Ikerprímkeresés szitálással

Implementáció Haskellben

Dr. Érdi Gergő
http://gergo.erdi.hu/

1. Eratoszthenészi szita

A becélzott generátor-intervallum szitálásához szükségünk lesz kis prímszámokra. Ezeket egyszerűen előállíthatjuk az eratoszthenészi szita néven ismert algoritmus naív implementációjával[1, 2]:

$$
\begin{aligned}
&\mathit{primes} :: [\mathbb{Z}] \\
&\mathit{primes} = \mathit{sieve}\ [2\,..] \\
&\quad \textbf{where}\ \mathit{sieve}\ (p : ns) = p : \mathit{sieve}\ [n \mid n \leftarrow ns, p \not| n]
\end{aligned}
$$

2. Intervallumok szitálása

Az ikerprímszámokat valamilyen intervallum prímszám elemei segítségével generáljuk. A *sieveInterval* függvény előállítja a 2^k és 2^{k+n} közötti számok közül azokat, amelyeket a „kis” prímek nem szitálnak ki.

$$
\begin{aligned}
&\mathit{sieveInterval} :: \mathbb{Z} \to \mathbb{Z} \to [\mathbb{Z}] \\
&\mathit{sieveInterval}\ k\ n = \mathit{filter}\ (\neg \circ \mathit{hasSmallPrimeDiv})\ [a\,..\,b] \\
&\quad \textbf{where}\ a = 2^k \\
&\qquad\qquad b = 2^{k+n} \\
&\qquad\qquad \mathit{hasSmallPrimeDiv}\ x = \mathit{any}\ (\lambda p \to p \not\equiv x \wedge p|x)\ \mathit{smallprimes} \\
&\qquad\qquad\quad \textbf{where}\ \mathit{smallprimes} = \mathit{take}\ 100\ \mathit{primes}
\end{aligned}
$$

3. A Miller-Rabin prímteszt

Az előbbi értelemben átszitált intervallum generátor-kandidátusait, és a generált ikerprím-kandidátusokat a Miller-Rabin prímteszt segítségével vizsgáljuk[3].

Ehhez szükségünk lesz a pszeudoprímség-vizsgálatra:

$$
\begin{aligned}
&\mathit{isPseudoPrime}\ n\ a = \mathit{isPseudoPrime}'\ n\ (s, d)\ a \\
&\quad \textbf{where}\ (s, d) = \mathit{split}_2\ (n - 1) \\
&\mathit{split}_2\ n \mid 2 \not| n \qquad\quad = \quad (0, n) \\
&\qquad\qquad \mid \mathit{otherwise} = \quad \textbf{let}\ (s, d) = \mathit{split}_2\ (n \div 2) \\
&\qquad\qquad\qquad\qquad\qquad\quad \textbf{in}\ \ (s + 1, d) \\
&\mathit{isPseudoPrime}'\ n\ (s, d)\ a \mid a \geqslant n \qquad = \textbf{error}\ \texttt{"a >= n"} \\
&\qquad\qquad\qquad\qquad\qquad\quad \mid \mathit{otherwise} = \mathit{repeatSquare}\ (s - 1)\ x \\
&\quad \textbf{where}\ x = [\![a^d]\!]_n \\
&\qquad\qquad \mathit{repeatSquare}\ r\ x \mid x \equiv 1 \vee x \equiv n - 1 = \textsc{True} \\
&\qquad\qquad\qquad\qquad\qquad\quad \mid r > 0 \qquad\qquad\quad = \textbf{let}\ x' = [\![x^2]\!]_n \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad\ \ \textbf{in}\ \ \mathit{repeatSquare}\ (r - 1)\ x' \\
&\qquad\qquad\qquad\qquad\qquad\quad \mid \mathit{otherwise} \qquad\qquad = \textsc{False}
\end{aligned}
$$

Az algoritmus egy adott $n \in \mathbb{Z}$ számról megmondja, ha a szám egy vagy páros (ekkor nyilvánvalóan nem prím), vagy talál egy $a \in \mathbb{Z}$ számot, amihez képest n nem pszeudoprím (n ekkor sem lehet prím), vagy „valószínűleg prím” eredménnyel áll le:

```
data Primality = One | Even | Witness ℤ | Prime
  deriving (Show, Eq)
```

A tényleges Miller-Rabin prímteszt implementálásakor feltesszük az általánosított Riemann–sejtést, és ezt kihasználva n prímségének vizsgálatakor csak a $2, \ldots, \lfloor 2 \ln^2 n \rfloor$ halmazon keresünk tanút[4], illetve ugyanezen elv alapján készítjük el a teszt determinisztikus változatát.

```
millrab :: ℤ → ℤ → Primality
millrab 1 k              = One
millrab 2 k              = Prime
millrab 3 k              = Prime
millrab n k | 2|n        = Even
            | otherwise  = combine [mr a | a ← [2 .. k']]
  where k' = minimum [k, n − 1, ⌊2 ∗ (ln n)^2⌋]
        (s, d) = split_2 (n − 1)

        mr a = if isPseudoPrime' n (s, d) a then Prime else Witness a

        combine (Prime : rs) = combine rs
        combine []           = Prime
        combine (r : rs)     = r

millrabDet n = millrab n n
isMillRabPrime p = case millrabDet p of
                     Prime → True
                     ⊥     → False
```

