzení jsou ekvivalentní:*

1. *Pro každé celé číslo $a$ platí:*

$$a^n \equiv a \mod n.$$

2. *Pro každé číslo $a$ nesoudělné s $n$ platí:*

$$a^{(n-1)} \equiv 1 \mod n.$$

3. *Neexistuje prvočíslo $q$, takové, že $q^2$ dělí $n$ a zároveň pro každé prvočíslo platí, že pokud $p$ dělí $n$, pak i $(p-1)$ dělí $(n-1)$.*

*Důkaz.* Důkaz se vynecháme. Zájemce jej může nalézt v [8]. □

Pokud bychom tedy volili jako mocněná čísla $a$ postupně čísla ve tvaru $1 +$ (dělitel $n-1$), po vyzkoušení všech možností by jsme měli jistotu, že testované číslo není Carmichaelovo.

### 1.2.5 Solovayův-Strassenův test

Připomeňme si nyní dvě definice, které nám budou užitečné pro popis Solovayova-Strassenova testu.

**Definice 1.11** (Legendrovy symboly)**.** *Nechť $p$ je prvočíslo, $a \in \mathbb{N}$, pak hodnotu symbolu $\left(\frac{a}{p}\right)$ definujeme takto:*

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{pokud existuje } b \text{ takové, že } a \equiv b^2 \mod p \text{ a } a \not\equiv 0 \mod p \\ -1 & \text{pokud neexistuje } b \text{ takové, že } a \equiv b^2 \mod p \\ 0 & \text{pokud platí } a \equiv 0 \mod p \end{cases}$$

**Definice 1.12** (Involuce)**.** *Mějme grupu s binární operací $\oplus$ a neutrálním prvkem $e$. Prvek $g$ nazveme involucí, pokud pro něj platí, že $g$ není $e$ a $g \oplus g = e$.*

**Poznámka 1.13.** *Poznamenejme, že v cyklické grupě sudého řádu je pouze jediná involuce. V $\mathbb{Z}_p^*$ je to $p-1$. V cyklické grupě lichého řádu neexistuje involuce.*

**Věta 1.14.** *Mějme $p > 2$ prvočíslo, pak platí:*

$$a^{\frac{(p-1)}{2}} \equiv \left(\frac{a}{p}\right) \mod p.$$

*Důkaz.* Označme si $c = a^{\frac{p-1}{2}}$. Pokud $a \equiv 0 \mod p$, pak $c = 0$. V opačném případě víme, že $c^2 \equiv 1 \mod p$ podle Fermatovy věty. Protože $p$ je prvočíslo, $\mathbb{Z}_p^*$ je cyklická grupa a existují v ní pouze dva prvky jejichž druhá mocnina je kongruentní jedné. Jsou to 1 a $-1$.
Pokud by existovalo $b$ takové, že $b^2 \equiv a \mod p$, potom:

$$1 \equiv b^{p-1} \equiv \left(b^2\right)^{\frac{p-1}{2}} \equiv a^{\frac{p-1}{2}} \equiv c \mod p.$$

Nechť naopak platí $a^{\frac{p-1}{2}} \equiv 1 \mod p$. Vezměme si generátor $g$ grupy $\mathbb{Z}_p^*$, potom musí existovat $k$ takové, že $g^k \equiv a \mod p$. Protože je $-1$ jediná involuce v $\mathbb{Z}_p^*$, musí platit, že $g^{\frac{p-1}{2}} \equiv -1 \mod p$. Celkem máme

$$1 \equiv a^{\frac{p-1}{2}} \equiv \left(g^k\right)^{\frac{p-1}{2}} \equiv \left(g^{\frac{p-1}{2}}\right)^k \equiv (-1)^k \mod p.$$

Tedy $k$ je sudé a $a \equiv \left(g^{\frac{k}{2}}\right)^2 \mod p$. □

V [9] je dokázáno, že méně než polovina voleb $a$ nám dá u složeného čísla stejný výsledek, jako u prvočísla. Proto volbou mocněného čísla náhodně zajistíme, že pravděpodobnost jenom lživých výsledků klesá minimálně na polovinu s každou další iterací.

### 1.2.6 Lucasův-Lehmerův test

Pokud $n$ je prvočíslo, pak multiplikativní grupa $\mathbb{Z}_n^*$ je cyklická řádu $n-1$. Musí tedy existovat generátor, který bude mít řád $n-1$ a naopak, pokud takový generátor existuje, je již tato grupa cyklická a $n$ je prvočíslo.

Lucasův-Lehmerův test probíhá opakováním iterací podobně jako Solovayův-Strassenův test. V každé iteraci se zvolí báze $a$, ta se umocní na $n-1$. Pokud je výsledek ve sporu s Fermatovou větou, dokázali jsme, že $n$ je složené číslo. Pokud je výsledek 1, pak víme, že řád $a$ dělí $n-1$. V dalším kroku mocníme zvolenou bázi $a$ na exponenty $e_p = \frac{n-1}{p}$, kde $p$ prochází přes všechny prvočíselné dělitele čísla $n-1$. Jestliže všechny tyto mocniny byly různé od $1 \mod n$, nalezli jsme prvek řádu $n-1$. Pokud některý z výsledků je roven jedné, pak se bohužel nedá s jistotou rozhodnout, zda testované číslo je složené, či nikoliv. V tom případě totiž řád naší báze dělí příslušné $e_p$ a tedy není generátorem.
Tento postup opakujeme s různými bázemi a pokud žádná z nich neprokáže prvočíselnost, či složenost testovaného čísla, je výsledek „pravděpodobně složené“. Nevýhodou tohoto testu je nutnost znát rozklad čísla $n-1$, který získat může být samo o sobě náročné. Na druhé straně tento test nabízí možnost důkazu prvočíselnosti testovaného čísla.

## 1.3 Algoritmus Rabinova-Millerova testu

Rabinův-Millerův test je podobný Fermatovu testu. Používá se vlastnost, která pro prvočísla platí bez vyjímek, ale složená čísla ji splňovat nemusí. Stejně jako v případě Fermatova testu souvisí tato vlastnost s mocněním modulo testované číslo. Mocněné číslo budeme obvykle nazývat bází. Pro snažší vyjadřování si zavedeme následující značení.

**Označení 1.15.** *Domluvme se, že dále budeme proměnnou $n$ používat pro lichá čísla, která testujeme na prvočíselnost. Také proměnné $d$ a $k$ budou, pokud nebude uvedeno jinak, odpovídat rozkladu $n-1 = 2^k \cdot d$, kde $d$ je liché.*

**Require:** $n$ - testované číslo, $a$ - báze nesoudělná s $n$.
**Ensure:** $n$ je složené, pokud je odhaleno jako složené, jinak $n$ je asi prvočíslo.

```
1  x ← a^d mod n
2  if x = ±1 mod n then
3    return n je asi prvočíslo.
4  else
5    for s = 1, ..., k − 1 do
6      x ← x^2 mod n
7      if x = 1 then
8        return n je složené.
9      end if
10     if x = n − 1 then
11       return n je asi prvočíslo.
12     end if
13   end for
14   return n je složené.
15 end if
```

Jeden průběh tohoto algoritmu nazveme smyčkou Rabinova-Millerova testu.

Tento test nám o prvočíslu vždy potvrdí, že je prvočíslo, což si následně ukážeme. Pokud je číslo odhaleno jako složené, musí již být složeným. Bohužel se však může stát, že nám test o složeném čísle bude naznačovat, že je prvočíslo.

Myšlenku testu můžeme přeformulovat takto: V multiplikativní grupě $\mathbb{Z}_p^*$ je pouze jedna involuce a tou je $p-1$. V průběhu testu tedy hledáme jednak spor s Fermatovou větou, ale zároveň i jinou involuci, než $p-1$. Pokud se nám povede jedno z toho najít, dokázali jsme složenost testovaného čísla.

**Věta 1.16.** *Nechť $n$ je prvočíslo, pak smyčka Rabinova-Millerova testu dosáhne správného výsledku při každé volbě báze.*

*Důkaz.* Začneme tím, že v případě prvočísla je multiplikativní grupa tělesa $\mathbb{Z}_p$ cyklickou grupou s řádem $p-1$ a tedy je izomorfní aditivní grupě $\mathbb{Z}_{p-1}$. Snadno si uvědomíme, že tato grupa má pouze jednu involuci a to $n-1$. Podle Fermatovy věty 1.8 musí být $a^{(n-1)} \equiv 1 \bmod n$, musí tedy být buď

$$a^d \equiv 1 \mod n,$$

nebo pro nějaké $0 < j \leq k$ platit

$$a^{2^j \cdot d} \equiv 1 \mod n.$$

To ale znamená, že $a^{2^{(j-1)} \cdot d} \bmod n$ je involuce, tedy $n-1$. Algoritmus se musí zastavit se správným výsledkem. □

Dřív, než se budeme dále zabývat vlastnostmi tohoto algoritmu, ukažme si názorně průběh testu na konkrétních číslech.

Testujme například číslo $n = 133$. Číslo $n - 1 = 132$ si rozložíme na $132 = 2^2 \cdot 33$. Dejme tomu, že jsme náhodou zvolili jako bázi číslo 12. Dostáváme tedy:

$$12^{33} \equiv 132 \mod 133.$$

Tento výsledek testu nám bohužel složenost čísla 133 nedokázal, zvolme tedy novou bázi. Nechť padne los na číslo 93. Počítejme:

$$93^{33} \equiv 64 \mod 133.$$

Nevyšlo nám 1 ani $n-1$, pokračujeme tedy mocněním výsledku na druhou:

$$\left(93^{33}\right)^2 \equiv (64)^2 \equiv 106 \mod 133.$$

Buď není splněna Fermatova věta, nebo je číslo 106 involucí, v obou případech jsme ukázali, že číslo 133 je složené.

## 1.4 Lháři

**Definice 1.17** (Lhář)**.** *Nechť $n$ je liché složené číslo, a rozepišme $n-1$ na $n-1 = 2^k d$, kde $d$ je liché. Číslo $a$, které je v rozmezí $1, \ldots, n-1$, pak nazýváme lhářem vzhledem k číslu $n$, pokud platí jedna z následujících podmínek: buď existuje $s \in \{0, 1, \ldots, k-1\}$ splňující*

$$a^{2^s d} \equiv -1 \mod n,$$

*nebo platí*

$$a^d \equiv 1 \mod n.$$

**Poznámky 1.18.**

1. Zde definovaný pojem "lháře" odpovídá pojmu "silného lháře" v [2], zatímco „slabým lhářem", nebo také "Fermatovým lhářem", jsou označovány prvky $a \in \{1, \ldots, n-1\}$ takové, že

   $$a^{n-1} \equiv 1 \mod n.$$

   Protože se v této práci zabýváme převážně Rabinovým-Millerovým testem, nebudeme potřebovat pojem slabého lháře. Bude tedy užitečnější jej vynechat úplně. Potom již nemůže dojít k nejasnostem a můžeme používat kratší název „lhář" místo „silný lhář".

2. Používá se také jiné označení skutečnosti, že $a$ je lhářem vzhledem k $n$. Říká se, že $n$ silné pseudoprvočíslo vzhledem k $a$.

3. Pokud bude z kontextu jasné vzhledem ke kterému číslu je lhář brán, pak toto určení budeme vynechávat.

4. Název „lhář" souvisí s Rabinovým-Millerovým testem, sama definice neříká nic jiného než to, že složené číslo $n$ se bude během testu chovat jako prvočíslo. Často se báze, které odhalí složenost čísla $n$ nazývají svědci. Lhář by se potom mohl zadefinovat jako číslo z patřičného rozmezí, které není svědkem.

5. Každý snadno nahlédne, že čísla 1 a $n-1$ jsou vždy lháři vzhledem k $n$. Přesto je zvykem používat pro rozmezí volby báze $\{1, 2, \ldots, n-1\}$.

Na úspěšnost testu má klíčový vliv počet lhářů. Následující věta odhaduje jejich počet, což z ní dělá nejdůležitější větu této kapitoly.

**Věta 1.19.** *Pro každé liché složené číslo $n$ platí, že počet lhářů vzhledem k němu je menší než $\frac{n}{4}$.*

*Důkaz.* Důkaz neuvedeme, protože je spíše technického rázu a je poměrně dlouhý. Přesný důkaz se dá najít jak v [2], tak v [1].

□

Zbytek této kapitoly je věnován několika vlastnostem lhářů, které se pokusíme využít v následující kapitole.

