Pilpach 4.A

# Seminární práce

## Algoritmus pro hledání prvočísel

Program se skládá ze souboru se samotným algoritmem, souboru pro otestování jeho funkčnosti a souboru pro testování rychlosti algoritmu pro odlišná vstupní data. Seznamy se vstupními daty jsem uložil do samostatných souborů pro větší přehlednost. Celý program je dostupný na Githubu: <https://github.com/Pilpa12/Seminarni-prace---Algoritmus.git>

### Průběh algoritmu

Algoritmus dostane jako vstup seznam čísel, ve kterých bude hledat prvočísla. Zkontroluje každé číslo (označíme jej x) v seznamu, pokud je číslo x rovno 2 nebo 3, přidá se do seznamu prvočísel rovnou. Pokud je x větší než 3, provádí se následující operace:

Do proměnné *maxdivisor* se uloží druhá odmocnina z čísla x zaokrouhlená dolů (tzn. 3.65 ≐ 3). Tím získáme množinu čísel, kterými budeme zkoušet dělit číslo x (využíváme toho, že pokud číslo x není prvočíslo, jeden z jeho dělitelů musí být menší (a druhý dělitel naopak větší) než právě druhá odmocnina z čísla x). Pak číslo x dělíme každým číslem v intervalu <2, maxdivisor+1). Pokud v tomto intervalu najdeme číslo, které dělí číslo x beze zbytku, číslo x není prvočíslo a přejdeme rovnou na další číslo v seznamu. V opačném případě přidáme číslo x do seznamu prvočísel. Výsledkem je seznam pouze s prvočísly z původního seznamu čísel.

### Testování funkčnosti algoritmu

Algoritmus otestujeme pomocí 3 seznamů čísel od 1 do 200 (seřazený, obrácený a zpřeházený). Pokaždé bychom měli dostat stejný výsledek – seznam se stejnými prvočísly.

Algoritmus nám vytvoří seznam prvočísel (*primes*). Pro kontrolu použijeme samostatný soubor primes-to-200.csv s prvočísly od 2 do 200. Zkontrolujeme, zda každé číslo v seznamu *primes* najdeme i v souboru, pokud ano, algoritmus funguje správně. Pokud nějaké číslo ale v souboru nenajdeme, znamená to, že algoritmus udělal chybu a přidal do seznamu *primes* číslo, které prvočíslem není.

### Testování časové složitosti algoritmu

Vyhodnotil jsem, že nejlepší časové složitosti lze dosáhnout se vstupními daty, kterými jsou pouze sudá čísla. Naopak nejhorší časové složitosti dosáhneme se seznamem obsahující pouze prvočísla (čím vyšší prvočísla, tím horší časová složitost). Průměrnou časovou složitost zjistíme se seznamem náhodně zvolených čísel.

Pro každou z těchto variant jsem vytvořil 10 seznamů. Každý obsahuje 2000 čísel, a to v intervalech po 100 milionech (1. seznam od 1 do 100 mil., 2. seznam od 100 mil. do 200. mil., atd.).

Pro test nejhorší časové složitosti jsem tedy vytvořil 10 seznamů obsahující prvních 2000 prvočísel v každém intervalu. Z těchto prvočísel jsem vytvořil i seznamy pro testování nejlepší časové složitosti, ale každé prvočíslo jsem zvětšil o 1, aby čísla byla sudá.

Otestoval jsem, jak dlouho v každém intervalu trvá najít prvních 2000 prvočísel, a ukázalo se, že za takovou dobu algoritmus projde přibližně 40-50 tisíc čísel. Abych zajistil, že čísla ve všech třech případech testování budou přibližně stejně velká, použil jsem pro průměrnou časovou složitost náhodně zvolená čísla v následujících intervalech: čísla v prvním seznamu jsou od 1 do 50000, v druhém seznamu od 100\_000\_000 do 100\_050\_000, ve třetím seznamu od 200\_000\_000 do 200\_250\_000, atd.

Na grafu jsou tři křivky znázorňující čas potřebný k výběru 2000 prvočísel v různých intervalech. Oranžová křivka je pro vybírání prvočísel ze seznamu pouze se sudými čisly. Čas při tomto výběru zůstává téměř neměnný, jak pro malá, tak pro velká čísla. Šedou křivkou je znázorněn čas vybírání prvočísel z náhodně zvolených čísel, kde velikost čísel již hraje svou roli. Největší nárůst času je pak vidět na modré křivce (nejhorší časová složitost), kde algoritmus hledá prvočísla pouze v seznamu plném prvočísel. Zde už velmi záleží na tom, s jak velkými čísly bude algoritmus pracovat.