4. Prímkandidátusok generálása

A kiszitált intervallumból, a Miller-Rabin teszten is átment generátor-értékekkel az alábbi polinom segítségével készítünk ikerprím-kandidátusokat (az m paraméterrel a nagyságrendet lőjük be):

```
type 𝒢 = ℤ → ℤ
f_1 :: ℤ → 𝒢
f_1 m x = (h_0 + c ∗ x) ∗ 2^m − 1
  where h_0 = 5775
        c   = product (take 6 primes)
        e   = 3299
```

Végre tehát minden adott, hogy prím-párokat keressünk $f_1(2^k)$ és $f_2(2^{k+n})$ között:

```
findGenerators :: ℤ → ℤ → [ℤ]
findGenerators k n = filter isMillRabPrime (sieveInterval k n)

findCandidates :: (𝒢, 𝒢) → ℤ → ℤ → [(ℤ, ℤ)]
findCandidates (f_1, f_2) k n = let gs = findGenerators k n
                                in  map (λg → (f_1 g, f_2 g)) gs
```

5. Ikerprím-szitálás

A fentiek alapján végre elkészíthetjük a prímpár-szitánkat. Mivel a determinisztikus Miller-Rabin teszt $O(\log^4 n)$ lépésszámú, érdemes először elvégezni a kandidátus-pár mindkét tagján a valószínűségi Miller-Rabin tesztet, és csak ha mindkét szám átment ezen, akkor vizsgálni a determinisztikus módszerrel.

$$
\begin{aligned}
&\mathit{both} :: (a \to \mathbb{L}) \to (a, a) \to \mathbb{L} \\
&\mathit{both}\ p\ (x, y) = p\ x \wedge p\ y \\
\\
&\mathit{findPrimePairsProb} :: (\mathcal{G}, \mathcal{G}) \to \mathbb{Z} \to \mathbb{Z} \to [(\mathbb{Z}, \mathbb{Z})] \\
&\mathit{findPrimePairsProb}\ (f_1, f_2)\ k\ n = \mathbf{let}\ \mathit{ns} = \mathit{findCandidates}\ (f_1, f_2)\ k\ n \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathbf{in}\ \ \mathit{filter}\ (\mathit{both}\ \mathit{isMillRabPrimeProb})\ \mathit{ns} \\
&\quad \mathbf{where}\ \mathit{isMillRabPrimeProb}\ p = \mathit{millrab}\ p\ 2 \equiv \textsc{Prime} \\
\\
&\mathit{findPrimePairs} :: (\mathcal{G}, \mathcal{G}) \to \mathbb{Z} \to \mathbb{Z} \to [(\mathbb{Z}, \mathbb{Z})] \\
&\mathit{findPrimePairs}\ (f_1, f_2)\ k\ n = \mathit{filter}\ (\mathit{both}\ \mathit{isMillRabPrime})\ (\mathit{findPrimePairsProb}\ (f_1, f_2)\ k\ n)
\end{aligned}
$$

Ezekután az ikerprím-szitálás egy egyszerű speciális esete a prímpár-keresésnek, amikor a pár második tagja pontosan kettővel nagyobb az elsőnél.

$$
\begin{aligned}
&\mathit{findTwinPrimes}\ f_1 = \mathit{findPrimePairs}\ (f_1, f_2) \\
&\quad \mathbf{where}\ f_2\ x = f_1\ x + 2
\end{aligned}
$$

A. Eredmények

Az első próbálkozások lehangoló eredményeket hoztak, amennyiben már tíz-tizenkét decimális számjegyű számok esetén is a memóriaigény annyira rohamosan kezdett nőni, hogy elfoglalta az összes, rendelkezésre álló tárhelyet, és a programot az operációs rendszer lelőtte. Hamar kiderült, hogy ennek oka a $[\![x^y]\!]_n$ függvény naív implementációja. Ehelyett ezért az alábbi, a modulus-képzést lépésenként elvégző implementációt használtuk:

$$
\begin{aligned}
&[\![\cdot^{\cdot}]\!]_{\cdot} :: \mathbb{Z} \to \mathbb{Z} \to \mathbb{Z} \to \mathbb{Z} \\
&[\![x^1]\!]_n \qquad\quad = x \mod n \\
&[\![x^y]\!]_n \mid 2|y \ = ([\![x^{y \div 2}]\!]_n \uparrow 2) \mod n \\
&\qquad\ \ \mid 2 \not| y = ([\![x^{y-1}]\!]_n * x) \mod n
\end{aligned}
$$

Végezetül a fenti h_0 és c választással, a nagyságrendet m változtatásával beállítva kerestünk ikerprímeket. A futtatás során mértük az első, adott m paraméterű ikerprím-pár megtalálásához, és az adott pár első tagjának determinisztikus prímtesztéléséhez szükséges időt. Az eredményeket az 1. és a 2. ábra mutatja összevetve az elméletileg várt $\Theta(\log^4 p)$-es futással.