**Lemma 1.20.** *Pokud je $a$ lhář vzhledem k číslu $n$, pak i $n-a$ je lhář vzhledem k číslu $n$.*

*Důkaz.* Ověřme, že pro každé $a$ a každé $k$ platí:

$$a^k = (n-a)^k \mod n.$$

Podle binomické věty snadno dostáváme:

$$(n-a)^k = n^k + kn^{k-1}a + \cdots + kna^{k-1} + a^k.$$

Na pravé straně předchozí rovnosti jsou všechny členy kromě posledního dělitelné číslem $n$, platí tedy:

$$(n-a)^k \equiv a^k \mod n.$$

Z předchozího již snadno plyne tvrzení. □

**Lemma 1.21.** *Pokud $a$ je lhář vzhledem k číslu $n$, pak i $a^2 \mod n$ je lhář vzhledem k $n$.*

*Důkaz.* Uvažujme nejdříve případ, kdy $a^d \equiv \pm 1 \mod n$, potom je

$$(a^2)^d \equiv (a^d)^2 \equiv 1 \mod n,$$

tedy $a^2$ je lhář. Nyní vyšetříme druhý případ: pokud existuje $k > 0$ takové, že $a^{2^k d} \equiv -1 \mod n$, pak dostáváme

$$(a^2)^{2^{k-1}d} \equiv a^{(2\cdot 2^{k-1}d)} \equiv a^{2^k d} \equiv -1 \mod n.$$

Poznamenejme ještě, že nemůže existovat $l$ s vlastností $0 \leq l < k-1$ takové, že $a^{2^l d} \equiv 1 \mod n$, protože pak by mocněním $1 \mod n$ nemohla vzniknout $-1 \mod n$. Číslo $a^2$ je tedy lhář vzhledem k $n$ i v tomto případě. □

**Lemma 1.22.** *Nechť $a$ je lhář vzhledem k $n$ a $b$ je v rozmezí $1, \ldots, n-1$ takové, že $a \cdot b \equiv 1 \mod n$. Potom $b$ je také lhářem.*

*Důkaz.* Ukážeme, že $a^k \equiv 1 \mod n$, právě když $b^k \equiv 1 \mod n$. Dále dokážeme, že $b^k \equiv -1 \mod n$, právě když $a^k \equiv -1 \mod n$. Nechť nejprve $a^k \equiv 1 \mod n$, potom

$$1 \equiv a^k \cdot b^k \equiv b^k,$$

přičemž první kongruence platí z toho, že $a \cdot b \equiv 1 \mod n$ a druhá z výchozího předpokladu. Pokud bude $a^k \equiv -1 \mod n$, pak dostáváme

$$-1 \equiv (-1) \cdot a^k \cdot b^k \equiv (-1) \cdot (-1) \cdot b^k \equiv b^k \mod n,$$

kde první kongruence opět platí z volby $a$ a $b$, druhá z výchozího předpokladu. Opačné implikace z dokazovaných ekvivalencí dostaneme stejnými úvahami, protože role $a$ i $b$ jsou symetrické. Z dokázaných ekvivalencí již tvrzení přímo plyne. □

Předcházející lemma v podstatě říká, že pokud $a$ je lhář a je invertibilní v multiplikativní grupě, pak i jeho inverze je lhářem. Při pohledu na lemma 1.21 by se mohlo zdát, že by o lhářích mohlo platit i to, že součin dvou lhářů je také lhářem. To by nám dávalo dobrou charakteristiku množiny lhářů, byla by to totiž podgrupa multiplikativní grupy (jednička je vždy lhářem). Bohužel však taková věc neplatí. Protipříkladem je třeba číslo 29341, jehož nejmenších šest lhářů je $1, 2, 4, 6, 8, 9, 16$. Mezi lháři ale chybí číslo 12, což vyvrací naši domněnku, protože je součinem dvou lhářů 2 a 6. Součin lhářů však často lhářem je. Dokážeme si tedy další lemma, které nám některé takové případy ukáže. K tomu potřebujeme další definici.

**Definice 1.23** (Řád lháře)**.** *Nechť $a$ je lhář vzhledem k číslu $n$, kde $n-1 = 2^k \cdot m$ a $m$ je liché. Potom:*

- *Pokud*

  $$a^d \equiv \pm 1 \mod n,$$

  *pak říkáme, že $a$ je lhář nultého řádu.*

- *Když pro nějaké $l > 0$ platí, že*

$$a^{2^l \cdot d} \equiv -1 \mod n,$$

*pak nazveme $a$ lhářem $l$-tého řádu.*

**Poznámky 1.24.**

1. Poznamenejme nejprve, že předešlá definice je korektní. To znamená, že pro každého lháře je řád určen a zároveň je určen jednoznačně. Alespoň jedna podmínka bude splněna, protože a je lhářem. Pokud by byla splněna první podmínka, pak již druhá podmínka naplněna nebude, neboť sudá mocnina plus nebo minus jedné je vždy jedna.

2. Řád lháře v podstatě neznamená nic jiného, než kolikrát během testu mocníme na druhou.

3. Pro názornost si ukažme řády některých lhářů. Například číslo 2 je lhářem řádu jedna vzhledem k číslu 29341, protože $29340 = 2^2 \cdot 7335$ a $(2^{7335})^2 \equiv 29340 \mod 29341$. Číslo 4 je ale vzhledem k stejnému číslu lhářem řádu nula, protože $4^{7335} \equiv 29340 \mod 29341$. Do třetice přidejme, že číslo 6 je opět lhářem řádu jedna, tedy $(6^{7335})^2 \equiv 29340 \mod 29341$. Následující lemma ukáže, že to není náhoda.

**Lemma 1.25.** *Pokud $a$, $b$ jsou dva lháři rozdílného řádu vzhledem k stejnému číslu $n$, nebo je jejich řád nula, pak jejich součin $a \cdot b \mod n$ je také lhářem.*

*Důkaz.* Nechť platí, že řád obou lhářů je nula. Potom při zachování značení z této kapitoly platí

$$(a \cdot b)^d \equiv a^d \cdot b^d \equiv \pm 1 \cdot \pm 1 \equiv \pm 1 \mod n,$$

tedy $a \cdot b \mod n$ je lhářem podle první podmínky. Nechť mají lháři rozdílný řád. Bez újmy na obecnosti můžeme předpokládat, že existují $0 \leq m < l$ takové, že platí následující:

$$a^{2^l \cdot d} \equiv -1 \mod n,$$

a

$$b^d \equiv 1 \mod n,$$

nebo

$$b^{2^m \cdot d} \equiv -1 \mod n.$$

Z toho ale plyne, že

$$b^{2^l \cdot d} \equiv b^{2^{l-m} \cdot 2^m \cdot d} \equiv \left(b^{2^m \cdot d}\right)^{2^{l-m}} \equiv (\pm 1)^{2^{l-m}} \equiv 1 \mod n,$$

neboť poslední exponent je sudý. Celkem tedy dostáváme:

$$(a \cdot b)^{2^l \cdot d} \equiv a^{2^l \cdot d} \cdot b^{2^l \cdot d} \equiv -1 \cdot 1 \equiv -1 \mod n.$$

Proto $a \cdot b \mod n$ je také lhářem podle definice. □

Zdůrazněme, že pokud je řád lhářů stejný, není jisté, zda bude jejich součin lhářem. Na jedné straně uveďme případ lhářů 2 a 4 vzhledem k číslu 29341, kteří mají oba řád jedna a 8 je také lhářem. Na straně druhé je tu příklad lhářů z poznámky 1.24 3), čísla 2 a 6 jsou lháři vzhledem k číslu 29341, ale 12 není. Celkově tedy součin dvou lhářů není lhářem pouze v některých případech součinu lhářů se stejným řádem.
Z předchozích tvrzení také plyne, že každá mocnina lháře $a$ je také lhářem. To lze nahlédnout tak, že exponent rozdělíme na součet mocnin dvojky a součin $a \cdot a \cdots a$ rozdělíme na odpovídající činitele, přičemž každá taková mocnina našeho lháře je lhářem o jedna menšího řádu, než ta s polovičním exponentem, nebo je nulového řádu. Výsledné číslo je tedy součinem lhářů různého řádu a několika lhářů nulového řádu.

**Lemma 1.26.** *Lháři řádu nula tvoří podgrupu grupy $\mathbb{Z}_n^*$.*

*Důkaz.* Díky Lemmatu 1.25 je zřejmé ,že množina lhářů řádu nula je uzavřená na součin. Neutrální prvek vzhledem k násobení obsahuje triviálně, stačí tedy ukázat, že i inverzní prvek má pořád řád nula, ale to máme okamžitě z průběhu důkazu Lemmatu 1.22. □

**Lemma 1.27.** *Nechť $n$ je liché složené číslo. Následující tvrzení jsou ekvivalentní:*

- *Pro každé dva lháře vzhledem k $n$ platí, že i jejich součin je lhářem.*
- *Číslo $n$ je buď mocninou prvočísla, nebo $n$ dělí prvočíslo tvaru $p = 4k + 3$ pro nějaké $k$.*

*Důkaz.* Nejprve dokážeme zpáteční implikaci. Nechť $n$ dělí prvočíslo $p$ daného tvaru. Ukážeme, že všichni lháři budou nultého řádu. Kdybychom měli lháře $a$ nenulového řádu, tak máme $a^{(2^l \cdot d)} \equiv -1 \mod n$. Z toho plyne, že $n$ dělí $a^{(2^l \cdot d)} + 1$, tedy i $p$ dělí $a^{(2^l \cdot d)} + 1$. Když označíme $c = a^{2^{(l-1)} \cdot d} \mod p$, vidíme, že $-1 \equiv c^2 \mod p$. Z toho ale dostáváme spor neboť:

$$1 = \left(\frac{-1}{p}\right) = (-1)^{\frac{(p-1)}{2}} = (-1)^{\frac{(4k+3-1)}{2}} = (-1)^{2k+1} = -1,$$

kde první rovnost jsme dokázali před chvílí, druhá platí podle Věty 1.14 a zbytek je prosté počítání.

Předpokládejme tedy nyní, že $n = p^k$ pro nějaké $p$ tvaru $4m + 1$. Případ, kdy lháři nejsou stejného řádu, nebo mají nulový řád, dostáváme zadarmo z Lemmatu 1.25. Zbývající případ vyřešíme nyní.

Víme, že $\mathbb{Z}_{p^k}^*$ je izomorfní $\mathbb{Z}_{p-1} \times \mathbb{Z}_{p^{k-1}}$. Všimněme si, že v této grupě máme pouze jediný prvek řádu 2 a tím je dvojice $[\frac{p-1}{2}, 0]$.

Nyní zpátky k grupě $\mathbb{Z}_{p^k}^*$. Prvek $p^k - 1$ v grupě $\mathbb{Z}_p$ má multiplikativní řád 2. To tedy znamená, že zmíněný izomorfismus jej zobrazí na dvojici $[\frac{p-1}{2}, 0]$, nemá jinou možnost. Pro dva lháře $a$ a $b$ stejného řádu $l > 0$ platí

$$(a \cdot b)^{2^l \cdot d} \equiv (-1) \cdot (-1) \equiv 1 \mod n.$$

Protože ale pouze pro $c = -1 \mod p^k$ platí, že $c^2 \equiv 1 \mod p^k$ musí nastat jedna ze dvou následujících variant:

$$\text{Buď existuje } 0 \leq s < l, \text{ takové že } (a \cdot b)^{2^s \cdot d} \equiv -1 \mod p^k,$$

$$\text{nebo } (a \cdot b)^d \equiv 1 \mod p^k.$$

V obou případech je $a \cdot b$ lhářem.

Obrácenou implikaci dokážeme obměnou. Mějme tedy dva lháře $a$ a $b$ takové, že jejich součin není lhářem. Víme, že lháři $a$ i $b$ musejí mít stejný řád $l$ a proto

$$(a \cdot b)^{2^l \cdot d} \equiv 1 \mod n.$$

Zaroveň ale musí existovat číslo $0 < r < l$ takové, že

$$(a \cdot b)^{2^l \cdot d} \equiv 1 \mod n.$$

To ale znamená, že existuje prvek $g$ různý od $n - 1$ takový, že $g \cdot g \equiv 1 \mod n$. Pokud by ale $n = p^k$, pak by $\mathbb{Z}^*_{p^k}$ takový prvek neměla. Tedy $n \neq p^k$.

Zbývá případ, kdy prvočíslo $p = 4o + 3$ dělí $n$. Pokud bychom ale měli dva lháře, jejichž součin by nebyl lhář, pak by samozřejmě oba museli být nenulového řádu. To by ale znamenalo, že musí existovat prvek $w$ takový, že $w^2 \equiv -1 \mod n$, což vede ke sporu stejně jako v první části důkazu. □

# 2. Volby bází Rabinova-Millerova testu

## 2.1 Testovací množina

Srovnávané algoritmy byly testovány na množině lichých složených čísel větších než 100 a menších než 200 000 000. Celkově tedy testujeme 88 921 038 čísel, ačkoliv u mnoha testů stačilo otestovat pouze 653 silných pseudoprvočísel vzhledem k 2 v daném rozsahu.

