Zápočtový program – Generátor velkých prvočísel

Jan Oupický

1. Zadání

Program má za úkol vygenerovat prvočísla zadané délky dle počtu cifer v desítkové soustavě.

Největší možné prvočíslo bude mít 200 cifer v desítkové soustavě.

Uživatel si může zvolit kolik prvočísel chce vygenerovat.

Vygenerovaná prvočísla budou postupně vypsána do souboru „vystup.txt“ na samostatný řádek v desítkové soustavě.

1. Uživatelský manuál

Po spuštění programu bude uživatel otázán kolik cifer mají mít vygenerovaná prvočísla. Stačí zadat celé číslo v rozmezí 1-200 a stisknout Enter.

Dále se program uživatele zeptá, kolik prvočísel má vygenerovat. Jako v minulém kroku, stačí zadat jakékoliv kladné celé číslo. Po stisknutí Enter program začne generovat prvočísla.

O skončení procesu program uživatele informuje a v souboru „vystup.txt“ budou vygenerovaná prvočísla.

1. Jak program funguje (algoritmus)

V dnešní době se pro generování velkých prvočísel používaných v kryptografii obvykle používá tento algoritmus:

1. Vygeneruj náhodné číslo *n* zadané délky *d*, které není sudé (dělitelné 2) a nekončí 5 (dělitelné 5).
2. Vygenerované číslo otestuj zda *n* je prvočíslo.

Intuitivně se může nejprve zdát, že u velkých čísel bude takové náhodné generování velice neefektivní. Naštěstí prvočísla nejsou tak vzácná. Dle prvočíselné věty pravděpodobnost, že dané číslo *x* je prvočíslo, je zhruba . Tedy například pokud *x* je číslo s 100 ciframi – 10^99 = *x*. Pravděpodobnost, že *x* je prvočíslo bude . Tedy jedno z čísel bude pravděpodobně prvočíslo. A pokud odebereme sudá tak to vyjde na 114 čísel, což není tak špatné.

Nejhlavnější algoritmus je ten, který testuje zda je *n* prvočíslo. Nejpoužívanější je tzv. Millerův-Rabinův test prvočíselnosti. Tento algoritmus je pravděpodobností, tedy pokud rozhodne, že *n* je prvočíslo, není to 100 % pravdivé. Tato pravděpodobnost, ale exponenciálně mizí v závislosti na počtu cyklů, který algoritmus provede. Počet cyklů budeme značit přirozené číslo *t*. Pravděpodobnost, že *n* bylo špatně označeno za prvočíslo je . V praxi, kde je zcela nutné, aby test byl co nejpřesnější, se počítá obvykle s *t=40* až *50*. Větší přesnost není potřeba, jelikož tato pravděpodobnost, je tak malá, že selhání hardwaru při výpočtu je více pravděpodobné. Při mém testování jsem ale po stovkách vygenerovaných prvočísel nenarazil na chybu, a to jsem použil *t=5*, tedy pravděpodobnost chyby je . Tedy 1 z cca 1024 prvočísel bude chybně vygenerovaných.

Algoritmus funguje následovně:

* Vstup:
  + *n* – prvočíslo, které chceme otestovat
  + *t* – parametr spolehlivosti
* Výstup:
  + *n* je pravděpodobně prvočíslo / *n* není prvočíslo

1. Rozepiš číslo (*n-1*) jako , kde *r* je liché
2. Proveď cyklus níže *t* krát
   1. Vygeneruj náhodné číslo *a* v rozmezí
   2. Spočítej
   3. Jestliže a zároveň
      1. Dokud platí a zároveň proveď
         1. Jestliže
            1. vrať „*n* není prvočíslo“
      2. Jestliže
         1. vrať „*n* není prvočíslo“
3. vrať „*n* je pravděpodobně prvočíslo

Naštěstí ale v praxi velikost parametru *t* nezpomaluje tolik hledání prvočísla, jelikož cyklus obvykle skončí hned při první průběhu a proběhne *t* krát pouze v případě, že *n* je prvočíslo (pravděpodobně).

Nejvíce komplikovaná i časově náročná část algoritmu je 2.2. Tomuto kroku se říká modulární umocňování a existuje pár efektivních algoritmů, jak toto číslo spočítat. Největší problém je, že *r* v exponentu je obvykle podobné velikosti jako *n*. Tedy často umocňujeme *a*, které může mít až *d* cifer, na *r*-tou, které může mít také *d* cifer.

Zvolil jsem algoritmus Montgomeryho umocňování. Tento algoritmus není tak jednoduchý na implementaci jako ostatní algoritmy, jelikož je potřeba implementovat speciální algoritmus pro násobení tzv. Montgomeryho násobení. Dle mých testů, byl tento algoritmus řádově rychlejší než ostatní.

Při následujícím popisu algoritmů budu používat zápis čísel v následující formě. Nechť *x* je číslo s *n* ciframi v desítkové soustavě (všude se předpokládá, že čísla jsou v desítkové soustavě, na výjimku upozorním) , kde a *i* značí řád. Tedy například číslo *x =* 10548, *n* = 5 by bylo zapsáno jako

Dále přiblížím, co je tzv. Montgomeryho redukce čísla *T* modulo *m* vzhledem k *R*. Nechť *R, T* jsou přirozená čísla, taková, že *R > m* (modulo, ve kterém počítáme) a NSD(m,R) = 1 (Největší Společný Dělitel). Dále pro *T* platí, že . Poté se nazývá Montgomeryho redukce čísla *T* modulo *m* vzhledem k *R*. Tato forma zapsaní čísla se používá v algoritmech níže.

Algoritmus pro Montgomeryho násobení:

* Vstup:
  + *m* (modulus) ve zvolené číslicové soustavě
  + *x (první činitel)*
  + *y (druhý činitel)*
    - kde
  + *R* parametr pro Montgomeryho redukci výsledku násobení. Obvykle, aby R splnilo požadavky výše, se , kde *b* je číselná soustava (v našem případě *b=10* a *n* je počet cifer *m*.
  + , tedy záporné inverzní číslo k *m* modulo *b*. Tedy kladné číslo od 0 do 9, pokud *b=10*. Toto číslo se vypočítá pomocí rozšířeného Euklidova algoritmu (popsán níže)