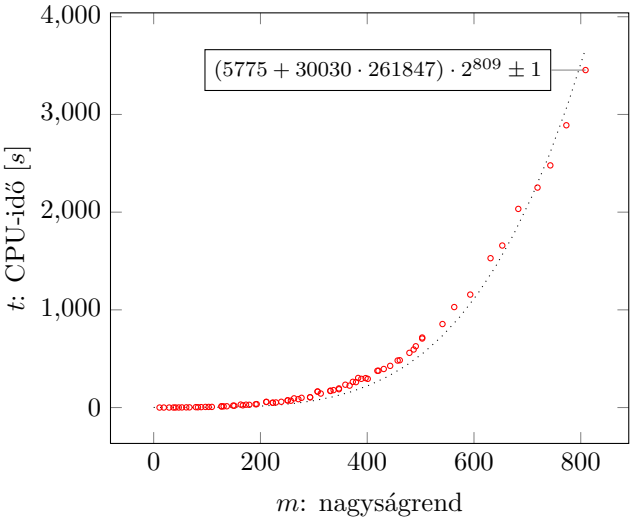
A tesztelés során talált legnagyobb ikerprím-pár az $(5775 + 30030 \cdot 261847) \cdot 2^{809} \pm 1 \sim 10^{254}$.

B. Továbbfejlesztési lehetőségek

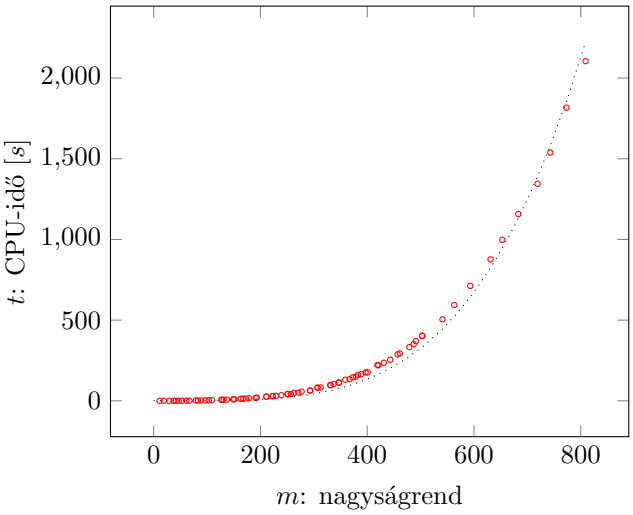
Ez a fajta prímkeresés jól adja magát párhuzamosításhoz. Először is az egyes kandidátusok Miller-Rabin tesztelése egymástól függetlenül elvégezhető. Ez egy triviális párhuzamosítás, amit egy alkalmas Haskell fordító automatikusan el is végezhet[5].

Másodrészt egy tesztelésen belül a különböző alapokhoz tartozó pszeudo-prímség eldöntése is párhuzamosítható. Itt már szükség van egyfajta szinkronizációra is, ugyanis ha bármely alaphoz nem pszeudoprím az adott kandidátus, akkor már felesleges befejezni a többi alappal való tesztelését.

Ezen felül felgyorsíthatjuk a nem ikerprímek kiszitálását, ha p és $p+2$ determinisztikus Miller-Rabin tesztjeit összefésüljük olymódon, hogy először megvizsgáljuk mindkét számra, hogy 2 alappal pszeudoprímek-e, majd mindkettőre, hogy 3 alappal, s.í.t. Ílymódon ha a pár első tagja prím, de a második nem, nem kell végigvárni a teljes Miller-Rabin teszt lefutását az első tagra.



1. ábra. Az első, adott nagyságrendű ikerprím-pár megtalálási ideje



2. ábra. A determinisztikus Miller-Rabin prímteszt futásideje

Hivatkozások

[1] Melissa E. O'Neill. The Genuine Sieve of Eratosthenes. *Journal of Functional Programming*, 19(01):95–106, 2009.

[2] Lambert Meertens. FUNCTIONAL PEARL Calculating the Sieve of Eratosthenes. *Journal of Functional Programming*, 14(06):759–763, 2004.

[3] Michael O. Rabin. Probabilistic algorithm for testing primality. *J. Number Theory*, 12(1):128–138, 1980.

[4] Gary L. Miller. Riemann's Hypothesis and tests for primality. In *STOC '75: Proceedings of seventh annual ACM symposium on Theory of computing*, pages 234–239, New York, NY, USA, 1975. ACM.

[5] Philip W. Trinder, Kevin Hammond, James S. Mattson Jr., Andrew S. Partridge, and Simon L. Peyton Jones. GUM: a portable implementation of Haskell. In *Proceedings of Programming Language Design and Implementation*, Philadephia, USA, May 1996.