Nejprve se budeme zabývat výsledky obvyklých voleb bází, abychom je později mohli srovnat s výsledky těch netradičních.

## 2.2 Náhodná báze

Abychom mohli náhodilé výsledky aproximovat absolutními čísly, byla testovací množina testována desetkrát a z tohoto náhodného výběru spočítány výsledky. Pokud budeme za náhodnou veličinu považovat maximální počet iterací nutných k odhalení každého složeného čísla v testovací množině, dostaneme následující náhodný výběr: $\{6, 4, 4, 5, 6, 5, 5, 4, 4, 4\}$. Střední hodnotu naší náhodné veličiny můžeme odhadnout výběrovým průměrem. Tento odhad nám dává $\frac{47}{10}$, odhad rozptylu této náhodné veličiny je $\frac{61}{90}$.

Pro detailnější srovnání s ostatními algoritmy spočítáme taky průměr z histogramů, tedy zprůměrujeme množství čísel s danými počty iterací. Tím sice dostaneme neceločíselné hodnoty, ty ale vystihují přibližnou pravděpodobnost tohoto výsledku. V průměru bylo 88 920 339 čísel odhaleno první smyčkou; $671,1$ čísel odhaleno v druhé iteraci; průměrně 24 jich bylo odhaleno až při třetí volbě báze; $3,3$ je průměrný počet čísel potřebujících o odhalení čtyři iterace; $0,4$ a $0,2$ jsou pravděpodobnosti výskytu čísel s pěti a šesti náhodnými volbami bází.

## 2.3 Báze volená jako malá prvočísla

Při tomto postupu se za báze volí vzestupně malá prvočísla. Tato varianta je v literatuře nejstudovanější a mezi implementacemi poměrně oblíbená. Jsou známy množiny čísel vzdorujících danému počtu iterací tohoto testu. Tomuto tématu se věnuje například článek [7].

Je znám heuristický algoritmus pro hledání silných pseudoprvočísel, vzhledem k dané množině bází. Tedy algoritmus pro nalezení čísel, na kterých test s touto volbou bází selhává. Takový lze najít například v [4]. Ve zmíněném článku je uvedeno dokonce složené číslo, vzhledem ke kterému jsou lháři všechna prvočísla menší než 200. Nebo také číslo ze stejného zdroje:

$$1195068768795265792518361315725116351898245581,$$

které je silným pseudoprvočíslem vzhledem k bázím $2, 3, 5, 7, 11, 13, 19, 23, 29$. Na našem testovacím vzorku potřeboval tento algoritmus pouze maximálně 4 volby bází.

## 2.4 Volba báze odvozená od dělitelů $(n \pm 1)$

Motivace tohoto algoritmu se nachází mezi silnými pseudoprvočísly vzhledem k malým prvočíslům.
Podívejme se například na číslo

$$3215031751 = 151 \cdot 751 \cdot 28351.$$

Toto číslo projde testy pří bázích $2, 3, 5$ i $7$. O jedna menší číslo lze rozložit:

$$3215031750 = 2 \cdot 3^4 \cdot 5^3 \cdot 7 \cdot 37 \cdot 613.$$

Báze $163 = 2{\cdot}3^4{+}1$ i $82 = 3^4{+}1$ jsou svědky složenosti. Všimněme si také, že $2{\cdot}5^3{+}1$, $2{\cdot}3^4{\cdot}5{+}1$, $3^4{\cdot}5^3{+}1$, $2{\cdot}3^4{\cdot}5^3{+}1$, $3^4{\cdot}7^1{+}1$ a několik dalších čísel podobného tvaru jsou svědci. Bohužel však neplatí, že by všechny součiny maximálních mocnin zvětšené o jedna byly svědky, jako příklad postačí $2 \cdot 7 + 1$. Čísel s podobnými vlastnostmi je celá řada, například číslo z předchozího odstavce, které je silným pseudoprvočíslem vzhledem k bázím $2, 3, 5, 7, 11, 13, 19, 23, 29$, ale báze $109 = 2^2 \cdot 3^3 + 1$ je svědkem, přičemž o jedna menší číslo je dělitelné $2^2$ a $3^3$.

Zajimavé je také, že číslo 3215031751 je Carmichaelovo číslo, neboť

$$151 = 2 \cdot 3 \cdot 5^2 + 1, \quad 751 = 2 \cdot 3 \cdot 5^3 + 1 \quad \text{a} \quad 28351 = 2 \cdot 3^4 \cdot 5 \cdot 2 \cdot 7 + 1.$$

Pro Carmichaelovy čísla by volba bází mezi děliteli $n-1$ vedla v každém případě k odhalení složenosti už při použití obyčejného testu založeného pouze na Fermatově větě. Našli bychom totiž přímo dělitele. Nabízí se otázka, zda by tato čísla nepomohla usvědčit i silná pseudoprvočísla vůči více bázím.
V praxi by pravdivost této domněnky znamenala možnost vyměnit faktorizaci čísla $n-1$, která je obvykle celkem jednoduchá, za větší pravděpodobnost odhalení složenosti.
Budeme uvažovat tyto možnosti volby báze odvozené od dělitelů.

### 2.4.1 Dělitelé $(n-1)$ zvětšení o jedna

Když byly báze voleny jako čísla tvaru $1 + (\text{dělitel } n-1)$, byla všechna složená čísla v testovací množině odhalena méně, než pěti testy. Další možností je vynechat z této množiny sudé báze, volit tedy pouze sudé dělitele čísla $n-1$. Takto pozměněný test má velmi podobné vlastnosti.

### 2.4.2 Částečná faktorizace $(n-1)$, pouze maximální mocniny

V tomto algoritmu používáme jako základ bází pouze dělitele s maximálními mocninami prvočísel, které ještě dělí $n-1$. I zde jsme se omezili na sudé dělitele, aby výsledná báze byla lichá. Maximální počet testů je opět pět. Nicméně zatím nebylo odhaleno číslo, které by mezi takto specifikovanými bázemi nemělo svědka.

### 2.4.3 Částečná faktorizace $(n+1)$ zvětšená o jedna

Když testujeme báze odvozené od dělitelů $n-1$, nabízí se zvážit také báze odvozené od dělitů $n+1$. Tento algoritmus se však ukázal neužitečný: byl nalezen příklad čísla, které je silným pseudoprvočíslem vzhledem ke všem o jedna zvětšeným dělitelům čísla $n+1$. Tímto protipříkladem je číslo 13694761. Jeho faktorizace a faktorizace zvětšeného čísla jsou:

$$13694761 = 2617 \cdot 5233, \quad 13694761 + 1 = 2 \cdot 68473881.$$

Přitom ale $1+1$, $2+1$ i $6847381+1$ jsou lháři. V testovací množině je čísel, na kterých tento algoritmus selhává, dokonce osm.

### 2.4.4 Přímo dělitelé $(n \pm 1)$

Pro oba tyto způsoby volby báze byly nalezeny příklady čísel na kterých selžou. Číslo $1530787 = 619 \cdot 2473$ nemá mezi děliteli 1530786 ani jednoho svědka. Jako protipříklad pro volbu dělitelů $n+1$ může sloužit rovnou číslo $2047 = 23 \cdot 89$, které nemá mezi přípustnými bázemi ani jednoho svědka. To, že tyto testy budou dopadat špatně se ostatně dalo předvídat. Pokud je dané prvočíslo lhářem, je potom lhářem i každá mocnina tohoto prvočísla. Což v případě čísla $n = 2047$ dává všechny dělitele $n+1$.

### 2.4.5 Volba báze jako prvočísla nedělící $n \pm 1$

Špatné výsledky předchozích algoritmů navádějí zkusit použít jeho opak. Tedy volit naopak čísla, která nedělí $n-1$, či $n+1$. Například mezi prvočísly, či náhodnými čísly.

Prvočísla nedělící $n+1$ se ukázala jako poměrně dobré báze. Všechna čísla v testovací množině byla odhalena do méně než čtyř testů. Prvočísla nedělící $n-1$ potřebují na otestování všech čísel do pěti testů.

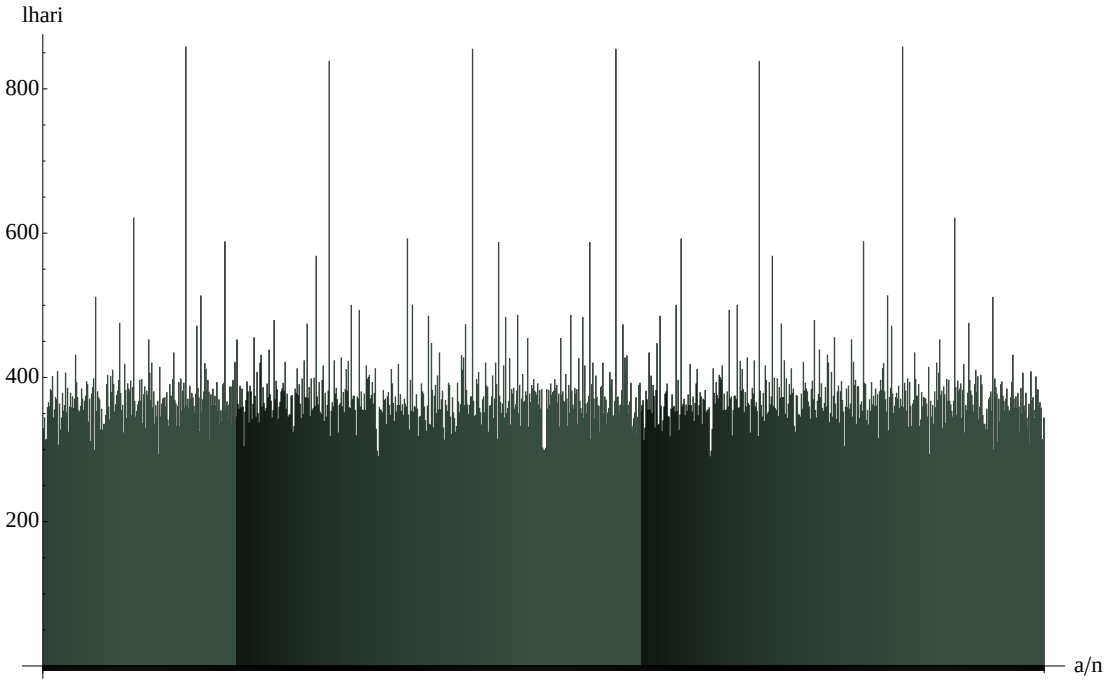
## 2.5 Ostatní algoritmy

### 2.5.1 Volba báze blízko $\frac{n}{2}$

Dalším z algoritmů pro volbu báze, který byl zkoumám, je volba bází blízko středu. Tedy první báze je zvolena jako $\frac{n-1}{2}$ ($n$ je liché, takže jde o přirozené číslo) a další báze jsou voleny vždy o jedna menší číslo, než předešlé. Protože jsou podle lemmatu 1.20 lháři souměrní podle $\frac{n}{2}$, výsledek bude stejný, jako kdybychom začali s volbou báze $\frac{n+1}{2}$ a pokračovali opačným směrem. Tento algoritmus je inspirován pokusem, při kterém byli hledáni všichni lháři všech složených čísel v rozmezí 10 000 až 50 000. Zároveň byl zaznamenáván poměr mezi lhářem a testovaným číslem ($\frac{a}{n}$). V následujícím grafu je vyobrazen výsledek tak, že je interval $[0,1]$ rozdělen na 1000 stejně dlouhých dílů a spočteny počty lhářů, kteří mají poměr $\frac{a}{n}$ v odpovídajícím dílu. Na vodorovné ose je tedy nanášen poměr $\frac{a}{n}$ a na svislou osu počet lhářů s daným poměrem.

Kolem středu vidíme úzkou oblast s menším množstvím lhářů.

Za použití tototo algoritmu bylo při testování čísel z testovací množiny nutno volit bází maximáně pětkrát.




Obrázek 2.1: Hustota lhářů v závislosti na poměru lháře ku testovanému číslu.

### 2.5.2 Volba báze blízko $\frac{n}{3}$

Tento způsob volby bází je také inspirovaný grafem 2.1. Na tomto malém vyobrazení to sice není patrné, ale kolem $\frac{n}{3}$ je také minimum. Dokonce ještě menší, než u $\frac{n}{2}$. Jako báze se tedy volí sestupně čísla menší nebo rovna horní celé části $\frac{n}{3}$. Test s takto volenou bází potřebuje na naší reprezentativní množině až 4 volby báze.

### 2.5.3 Prvočísla obohacená o součiny vybraných lhářů

Další myšlenkou pro vylepšení volby báze je, pokusit se získat nějaké další informace z dosavadního průběhu testu. Pokud je testované číslo složené, pak musí být všechny dosud zvolené báze lháři. V oddílu 1.4 jsou rozebrány případy, kdy je součin lhářů určitě lhářem. Pokud však mají oba lháři nenulový řád, řekněme $k$, a k tomu ještě rozdílnou $2^{k-1} \cdot d$ mocninu, pak jejich součin nemusí být lhářem. Lemma 1.27 sice ukazuje, že pro některá čísla bude tento postup naprosto zbytečný, přesto se nad touto možností zamyslíme trochu více.

Nabízí se tedy možnost vyzkoušet jako přístí bázi součin takových lhářů. Zbývá zjistit, kde k takovým lhářům přijít. Když se nad věcí zamyslíme ještě trochu podrobněji, přijdeme na to, že nepotřebujeme ani lháře se stejným řádem, stačí nám pouze lháři s nenulovými řády. Podle lemmatu 1.21, totiž vidíme, že můžeme postupným mocněním z lháře s vyšším řádem udělat lháře nižšího řádu. Tím získáváme potenciálního svědka za každou dvojici lhářů s nenulovým řádem. Výsledek si navíc můžeme ještě odhadnout prostým vynásobením jednotlivých, již získaných, mocnin.

Námi implementovaný algoritmus si pouze předpočítává poslední mocninu, která nemusí být rovna jedné a pokud je rozdílná od $-1$, tak tuto bázi vyzkouší.

Tento algoritmus sám o sobě nemůže být použit jako volba báze, potřebuje totiž mít již nějaké lháře do začátku. Lze jej ale použít v kombinaci s jiným algoritmem. Vyzkoušíme to na příkladu volby malých prvočísel a náhodné volby.

Vysledky této modifikace však nejsou zdaleka dostačující. V drtivé většině případů byl výsledek horší, než výsledek při použití původní volby báze. Nehledě na to, že v případě, kdy by testované číslo bylo dělitelné prvočíslem typu $4l + 3$, není tento postup použitelný.

## 2.6 Shrnutí

Mezi testovanými algoritmy byly některé prokázány jako nevhodné. Jsou to volby bází tvaru: $1 + (\text{dělitel } n + 1)$, přímo dělitel $n \pm 1$ a pokusy o nalezení svědků mezi součiny lhářů. U ostatních algoritmů voleb byl mezi zvolenými bázemi vždy alespoň jeden svědek. Maximální počet nutných voleb se pohyboval mezi čtyřmi až pěti. Výsledky shrneme v následující tabulce.

| **volba báze, # testů:** | 1 | 2 | 3 | 4 | 5 | 6 | $\sum$ |
|---|---|---|---|---|---|---|---|
| náhodná, průměr | 88920339 | $671,1$ | 24 | $3,3$ | $0,4$ | $0,2$ | $88921769,6$ |
| malá prvočísla | 88920385 | 622 | 29 | 4 | 0 | 0 | 88921732 |
| 1+dělitel $n-1$ | 88920385 | 622 | 22 | 7 | 2 | 0 | 88921733 |
| liché 1+dělitel $n-1$ | 88920385 | 622 | 29 | 0 | 2 | 0 | 88921726 |
| 1+děl. $n-1$ max. mocn. | 88920385 | 621 | 25 | 5 | 2 | 0 | 88921732 |
| prvočísla neděl. $n-1$ | 88920590 | 425 | 18 | 4 | 1 | 0 | 88921515 |
| prvočísla neděl. $n+1$ | 88920203 | 802 | 27 | 6 | 0 | 0 | 88921901 |
| báze blízko $n/2$ | 88920415 | 593 | 27 | 2 | 1 | 0 | 88921695 |
| báze blízko $n/3$ | 88920227 | 781 | 5 | 23 | 2 | 0 | 88921906 |

Tabulka 2.1: Výsledky testů s různými volbami báze na testovací množině.

Pokud bychom algoritmy srovnávali pomocí maximálního počtu smyček k odhalení složenosti čísla, potom se o první místo dělí klasický algoritmus volby malých prvočísel s volbou báze jako prvočísla nedělící $n + 1$. Ostatní dosahují stejného výsledku pěti bází, přičemž náhodný výběr báze dostává průměrnou hodnotu o něco lepší: $4,7$.

Další možností, jak algoritmy hodnotit, je pomocí celkového úhrnu všech smyček nutných k otestování celé testovací množiny. Z tohoto hlediska vede algoritmus volící prvočísla nedělící $n-1$, za ním se umísťuje volba báze blízko $n/2$. Rozdíly v množství použitých bází jsou však v tomto množství testů zanedbatelné, řádově stovky při milionech testů.

Pro zajímavost uveďme podobnou tabulku pro jiný soubor čísel. V článku [7] je uvedena tabulka silných pseudoprvočísel vzhledem k $2, 3$ a $5$ menších než $10^{12}$, které mají maximálně tři faktory. Je nutné si uvědomit, že tato čísla jsou vybrána podle toho, aby na nich klasický přístup volby malých prvočísel selhával. Z toho plyne, že špatné výsledky této volby nebudou žádným překvapením. Zajímavé je, jak si s těmito čísly poradí netradiční volby bází.

Jako nejlepší se v této perpsektivě jeví náhodný test. Druhá je volba báze jako prvočísla nedělící $n-1$. Za zmínku jistě stojí, že báze ve tvaru $1 + (\text{dělitel } n - 1)$ dopadly velmi špatně, dokonce hůř, než klasické volby malých prvočísel. Používání

| **volba báze, # testů:** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | $\sum$ |
|---|---|---|---|---|---|---|---|---|---|
| náhodná, průměr | 88 | 10, 7 | 1, 7 | 0, 5 | 0, 1 | 0 | 0 | 0 | 117 |
| malá prvočísla | 0 | 0 | 0 | 92 | 9 | 0 | 0 | 0 | 413 |
| 1+dělitel $n-1$ | 0 | 0 | 0 | 27 | 60 | 11 | 1 | 2 | 497 |
| liché 1+dělitel $n-1$ | 0 | 0 | 27 | 26 | 42 | 6 | 0 | 0 | 431 |
| 1+děl. $n-1$ max. mocn. | 0 | 36 | 58 | 6 | 1 | 0 | 0 | 0 | 275 |
| prvočísla neděl. $n-1$ | 64 | 33 | 4 | 0 | 0 | 0 | 0 | 0 | 142 |
| prvočísla neděl. $n+1$ | 0 | 7 | 87 | 7 | 0 | 0 | 0 | 0 | 303 |
| báze blízko $n/2$ | 2 | 14 | 81 | 0 | 4 | 0 | 0 | 0 | 319 |
| báze blízko $n/3$ | 0 | 2 | 0 | 3 | 85 | 2 | 4 | 5 | 521 |

Tabulka 2.2: Výsledky testů s různými volbami báze na spsp(2,3,5) $< 10^12$ z článku [7].

pouze maximálních mocnin však tento výsledek poměrně výrazně vylepší. Úplně nejhůř dopadla volba báze blízko $\frac{n}{3}$, což je v zajímavém konstrastu s poměrně dobrými výsledky této volby na předchozím souboru čísel.

U algoritmů založených na bázích tvaru $1 + (\text{dělitel } n-1)$ se neprokázalo, že byla větší pravděpodobnost volby svědka. Na druhé straně ani nebylo nalezeno číslo, které by nemělo mezi takovými bázemi svědka. Dokonce bychom mohli podle pozorování volbu zúžit jen na dělitele s maximálními mocninami dělících prvočísel. Pokud by tato domněnka byla pravdivá, potom by k prvočíselnému testu stačilo vyzkoušet všechny takové báze. Těch je méně než $\log_2 n$, neboť zbytek po každém novém dělení bude menší než polovina předchozího.

V článku F. Arnaulta[4] je popsán heuristický algoritmus pro nalezení silných pseudoprvočísel vzhledem k více bázím. (Připomeňme znovu, že tímto způsobem bylo nalezeno silné pseudoprvočíslo vůči všem prvočíselným bázím menším než 200.) Čísla, která tento algoritmus uvažuje, jsou vždy tvaru $n = p_1 \cdot p_2$, předpokládajíc navíc, že platí

$$p_1 = 2q + 1, \quad p_2 = 4q + 1,$$

kde $q$ je přirozené. Podíváme-li se, jak v tomto případě vypadá $n - 1$, zjistíme, že:

$$p_1 \cdot p_2 - 1 = (p_1 - 1) \cdot (p_2 + 2).$$

Mezi čísly ve tvaru $1 + (\text{dělitel } n-1)$tedy existuje dokonce přímo dělitel testovaného čísla.

# 3. Uživatelská dokumentace

## 3.1 Účel programu

Program slouží k testování hypotéz ohledně alternativních způsobů volby báze v Rabinově-Millerově testu. Jeho účelem není co nejrychlejší testování prvočíselnosti, ani hledání prvočísel, ale poskytnutí uživateli tolik detailních informací o průběhu testů, kolik uživatel žádá.

## 3.2 Spouštění programu

Program není třeba instalovat, ale vyžaduje, aby na počítači byla naistalována Java verze 1.6.0_24-b07-334. Návod k instalaci Javy najde zájemce na internetové stránce `http://www.java.com/en/download/help/download_options.xml`.

Ke spuštění postačí použít spustitelný Java archív nazvaný „RMTestVariousBases.jar“. Tím otevře okno aplikace, jímž se program ovládá. Toto okno má tři záložky. První záložka, pojmenovaná „one test“, slouží k testování čísel, která zadá uživatel přímo do okna aplikace. Druhá záložka testuje čísla, která získá ze souboru zadaného uživatelem. Tato záložka je nazvána „file tester“. Poslední záložka je určena ke generování seznamů silných pseudoprvočísel vzhledem k zadaným bázím, je nazvána „pseudoprimes generator“. Po spuštění programu je aktivní první záložka. V následujících oddílech bude popsán obsah všech tří záložek.

## 3.3 Záložka „one test“

Okno s aktivní záložkou „one test“ je zobrazeno na obrázku 3.1. Popišme si nyní funkce jednotlivých ovládacích prvků zobrazených v okně.

Na prvním řádku je textové pole uvedené štítkem „number:“, toto pole slouží k zadávání čisla, které si uživatel přeje testovat na prvočíselnost. Číslo může uživatel zadat přímo v dekadickém zápisu, nebo jako aritmetický výraz. Tyto výrazy jsou popsány v oddíle 3.8.

V druhém řádku se nachází ovládání způsobu výběru báze a také počet voleb bází. Algoritmus volby báze ja volen z nabídky, která se rozkryje po kliknutí myší na komponentu uvedenou štítkem „how to choose base:“. Při startu je zvolena náhodná volba báze. Jednotlivé algoritmy, které je možné vybrat, jsou rozebrány v oddílu 3.7. Význam a popisek textového pole v druhé části řádku se mění se zvoleným algoritmem pro volbu báze, proto bližší vysvětlení tohoto pole přenecháme kapitole zabývající těmito algoritmy 3.7.

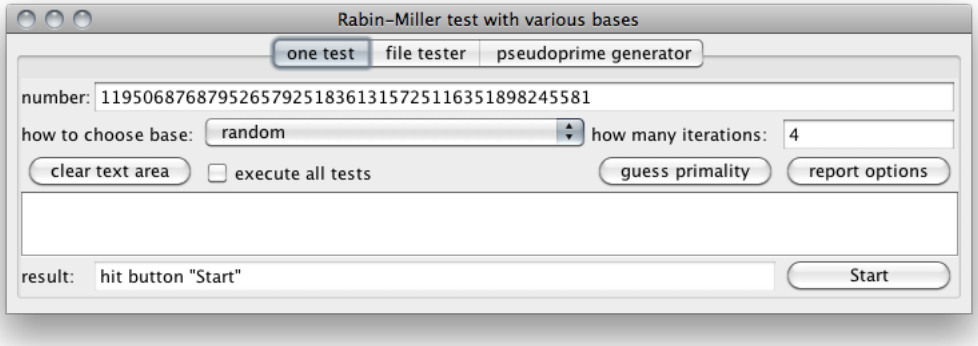
Třetí řádek záložky je věnován ovládacím prvkům výpisů zpráv. První zleva je tlačítko, které čistí textovou oblast pro výpisy z testů. Toto tlačítko nese popis „clear text area“. Hned vedle, jako druhá zleva, se nachází zaškrtávací volba označená „execute all tests“. Pokud se během testu odhalí, že testované číslo je složené, pak není třeba pokračovat. Odhalení složenosti pomocí Rabinova-Millerova testu je totiž definitivní a testované číslo je pak určitě složené. Je zvykem zastavit testování po získání tohoto výsledku. Pokud však uživatele zajímají například

všichni lháři či svědci do určitého limitu, stačí zaškrtout tuto volbu a provedou se všechny předepsané testy nehledě na jejich výsledky.

Zleva třetí se na třetím řádku nalézá tlačítko s názvem „guess primality“. Toto tlačítko spustí vestavěný prvočíselný test jazyka Java na testované číslo. Výsledek se zobrazí v dialogu. Stisknutím potvrzovacího tlačítka dialog opět zmizí. Díky tomuto tlačítku má uživatel na dosah výsledek standartního testu prvočíselnosti. Jako poslední se na třetím řádku nachází tlačítko s textem „report options“. Jeho stiskem vyvolá uživatel dialog, ve kterém může pomocí zaškrtávacích voleb určit, které typy zpráv z průběhu prvočíselného testu jej zajímají a které se vypisovat nemají. O typech zpráv a jejich významu se dá víc dozvědět v kapitole 3.9.

Pod výše zmíněnými ovládacími prvky se rozkládá textová oblast, která slouží k výpisům zpráv z testu. Jak jsme již zmínili, dá se vyčistit pomocí tlačítka. Pokud by byla nějaká část textu mimo zobrazované pole v okně, objeví se na stranách posuvní jezdci pro pohyb zobrazovaného okna v textu. Bez zásahu uživatele budou zobrazovány vždy pouze nejnovětší výpisy.

Spodní část okna slouží k výpisu výsledku testu a k spuštění testu. To první je obstaráno pomocí textového pole uvozeného štítkem „result:“. To druhé zajistí tlačítko s nápisem „Start“. Pokud je testované číslo považováno za prvočíslo, je jako výsledek vypsán text „probably prime“, v případě, že je číslo odhaleno jako složené, je vypsán výsledek „composite“ spolu s určením kolikátý test jej odhalil. Pokud se během výpočtu vyskytne nějaká výjimka, nebo chyba, je na ni uživatel upozorněn zprávou v textové oblasti. Souhrnně o tom pojednává oddíl 3.10.




Obrázek 3.1: Založka „one test“.

## 3.4 Záložka „file tester“

Druhá záložka, jak název napovídá, je určena k testování celých souborů čísel. Kromě některých změn je ale velmi podobná předešlé záložce.

První řádek slouží k určení souboru ve kterém má program hledat čísla k testování. Po stisknutí tlačítka „choose file“, které je úplně vpravo, se objeví dialog pro vybírání souboru. Když uživatel soubor označí a potvrdí, dialog zmizí. Jméno zadaného souboru se vypíše do textového pole uvedeního štítkem „file with numbers:“. Podporovaný formát tohoto souboru je popsán v oddílu 3.5.

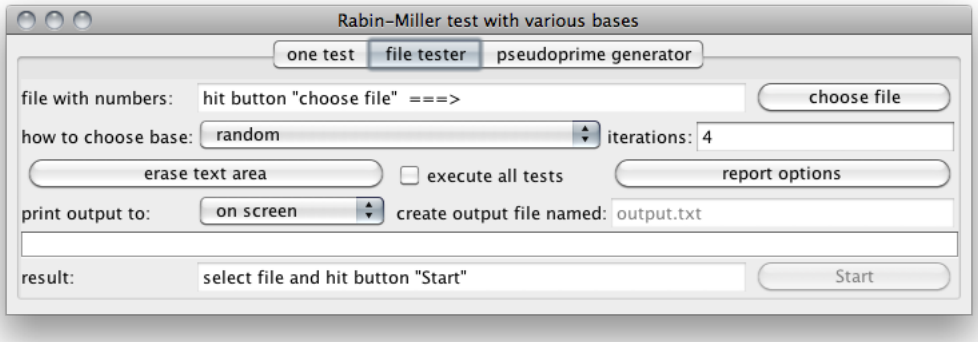
Druhý řádek, stejně jako v první záložce, slouží k určení způsobu volby bází v testu a jejich počtu a je navlas stejný jako odpovíjící řádek první záložky, proto se jím dále nebudeme zabývat.

Také třetí řádek je velmi podobný odpovídajícímu řádku v dříve rozebírané záložce. Jediným rozdílem je absence tlačítka s názvem „guess primality“, které by tu nemělo význam.

Čtvrtý řádek je nový vzhledem k první záložce. V souboru může být velmi mnoho čísel k testování mohlo by být velmi nepraktické vypisovat průběhy těchto testů na obrazovku. Proto si může uživatel zvolit, jestli si přeje výpis zobrazit do textové oblasti, nebo do souboru, popřípadě obojí. Pokud se má zapisovat také do souboru, musí uživatel určit jméno souboru, který se má vytvořit. To provede vepsání žádaného jména do textového pole v pravé části tohoto řádku.

Stejně jako v prvním případě se pod uvedenými ovládacími prvky nachází textová oblast, do které se mohou vypisovat zprávy pro uživatele. Poslední řádek plní stejnou funkci jako v předchozí záložce. Jediným rozdílem je, že v textovém poli se nevypisuje složenost či prvočíselnost testovaného čísla, ale zpráva o průběhu testu. Případná chybová hlášení jsou vypisována do textové oblasti.

Vzhled okna, když je tato záložka aktivní je zobrazen na obrázku 3.2.




Obrázek 3.2: Záložka „file tester“.

## 3.5 Formát vstupního souboru

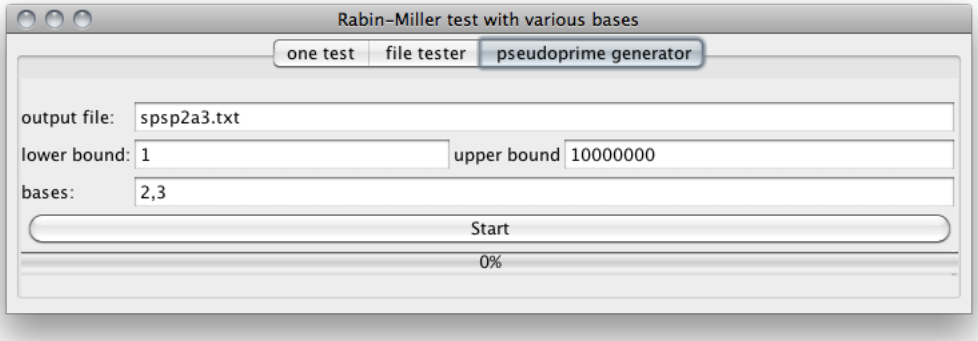
Vstupní soubor má obsahovat množinu čísel, které si uživatel přeje otestovat. Každé číslo má být na samostatném řádku. Čísla jsou zadávána v dekadickém zápisu, či jako aritmetické výrazy. Řádek začínající znakem středníkem je považován za komentář a program jej ignoruje. Ostatní řádky, které nemohou být převedeny na číslo, jsou také ignorovány, ale program o tom podává zprávu.

## 3.6 Záložka „pseudoprime generator“

Třetí záložka slouží k generování množin silných pseudoprvočísel vzhledem k zadaným bázím. Uživatel zvolí v jakém rozmezí mají čísla být a které báze se mají uvažovat. Program podle toho vypíše do souboru čísla v odpovídajícím rozmezí,

která jsou složená a zadané báze jsou vzhledem k nim lháři. Okno s touto záložkou je vyobrazeno na obrázku 3.3.

V textovém poli na prvním řádku uživatel zadá jméno souboru, do kterého má program uložit výsledky. Na druhém řádku určí uživatel rozmezí čísel. Maximální hodnota horní meze je ale 200 000 000. Potom se zadávají báze tak, že jsou vepsány do textového pole na třetím řádku formuláře. Jednotlivé báze se na tomto řádku oddělují čárkou. Báze mohou být zadány i ve formě aritmetických výrazů. Předposlední řádek je zabrán tlačítkem, které spustí výpočet. Dole se nachází ukazatel postupu. Po splnění úkolu je uživateli ohlášen výsledek v dialogovém okně.




Obrázek 3.3: Záložka „pseudoprime generator“.

## 3.7 Algoritmy pro volbu báze

Uživatel volí jeden ze seznamu algoritmů pro volbu bází. Kvůli přehlednosti jsou v seznamu algoritmy uvedeny pod zkrácenými názvy. V následujících odstavcích bude dopodrobna popsáno, co který značí. Volba algoritmu také mění význam druhého pole na stejném řádku. Ve většině případů však pole značí počet iterací testu, tedy počet voleb báze. Proto, pokud není řečeno jinak, má obsah druhého pole tento význam. Proměnná $n$ má význam testovaného čísla.

### 3.7.1 „random“

Tato volba znamená, že báze budou voleny náhodně v rozmezí $1, \ldots, n-1$. Při většině teoretických popisů tohoto algoritmu a jeho účinnosti se bere v potaz právě náhodná volba báze. Proto je to jedna z dobrých voleb pro porovnání a kontrolu úspěšnosti algoritmů pro volbu báze.

### 3.7.2 „small primes“

Za báze se volí postupně malá prvočísla. Začne se nejmenším a postupuje se k větším. Tento postup je v praxi asi nejpoužívanější. Z tohoto důvodu je tento algritmus rozvněž dobrá volba pro porovnávání úspěšnosti algoritmů.

### 3.7.3 „given base“

Pokud je zvolena tato možnost, volí se za bázi pouze číslo zadané v druhém textovém poli, které je v tomto případě nadepsáno „base:“.

### 3.7.4 „number under limit“

Tento algoritmus volí vzestupně celá čísla začínajíc číslem 2. Pole vedle značí limit do kterého se mají otestovat všechny báze, toto pole je tedy uvozeno „limit:“. Tuto volbu ocení uživatel pokud jej zajímá například nejmenší svědek.

### 3.7.5 „(divisors of n-1)+1“

V tomto algoritmu volíme za báze dělitele čísla $n - 1$ zvětšené o jedna. První zvolenou bází je číslo 2. Pokud si jej představíme jako $1 + 1$, vidíme, že odpovídá algoritmu, neboť jednička je nejmenší číslo dělící $n - 1$. Potom se najde první prvočíselný dělitel čísla $n - 1$, tedy dvojka, a určí se jeho valuace (maximální mocnina, která je ještě dělitelem). Během následujících voleb bází se používají předchystaní dělitelé ve formě mocnin tohoto prvočísla. V momentě, kdy dojdou, zkoušíme dělit $n-1$ následujícími pvočísly. Jakmile najdeme nového pročíselného dělitele, opět zjistíme jeho valuaci. Zbývající dělitele, které jsou součinem dosud odhalených dělitelů, vytvoříme tak, že příslušné mocniny pravě nalezeného prvočísla vynásobíme s již použitými děliteli. Takto si vytvoříme novou zásobu ještě nepoužitých dělitelů čísla $n - 1$. Ty potom používáme tak, že je zvětšíme o jedna a označíme jako bázi. Postup opakujeme s novým prvočíselným dělitelem, dokud je potřeba, nebo dokud nedojdou dělitelé. Pokud by k tomu došlo, nastala by příslušná výjimka.

### 3.7.6 „(divisors of n+1)+1“

Když je nastaven tento způsob volby báze, provádí se ten samý postup jako v případě algoritmu z 3.7.5 kromě toho, že se hledají dělitelé čísla $n + 1$.

### 3.7.7 „(even divisors of n-1) + 1“

Tento algortimus volí stejné báze jako algoritmus 3.7.5 s tím rozdílem, že vybírá pouze sudé dělitele, aby výsledná báze byla lichá.

### 3.7.8 „(even div. of n-1 maximal pow)+1“

Opět jen lehká modifikace předchozího algoritmu. Rozdíl je v tom, že se vždy používá maximální mocnina prvočíselného dělitele a mocnina dvou je použita vždy. Vybraní dělitelé jsou tedy sudí.

### 3.7.9 „(primes dividing (n-1))+1“

Za báze jsou volena postupně malá prvočísla dělící $n - 1$ zvětšená o jedna.

### 3.7.10 „numbers around n/2“

První zvolenou bází je $\frac{n-1}{2}$, dále jsou voleny báze vždy o jedna menší, než předešlá.

### 3.7.11 „numbers around n/3“

Začínajíc na $\lceil \frac{n}{3} \rceil$, pokračují volené báze sestupně s krokem jedna.

### 3.7.12 „primes not dividing n-1“

Tato volba způsobí, že se za báze budou volit malá prvočísla stejně jako 3.7.2, ale budou vynechána ta prvočísla, která dělí o jedna zmenšené testované číslo.

### 3.7.13 „primes not dividing n+1“

Analogické předchozí volbě. Rozdíl je pouze v tom, že jsou vynecháváni dělitelé $n + 1$.

### 3.7.14 „primes and multiple of tested liars“

Při této volbě jsou za báze voleny malé prvočísla, jako při volbě „small primes“, rozdíl spočívá v tom, že pokud se vyskytnou dva lháři, jejichž součin má potenciál být svědkem, je jako příští báze vybrán právě tento součin.

### 3.7.15 „random numbers and multiple of tested liars“

Chová analogicky předchozí volbě, pokud však nemá k dispozici vhodný součin, volí za báze náhodná čísla.

### 3.7.16 „from formula“

Způsob volby báze také může být také zadán pomocí algebraických výrazů v závislosti na testovaném čísle. Sekundární pole bude v tomto případě nadepsáno „formula:“. Nejprve se specifikuje kolik iterací volby báze má být provedeno. To se určí tak, že údaj v tomto poli bude začínat tímto číslem a pokračuje znakem „x“. Pokud nebude počet iterací specifikován provede se jenom jeden test. Po zadání počtu iterací, následuje samotný výraz. Uživatel má k dispozici všechny operátory akceptované v algebraických výrazech a operátor dělení `/` se zaorkouhlováním dolů. Navíc ještě může být použito písmeno `n`, které značí testované číslo, nebo také písmeno `p`, značící prvočíslo. Během opakovaných iterací se význam znaku `p` ve výrazu bude měnit vzestupně přes všechny prvočísla. Nejsnáze se funkce těchto výrazů osvětlí na příkladech:

- `4x p` znamená, že se za báze budou používat postupně první čtyři prvočísla.
- `7x p^2 + 1` tato formula zajistí volbu báze ve tvaru prvočíslo na druhou plus jedna, sedm iterací.
- `n/4` jeden test, s bází jako dolní celá část výsledeku tohoto dělení.

Výsledné báze jsou brány modulo testované číslo.

## 3.8 Akceptované aritmetické výrazy

Pro větší pohodlí při práci s programem je možné zadávat některá čísla do programu prostřednictvím aritmetických výrazů. Aritmetický výraz se zapisuje pomocí operátorů `+`, `-`, `*` a `^`. Priority jsou nastaveny obvyklým způsobem, tedy nejvyšší prioritu má operátor `^`, za ním následuje `*`, s nejnižší prioritou jsou potom operátory `+` a `-`. Uvedené operátory mají běžný význam, tedy mocnění, násobení, sčítání a odčítání. Pro závorkování je použito běžných kulatých závorek `(` a `)`. Uživatel může těchto výrazů použít pro zadávání testovaných čísel jak v grafickém okně tak v souborech. Pokud se programu nepovede převést aritmetický výraz na číslo, bude to ohlášeno uživateli, jak je to popsáno v části 3.10.

## 3.9 volby v „report options“

Protože byl program určen k testování nových hypotéz, předpokládá se situace, kdy se bude uživatel zajímat o nějaký detail průběhu výpočtu, zatímco ostatní zprávy jej zajímat nebudou. Při dalším testu to však může být úplně jinak, protože jiná hypotéza může pracovat s úplně jinými detaily. Za těchto předpokladů se pro výpisy informací z testů nehodí klasický systém pro loggování, který zprávy třídí hierarchicky podle důležitosti a uživatel určuje úroveň důležitosti, od které májí být zprávy vypisovány. Požadavkům, zdá se, mnohem lépe odpovídá jednodušší model ve kterém každá zpráva má svůj typ a uživatel rozhodne o každém typu zpráv zvlášť, jestli má být vypisován. Jednotlivé typy zpráv s jejich významem jsou popsány v tabulce 3.1.

## 3.10 Chybové hlášení

V případě, že je programu poskytnut chybný vstup, nebo se vyskytne problém s přístupem k souboru, bývá obvykle vlákno výpočtu ukončeno. O tom je ale nutno podat zprávu uživateli.

V případě procesů z prvních dvou záložek jsou chybové zprávy vypisovány do textových oblastí mezi ostatní zprávy. Zprávy o chybách, které vznikly během převádění dat od uživatele na vstupy pro samotné testy, jsou vypisovýny přímo bez ohledu na to, jaké je nastavení zpráv v dialogu pod tlačítkem „report options“. Tyto chyby jsou obvykle způsobeny nekorektním zadáním číselných hodnot v grafickém rozhraní. Naproti tomu zprávy, které vznikly během testu už podléhají teto volbě a pokud uživatel nevybere typy `ERROR_MESSAGE` a `EXCEPTION_MESSAGE` mezi vypisované, nebudou mu tyto zprávy ukázány. Příkladem takové chyby může být výjimka vyvolaná vyčerpáním možností volby báze testovaného čísla při použití zadaného algoritmu.

| | |
|---|---|
| BASECHOOSER_ADJUSTED | Modul pro volby báze je připraven generovat čísla vztahující se k novému číslu. |
| BASECHOOSER_EXHAUSTED | Veškeré báze možné zvolit podle současně nastaveného algoritmu byly použity. |
| BASECHOOSER_NEW_PRIMEDIVISOR_FOUNDED | Byl nalezen nový prvočíselný dělitel čísla $n-1$. |
| BASE_CHOSED | Byla zvolena báze. |
| NEW_NUMBER_TO_TEST | Odteď bude testováno nové číslo. |
| PARTIAL_FACTORIZATION_OF_N1 | Výpis rozkladu čísla $n-1$ do formy $2^k \cdot m$, kde $m$ je liché. |
| TRIVIAL_CASE_PRIME | Testované číslo bylo shledáno prvočíslem bez testovaní. Je to tedy 2 nebo 3. |
| TRIVIAL_CASE_COMPOSITE | Testované číslo bylo shledáno složeným bez testování. Buď je sudé, nebo byl nalezen jeho dělitel. |
| REMAINDER_POW_M | Zpráva zveřejňující výsledek po umocnění báze na exponent $m$ modulo $n$. |
| REMAINDER_POW_M_SQUARED | Výslede po mocnění báze na exponent $2^l \cdot m$ pro nějaké $l$. |
| RESULT_DETERMINED_PRIME | Tato smyčka testu dospěla k výsledku „Pravděpodobně prvočíslo“. |
| RESULT_DETERMINED_COMPOSITE | Tato smyčka testu odhalila složenost. |
| EXCEPTION_MESSAGE | Během testu došlo k výjimce. |
| ERROR_MESSAGE | Během testu došlo k chybě. |
| OTHER | Zpráva, která není zahrnuta v jiných kategoriích. |

Tabulka 3.1: Typy zpráv.

# 4. Programátorská dokumentace

Tato kapitola je věnována popisu samotného programu. Nejdříve je rozebráno rozčlenění programu, později jsou uvedeny návody na jednoduché úpravy programu. V závěru jsou předloženy možné budoucí vylepšení programu.

## 4.1 Analýza požadavků

Jak již bylo zmíněno v předešlé kapitole tento program byl vytvořen za účelem testování úspěšnosti různých voleb báze Rabinova-Millerova testu. Požadavky na jeho funkce by se daly shrnout do následujících bodů:

- Velikost čísel, se kterými program pracuje, by neměla být omezená.
- Program bude muset umět vykonávat testy s různými volbami báze se snadnou implementací dalších možností pro volbu báze.
- Uživatel by měl mít dobrou kontrolu nad povahou výstupů programu a zpráv o průběhu testů.
- Možnost snadno testovat větší množství čísel, bez účasti uživatele.

Z bodů dva a tři logicky plyne rozdělení testu do tří částí. Tyto části komunikují skrz rozhraní. První část obstarává předávání informací o průběhu testu uživateli. Druhá samostatná část je zodpovědná za volbu báze a poslední část má na starost samotné výkonání testu. Jazykem, který byl použit pro tvorbu tohoto programu je Java. Pro operace s dlouhými čísly je používána třída `java.math.BigInteger`, která vyhovuje požadavku z prvního bodu. Její výhodou je i podpora operací z modulární aritmetiky.

## 4.2 Struktura programu

Algoritmus samotného testu je podrobně popsán v oddílu 1.3, a proto se ním nebudeme teď zabývat. Soustředíme se na uspořádnání programu. Program je rozdělen do pěti balíků tříd. První tři balíky logicky odpovídají částem testu nastíněných v předešlé kapitole. Zbylé dva balíky se týkají uživatelského rozhraní a prvočíselné databáze.

První je pojmenován „`base`“, jsou v něm převážně třídy řídící výpočet testů. Další balík je věnován třídám implementujícím algoritmy pro volbu bází pojmenovaný „`baseChoosers`“. Následuje skupina tříd obsluhující vypisování zpráv sdružená do balíku „`reporters`“. Předposlední je balík s uživatelským rozhraním „`userInterface`“ a poslední balík je věnován převážně databázi prvočísel.

Zmiňme dvě ústřední rozhraní. Prvním z nich je rozhraní `BaseChooser`. Jak název napovídá, toto rozhraní je implementováno třídami reprezentujícími algoritmy pro volby báze. Důležité je i rozhraní `Reporter`, které je implementováno třídami přebírájícími zprávy od ostatních částí programu. Pro snažší vyjadřování o těchto třídách si zavedeme následující označení.

- Třídu implementující rozhraní `BaseChooser` nazveme „hledač bází“.

- O třídě, která implementuje rozhraní `Reporter` budeme mluvit jako o „reportérovi“.

Přibližně v programu probíhá Rabinovův-Millerovův test takto:

1. Nejprve je zajištěn reportér, aby bylo kam předávat zprávy.
2. Potom je vytvořen hledač bází.
3. Nyní již může být podle potřeby iterována samotná volba báze pomocí hledače báze, potom proveden výpočet příslušných mocnin báze a předání výsledků reportérovi.
4. Po získání výsledku nebo vyčerpání iterací je testování ukončeno.

V následujících sekcích bude rozebrán obsah každého balíku.

## 4.3 Balík `rabinMillerTestVBC.base`

Tento balík obsahuje pouze tři třídy: `RMTestVBCmain`, `RMTest`, `RMTester`. Třída `RMTest` má jedinou metodu implementující samotný algoritmus Rabinova-Millerova testu, jak je popsán v části 1.3. Před vykonáním prvního kroku algoritmu se ještě program ujistí o nesoudělnosti zvolené báze s testovaným číslem prostřednictvím metody `BigInteger.gcd`.

Třída `RMTester` slouží k testování více čísel v nepřerušeném sledu. Tento úkol plní metoda `RMTester.testNumbersInFile`, která načítá čísla ze souboru a testuje je. Pokud je řádek označen jako komentář pomocí středníku, je ignorován. Řádek, který není označen jako komentář a nedá se převést jako aritmetický výraz na číslo, je také ignorován, ale je o tom podána zpráva reportérovi. Typ této zprávy je `ReportType.OTHER`. Potom je zavolána příslušná metoda pro přípravu hledače bází na vybírání bází vzhledem k tomuto číslu. Postupně jsou volány jednotlivé testy se zvolenými bázemi.

Třída `RMTestVBCmain` obsahuje metodu `main`, která je zavolána po spuštění programu. V této metodě je vytvořeno spuštěno grafického prostředí.

## 4.4 Balík `rabinMillerTestVBC.baseChoosers`

Tento balík obsahuje třídy implementující algoritmy pro volbu bází a některé pomocné třídy. Třídy zajišťující volbu báze implementují rozhraní `BaseChooser`. Zmíněné rozhraní vyžaduje implementaci dvou metod: `adjustToNum` a `nextBase`. Metoda `adjustToNum` slouží k tomu, aby připravila instanci třídy na generování bází vzhledem k určitému číslu, zatímco druhá metoda vrací zvolenou bázi. Metoda `nextBase` může vytvořit výjimku, pokud jsou již použity všechny báze odpovídající danému algoritmu. Obě metody mají jako jeden z argumentů repotéra, kterému předávají zprávy o své činnosti.

Do tohoto balíku patří také třída `PartialFactorization`, ačkoliv není hledačem bází. Slouží k implementaci částečného rozkladu daného čísla. Tato třída je využita v ústřední metodě `RMTest.oneTest`, která potřebuje rozložit $n-1$ do tvaru $2^k \cdot d$ (kde $d$ je liché). Proto je již v construktoru proveden pokus o faktorizaci pomocí prvočísla 2. Třída je však využívána i v některých hledačích báze

založených na práci s částečnou faktorizací čísla $n - 1$. Zároveň si třída udržuje seznam dosud zjištěných dělitelů. K dalšímu rozvinutí rozkladu slouží metoda `PartialFactorization.tryPrime`, která se pokusí zadaným (prvo)číslem doplnit rozklad. Pokud je tento postup úspěšný, vrátí hodnotu `TRUE`, pokud ne, vrátí `FALSE`.

Balíku obsahuje ještě další třídy které nejsou hledači bází, ale jsou natolik jednoduché, že nestojí za zmínku. Většina tříd ale implementuje jednotlivé algoritmy pro volbu báze, jak je popsáno v oddíle 3.7.

Pokud by hledač bází potřeboval k volbě báze i informace z průběhu předchozích testů, dá se tato situace vyřešit tak, že bude sám o sobě také reportérem, kterému budou posílány zprávy o průběhu testu. Takto tomu je v případě hledače báze `BCmultipleOfTwoLiarsWSimOrder`. V případě použití tohoto hledače bází tedy nesmí být zapomenuto zapojit jej do programu také jako reportéra, aby dostával pro něj nutnou zpětnou vazbu.

Nejvšestrannějším hledačem báze je hledač zadávaný z formule. Ten je implementován třídou `BCFromExpr`. Algebraické výrazy zadávané uživatelem jsou zpracovávány do formy binárního stromu z instancí tříd `BaseExpressionNode`. Třídy implementující základní aritmetické operátory do tohoto rozhranní jsou sdruženy do podbalíku `baseExpression`.

Převedení uživatelova zápisu na strom se dějě ve dvou fázích. První má na svědomí metoda třídy `ExpressionCruncher` nazvaná `evaluateBaseExpression`. Výsledkem této metody je zatím nefunkční strom, který má ve vrcholech uloženy jména budoucích operátorů. Vyrcholy jsou zatím reprezentovány třídou `BaseExpressionNodeDummy`. Prozatimní strom je později převeden na plně funkční s vrcholy typu `BaseExpressionNode` v konstruktoru hledače bází. Tento postup nám umožňuje používat při vlastní konstrukci stromu vnitřní třídy hledače bází `BCFromExpr`, které reprezentují například funkci `p` v používaných algebraických výrazech.

Tabulka 4.1 popisuje funkce jednotlivých tříd hledačů bází.

## 4.5 Balík `rabinMillerTestVBC.reporters`

Balíček `reporters` sdružuje reportéry. Ti slouží k předávání zpráv o průběhu výpočtu mezi programem v uživatelem. Aby bylo snažší se ve zprávách orientovat je zaveden výčtový typ `ReportType`, který dodává informaci o povaze zprávy. Jednotlivé typy zpráv jsou popsány v tabulce v oddílu 3.9.

Přibližme si nyní rozhraní `Reporter`, které reportéry definuje. `Reporter` má tři metody: `Reporter.write`, `Reporter.report`, `Reporter.end`.

Největší roli hraje druhá metoda, tedy `Reporter.report`. Přijímá dva argumenty, první typu `ReportType` a druhá typu `String`, samotnou zprávu. To umožňuje reportérovi poznat o jaký typ zprávy jde a zachovat se podle toho.

Metoda `write` slouží k posílání zpráv, které nejsou označeny žádným typem, tedy zprávy nepřicházející přímo z výpočtu, jako třeba úhrnné informace o proběhlém testu. Při současné implementaci je tato metoda použita pouze třídou `StatReporter`.

Poslední metoda slouží k správnému ukončení zápisu, například pokud byly používány soubory. V případě, že by nastal nějaký problém, zpravidla se soubory, vyhodí tato metoda výjimku, aby na to upozornila.

| | |
|---|---|
| `BCconstant` | Volí jako zadané číslo. |
| `BCFromExpr` | Báze podle algebracikých výrazů podle 3.7.16. |
| `BCmultipleOfTwoLiars` | Nutno složit s dalším hledačem bází. Ten je potom obohacen p součin použitých bází s potenciálem být svědkem. |
| `BCNearMiddle` | Báze postupně o jedna menší začínající číslem $\frac{n-1}{2}$ . |
| `BCNearOneThird` | Volí báze prstencovitě kolem $\lfloor \frac{n}{3} \rfloor$ |
| `BCNearOneThird2` | Volí báze sestupně od $\lceil \frac{n}{3} \rceil$ |
| `BCNumberUnderLimit` | Volí všechna čísla vzestupně počínajíc dvojkou. |
| `BCPFNMO` | Jako báze volí dělitele $(n-1)$. |
| `BCPFNMOPOimproved` | Báze volí ve tvaru $1+($ dělitele $(n+1))$, pouze sudé. |
| `BCPFNMOPOmaxPow` | Báze volí ve tvaru $1+($ dělitele s maximálními mocninami $(n+1))$, pouze sudé. Používá však vždy všechny dosud nalezené prvočíselné dělitele. |
| `BCPFNMOPOmaxPowAll` | Báze volí ve tvaru $1+($ dělitele s maximálními mocninami $(n+1))$, pouze sudé. |
| `BCPFNPO` | Jako báze volí dělitele $(n+1)$. |
| `BCPFNPOPlusOne` | Jako báze volí dělitele $(n+1)$ zvětšené o jedna. |
| `BCPFNPOPOmaxPow` | Dělitelé čísla $n+1$ s maximální valuací použitých prvočísel, zvětšení o jedna. Používá všechny objevené dělitele. |
| `BCPFNPOPOmaxPowAll` | Dělitelé čísla $n+1$ s maximální valuací použitých prvočísel, zvětšení o jedna. |
| `BCPFOfNMOPlusOne` | Dělitelé čísla $n-1$ zvětšení o jedna. |
| `BCPrimeDivN1` | Bázemi jsou prvočísla dělící $n-1$. |
| `BCPrimeDivN1PO` | Bázemi jsou prvočísla dělící $n-1$ zvětšená o jedna. |
| `BCPrimesNotDivN1` | Bázemi jsou prvočísla nedělící $n-1$. |
| `BCPrimesDivNPO` | Bázemi jsou prvočísla nedělící $n+1$ zvětšená o jedna. |
| `BCprimesUnderLimit` | Malá prvočísla vzestupně od dvojky. |
| `BCrandom` | Náhodné číslo. |

Tabulka 4.1: Třídy hledačů bází.

Účel jednotlivých reportérů si shrňme v tabulce 4.2.

Popis chování některých reportérů by byl pro tabulku příliš dlouhý, a proto se k nim vrátíme ještě v následujícím odstavci.

`SievedReporter` propouští jenom zprávy s vybranými typy zpráv. To je specifikováno pomocí instance třídy `Map<ReportType,Boolean>` která je předána jako parametr construktoru. Pokud je hodnota příslušného typu zprávy `TRUE`, pak je zpráva posílána dál.

`HeadReporter` nechává obsah některých zpráv. Nemění se znění zpráv typů `BASE_CHOOSED`, `BASECHOOSER_ADJUSTED`, `RESULT_DETERMINED_COMPOSITE`, `RESULT_DETERMINED_PRIME`, `TRIVIAL_CASE_COMPOSITE` a `TRIVIAL_CASE_PRIME`.

Závěrem popíšeme, jakým způsobem jsou reportéři používaní v grafickém prostředí. Reportéři často slouží jen k drobné změně textu a výsledek předávají dalšímu reportérovi. Reportér, který zprávy přijímá, bude nazýván vnitřním reportérem. Na začátku výpočtu se vytvoří instance buď třídy `PublishReporter` nebo `SimpleFileReporter`, popřípadě jsou oba zahrnuti pod `MultipleReporter`. Ten se potom použije jako vnitřní reportér pro reportéra `HeadReporter`, aby byl uživatel informován o povaze zpráv. Před něj je ještě postaven `SievedReporter`,

| | |
|---|---|
| `SimpleFileReporter` | Vypisuje zprávy do souboru zadaného v konstruktoru. |
| `StdOutReporter` | Vypisuje zprávy na standartní výstup. |
| `VoidReporter` | Zprávy nevypisuje nikam. |
| `SievedReporter` | Propouší zprávy reportérovi zadanému v construktoru podle nastaveného klíče. |
| `HeadReporter` | Před každou zprávu představí její typ, kromě výjimek. Tyto ohlavičkované zprávy pošle specifikovaném v konstruktoru reportérovi. |
| `MultipleReporter` | Tento reportér předává zprávy vícero reportérům, kteří byli zadáni v konstruktoru. |
| `StatsReporter` | Předává zprávy vnitřnímu reportérovi, po zavolání metody `end` mu pošle krátké statistiky o proběhlém testu metodou `write` a také jej ukončí. |
| `LiarsReporter` | Zapisuje, na řádky, do zadaného souboru testovaná čísla a báze dávající výsledek „asi prvočíslo“. Řádek s testovaným číslem je uveden znakem „$“. |
| `PublishReporter` | Tato třída je vnitřní třídou v `GUIframe` a vypisuje zprávy do textové oblasti. |

Tabulka 4.2: Názvy a funkce reportérů.

který odstíní nezajímavé zprávy. Jako úplně první stojí `StatsReporter`. Zpráva je tedy zpracována reportéry v tomto pořadí: `StatsReporter`, `SievedReporter`, `HeadReporter`(, `MultipleReporter` ) a `PublishReporter` či `SimpleFileRep`. Poznamenejme ještě, že `PublishRep` je vnitřní třída tříd `OneRMTestWorker` a `FileRMTestWorker`.

## 4.6 Balík `rabinMillerTestVBC.userInterface`

Program je ovládán uživatelem pomocí grafického okna, a proto jsou do tohoto balíčku zařazeny třídy, které toto okno používá. Ačkoliv je kód většiny z těchto tříd poměrně dlouhý, myšlenkově jsou velmi jednoduché. Ve většině případů se nejedná o nic jiného, než přizpůsobení standartních tříd z `javax.swing`.

Samo okno je instancí třídy `GUIframe`, která je potomek třídy `JFrame`, v něm jsou pak vytvořeny instance všech tří ovládacích panelů.

Záložka „one test“ je tvořena třídou `OneTestPanel`, která je samozřejmě potomkem třídy `JPanel`. Když uživatel spustí test, je vytvořena instance třídy `OneRMTestWorker`, která test provádí. Tato třída je přirozeně potomkem třídy `SwingWorker`.

Zprávy z výpočtu jsou předávány pomocí `SievedReporter`u a `HeadReporter`u do `PublishReporter`u, který je přidá do textové oblasti záložky pomocí metody `SwingWorker.process`. Chybové hlášení z procesu probíhá buď prostřednictvím zvoleného reportéra, nebo je metoda `OneRMTestWorker.doInBackground` ukončena a chybové hlášení je vypsáno v metodě `OneRMTestWorker.done`.

Třída `FileTestPanel` vytváří obsah záložky „file tester“ a pro výpočet používá instance třídy `FileRMTestWorker`. Všechny vlastnosti jsou analogické první záložce, jedinou komplikací jsou různé možnosti reportérů.

Poslední ovládací panel, `pseudoprime generator` je uskutečněn pomocí třídy

`UtilPanel` (potomek `JPanelu`). Výpočet je dále předán třídě `CreateSPSPWorker`. Samotný proces vytváření množin silných pseudoprvočísel probíhá tak, že jsou testována složená čísla v daném rozashu. Pokud dané číslo projde testy s vybranými bázemi, je zapsáno do výstupního souboru.

K zpracování aritmetických výrazů slouží třída `ExpressionCruncher`. Pokud není výraz korektní, je vytvořena vyjímka typu `ExpressionException`.

## 4.7 Balík `rabinMillerTestVBC.primeDatabase`

Pro hledání dělitelů čísel je dobré mít připravenu databázi prvočísel. V tomto programu tuto úlohu zastává třída `PrimeDatabase`. Obsah databáze je vytvořen pomocí algoritmu Eratosthenova síta, podle oddílu 1.2.1. Databáze je v paměti počítače reprezentována jako pole hodnot typu `boolean`. Každému lichému číslu menšímu než limit 200 000 000 odpovícá jeden prvek pole. Pokud je na patřičném místě v poli hodnota `TRUE`, je dané číslo prvočíslem. Složené číslo má samozřejmě hodnotu `FALSE`. Toto pole je vytvářeno v prvním momentě potřeby. Tedy většinou na začátku výpočtu jako součást konstruktoru hledače bází. Program má tuto strukturu pro čísla až do 200 000 000. Toto pole je společné pro všechny instance této třídy. Pro potřebu testování čísel v tomto rozsahu tedy stačí přečíst hodnotu v příslušném poli. K tomu slouží metoda `PrimeDatabase.isPrime`. Protože je rozsah databáze omezen víc než rozsah typu `Integer`, pracuje databáze pouze s tímto typem.

V hledačích báze, kteří potřebují částečnou faktorizaci nějakého čísla, se používá také metoda `PrimeDatabase.getNextPrime`. Podle očekávání vrací prvočísla od nejmenšího k větším začínajíc trojkou. Pokud by se chtěl uživatel vrátit na začátek, slouží k tomu metoda `reset`.

Situace, kdy se uživatel zeptá na prvočíslo mimo povolený rozsah, nebo již vyčerpal všechny prvočísla pomocí metody `getNextPrime`, je řešena pomocí vyjímky typu `PrimeDatabaseException`.

## 4.8 Návody k rozšíření programu

V této sekci je připravených několik návodů, jak funkce programu rozšířit.

### 4.8.1 Jak začlenit nového hledače báze do programu

Prvním krokem bude naprogramování samotné třídy hledače bází. Musí být naimplementovány metody rozhraní `BaseChooser`. To však není předmětem tohoto návodu, protože obsah těchto metod je dán implementovaným algoritmem. Zdůrazněme, že při psaní těchto metod autor nesmí zapomínat na hlášení jednotlivých kroků a výsledků reportérovi, kterého tyto metody dostávají jako parametr. Rozhodně musí hledač bází vypisovat zprávy typu `Base_Choosed` v těle metody `BaseChooser.nextBase` a `BaseChooser_Adjusted` v metodě `adjustToNum`.

Jakmile je toto hotovo, stačí k využití algoritmu v programu předat instanci této třídy jako argument metody `RMtest.oneTest`. Dále je ale potřeba začlenit tento algoritmus do grafického prostředí. K tomu bude nutné modifikovat

třídy `OneTestPane`, `FileTestPane` a příslušné třídy vykonávající samotné výpočty `OneRMTestWorker` a `FileRMTestWorker`. Postup popíšeme v následujících bodech:

- Třídy `OneTestPane` a `FileTestPane` mají mezi atributy pole textových řetězců s názvem „algorithmsOT“, které uchovávají jména implementovaných hledačů bází. Připišme tedy jméno nového algoritmu do těchto polí.

- Aby mohlo grafické prostředí reagovat na volbu nového hledače bází musíme upravit metody tříd panelů `OneTestPanel.actionPerformed` a také `FileTestPanel.actionPerformed`. Chceme reagovat na změnu výběru v instanci třídy `JComboBox`, která je atributem příslušné třídy se jménem „baseChooseAlg“. Stačí tedy, v případě, že zdrojem akce je „baseChooseAlg“, připsat jeden `if`-blok, který změní popis pole volby parametru pro algoritmus podle potřeby. Někdy totiž uživatel volí počet iterací testu, jindy zase limit, po jehož dosažení má test skončit. Tuto změnu musíme provést v obou třídách `OneTestPanel` i `FileTestPanel`.

- Po spuštění testu jsou informace z grafického prostředí předány do tříd zajišťujících běh vlákna výpočtu. Proto je nutné postarat se, aby i tyto třídy uměly spolupracovat s novým hledačem bází. Dotyčné třídy jsou rozšířením standartní třídy `SwingWorker`. Jmenují se `OneRMTestWorker` pro test jednoho čísla spuštěný ze záložky „one tester“ a `FileRMTestWorker` pro testování čísel v souboru. V metodě `doInBackground` stačí přidat příslušný `if`-blok, aby vytvořil instanci dané třídy hledače báze. Tato instance má být uloženai do atributu se jménem „bc“. Pokud je počet voleb bází závislý na nečem jiném než na počtu iterací, je nutné zde zároveň upravit hodnotu atributu „noIterations“. Obdobný postup je potřeba provést také v třídě `FileRMTestWorker`.

Po splnění všech bodů bode možné zvolit nový algoritmus ve výběru v grafickém prostředí a fungovat správně v testech.

### 4.8.2 Jak přidat nového reportéra

Nejprve je potřeba vytvořit třídu implementující rozhraní `Reporter`. Funkce jednotlivých metod je popsána v oddílu 4.5. V případě záložky „One test“ se používá kombinace reportérů, z nichž poslední vypisuje zprávy do textové oblasti. Pokud by však přece jenom bylo nutné změnit tuto část programu, jsou příslušné příkazy kódu v konstruktoru třídy `OneRMTestWorker` nebo taky ve vnitřní třídě `OneTestPanel.PublishRep`.

V případě testů souborů je možné připsat další typ reportérů do volby kam zapisovat výsledky. Tato změna se provede v `FileTestPanel.actionPerformed`, v bloku týkajícího se stisku tlačítka „startButton“. V této části se vytváří reportér pro instanci třídy `FileRMTestWorker` a do konstruktoru je potřeba předat patřičného reportéra podle volby uživatele. Také bude potřeba pozměnit příslušný atribut typu `JComboBox` nazvaný „outputType“ a případné změny v metodě `FileTestPanel.actionPerformed` související s volbou „outputType“.

### 4.8.3 Jak přidat nový typ zprávy

Třída `ReportType` je výčtovým typem, stačí tedy připsat nový druh zprávy do definice této třídy. Aby program správně fungoval za každé situace, bude také nutné upravit chování `HeadReporteru`. V jeho metodě `report` je nutné připsat část předepisující typ zprávy před její text a odeslání nové zprávy dál. Kromě toho bude potřeba upravit v grafickém prostředí vytváření `SievedReporteru`. K tomuto účelu slouží metoda `GUIfram.showReportSIeveDialog`, která zobrazí uživateli dialog pro označení vypisovaných zpráv a vrátí jeho volbu. V této metodě stačí přidat nový typ. Poslední nutnou úpravou pro zavedení nového typu je nastavení startovní hodnoty této volby v konstruktoru tříd `OneTestPanel` a `FileTestPanel`.

### 4.8.4 Nový prvek v algebraických výrazech určujících bázi

Nejprve si vytvoříme novou třídu implementující rozhraní `BaseExpressionNode`, jejíž funkce budou odpovídat našim požadavkům. Tato třída může být buď součást podbalíku balíku `baseChoosers` nazvaného `baseExpression`, nebo může být přímo vnitřní třídou hledače bází nazvaného `BCFromExpr`. Výhoda první varianty je v možnosti použití této třídy na více místech, druhá varianta zase umožňuje pohodlně používat atributy daného hledače bází. Nyní musíme upravit způsob převodu uživatelem zadaných výrazů na vnitřní stromovou reprezentaci. To se děje ve dvou krocích.

První krok provádí metoda `ExpressionCruncher.evaluateBaseExpression`. V ní se z předpisu vytvoří strom z instancí `BaseExpressionNodeDummy`. Musíme tedy upravit tuto metodu, aby znala naši novou funkci. Mělo by stačit připsat `if`-blok vedle `if`-bloků zpracovávajících znaky „p“ a „n“. Je přitom nutné, aby atribut vytvořené istance `BaseExpressionNodeDummy.num` zůstal nastaven na hodnotu `null`, pokud nejde o číselnou konstantu.

Druhým krokem je úprava metody `BCFromExpr.buildRealTree`, která podle šablony stromu z tříd `BaseExpressionNodeDummy` postaví funkční strom z instancí třídy `BaseExpressionNode`. Zde stačí opět připsat další `if`-blok analogický `if`-blokům ostatních operátorů.

## 4.9 Možnosti dalších vylepšení

Asi nejvhodnějším způsobem, jak vylepšovat tento program, je rozšiřování možností jazyku pro zadávání hledače bází přímo uživatelem. Při zadávání přípustných bází algebraickým výrazem by uživatel mohl mít k dispozici také znaky za dělitele čísla $n - 1$ či náhodná čísla. Předpisy pro báze by také mohlo jít načítat ze souboru.

Třída implementující databázi prvočísel by také stála za vylepšení. Bylo by možné zvětšit rozsah, například pomocí jednoduchého prvočíselného testu, o kterém by bylo dokázáno, že funguje bez chyby do daného limitu. Současná implementace je velmi naivní.

Modul pro generování množin silných pseudoprvočísel by mohl mít i možnost načítat čísla ze souboru. Taky by stálo za to zamyslet se nad nějakým jiným

způsobem hledání silných pseudoprvočísel než tupé prosévání všech složených čísel.

Za zvážení by jistě stálo taky použít jiný algoritmus pro převádění aritmetických výrazů na čísla. V současném stavu je použit kvadratický algoritmus vzhledem k délce výrazu, zatímco je možné získat čas asymptoticky se blížící k $n \log n$. Vzhledem k obvyklé délce převáděných výrazů ale není asymptotická časová náročnost rozhodující, pokud by se ale toto změnilo, bylo by toto vylepšení užitečné.

## 4.10 Prvočíselné testy některých programů

### 4.10.1 Java

V standartním balíku `java.math` je pro datový typ `BigInteger` možné použít metodu `isProbablyPrime`. Tato metoda má volitelnou jistotu a používá kombinaci Rabinova-Millerova testu a Lucasova-Lehmerova testu, přičemž Rabinův-Millerův test používá nahodnou volbu báze. Více informací se lze dočíst v [10].

### 4.10.2 Wolfram Mathematica

Pro rychlé testování prvočíselnosti je v jazyce systému Mathematica určen příkaz `PrimeQ`. Podle [11] tento příkaz nejprve číslo testuje pomocí Rabinova-Millerova testu s bázemi 2 a 3. Pokud ani jeden z těchto testů neprokáže složenost, je testované číslo podrobeno Lukasovu testu. Bylo prokázáno, že neexistuje žádné složené číslo vyhovující této kombinaci testů menší než $10^{16}$. Další možností je nechat systém Mathematica dokázat prvočíselnost testovaného čísla pomocí Prattova certifikátu prvočíselnosti nebo pomocí Atkinova-Morainova certifikátu prvočíselnosti. Tato možnost je výpočetně mnohem náročnější, ale dává spolehlivé výsledky.

### 4.10.3 MATLAB

Funkce `isprime` implementuje Rabin-Millerův test s volbou deseti nezávislých náhodných bází, zdroj [12].

# Závěr

Výsledky testů s námi navrhnutými volbami báze jsou srovnatelné s výsledky testů používajících tradiční metody volby bází. Některé testy se ukázaly nepoužitelnými, neboť byla nalezena složená čísla, na kterých test selže při volbě všech možných bází. Báze volené ve tvaru $1 + (\text{dělitel } n-1)$ se jeví mírně horší, v porovnání s standartně volenými bázemi. Nebyl nicméně zaznamenán případ, že by tento postup selhal. Překvapivé jsou poměrně dobré výsledky jednoduchých voleb bází, jako báze blízko $\frac{n}{3}$, nebo báza jako prvočísla nedělící $n-1$. Celkově jsou však výsledky, kromě neúspěšných testů, velmi podobné. Změna volby báze ze standartních na námi popsané by tedy nejspíš nevedla k lepším výsledkům testu.

# Literatura


[1] *Koblitz, N.:* **A Course in Number Theory and Cryptography**
Springer-Verlang New Yourk,Inc. 1994, 1987, ISBN 0-387-94293-9

[2] *Drápal, A.:* **Teorie čísel a RSA**
`http://www.karlin.mff.cuni.cz/~drapal/teorie_cisel.pdf`

[3] *Ribenboim, P.:* **The Little Book of Big Primes**
R.R. Donnelley & Dons, Inc. Harrisonburg, VA. 1991, ISBN 0-387-97508-X

[4] *Arnault, F.:* **RABIN-MILLER PRIMALITY TEST: COMPOSITE NUMBERS WHICH PASS IT**
MATHEMATICS OF COMPUTATION VOLUME 64, NUMBER 209 JANUARY 1995, PAGES 355-361

[5] *Ireland, L., Rosen, M.:* **A Classical Introduction to ModernNumber Theory**
Springer Science+Bussiness Media,Inc. 1972, 1982, 1990, ISBN 0-387-97329-X

[6] *Oystein, O.***Number Theory and Its History**
McGraw-Hill Book Company, Inc. 1048, 1976, ISBN 0-486-65620-9

[7] *Jaeschke, G.:* **ON STRONG PSEUDOPRIMES TO SEVERAL BASES**
MATHEMATICS OF COMPUTATION, VOLUME 61, NUMBER 204 OCTOBER 1993, PAGES 915- 926

[8] *Baker, M.:* **Korselt's criterion for Carmichael numbers**
`http://people.math.gatech.edu/~mbaker/pdf/korselt.pdf`

[9] *Solovay, R. a Strassen, V. :* **A Fast Monte-Carlo Test for Primality**
SIAM J. Comput. 6, pp. 84-85 (2 pages)

[10] *grepcode.com*
`http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/6-b14/java/math/BigInteger.java#BigInteger`

[11] *Wolfram Mathematica Documentation Center*
`http://reference.wolfram.com/mathematica/tutorial/SomeNotesOnInternalImplementation.html#6849`

[12] *MathWorks Product Documentation*
`http://www.mathworks.com/help/toolbox/mupad/stdlib/isprime.html`

# Seznam tabulek

# Seznam obrázků