Eötvös Loránd Tudományegyetem

Informatikai Kar

Komputeralgebra Tanszék

Prímszita algoritmusok összehasonlítása

Vatai Emil
Adjunktus

Nagy Péter
Programtervező Informatikus BSc

Budapest, 2018

Tartalomjegyzék

1. fejezet

Bevezetés

Sziták összehasonlítása azonos környezetben. Hatékonyság összehasonlítása az elmélet szerint várható értékekkel. Ellenőrzésként a prímek néhány statisztikájának összevetése az elméleti értékekkel, és az ismert eredményekkel. Hatékonyság és implementálhatóság.

2. fejezet

Felhasználói dokumentáció

A megoldott feladat

A program lehetővé teszi különböző sziták futási idejének ábrázolását grafikonon, és összevetését elméleti becsült értékekkel. A programmal egyéb forrásból származó minták megjelenítése is lehetséges.

A prímszámok statisztikáinak előállításához a programhoz tartozik egy optimalizát szita-implementáció, amivel 2^{64}-ig lehet szitálni, és az eredményt rögzíteni. Az elmentett eredményeket a program egy külön része összesíti néhány alapstatisztikára.

Különböző szita-algoritmusok futási idejének összehasonlításához a program tartalmazza Atkin szitájának egy implementációját, és Eratosztenész szitájának néhány variációját.

Felhasznált módszerek

A program két rész fő részből áll. A prímszámok generátora C-99 nyelven készült el. A program csak parancssorból futtatható, és a sztenderd könvytári függvényekből is csak néhányat használ a memóriájában előállított eredmények fájlba írásához. A C nyelv választását a hatékony végrehajtás és memórafelhasználás, és a hordozhatóság indokolja, a generátor elkülönítését az automatizálhatóság indokolja.

A prímgenerátor Eratosztenész szitájának szegmentált változata, a szitatábla egy 2^{30} hosszú darabját szitálja ki minden iterációban. A generátor önmaga is két részből áll. Egy egyszerűbb implementáció a prímek listáját állítja elő 2^{32}-ig.

A gyorsabb, de összetettebb szita 2^{32}-től $2^{64} - 2^{33}$-ig szitál, és a futásához szükséges a prímek listája 2^{32}-ig. A szitatáblát a memóriában a gyorsítótár hatékonyabb a kihasználásához kisebb résszegemensekre osztja, és a prímeket a nagyságuk és a következő szitálási pozíciójuk szerint csoportosítja. Továbbá nagyobb prímeknek

csak 30-cal relatív prím többszöröseit veszi figyelembe.

A program többi része a Java környezethez készült, a képzésben betöltött szerepe és a hordozhatósága miatt. A minták és függvények ábrázolásához használt Swing grafikus könyvtár a Java környezet része, egyszerű, hordozható, támogatja az élsimítást, és a sebessége elfogadható a feladatra. A Java sztenderd konténerosztájai habár kényelmesen használhatóak, és változatosak, de sebesség és memóriakihasználtsági problémák miatt több helyen is alacsonyabb szintű megoldást kellett választani.

A program a mintákat alapfüggvények lineáris kombinációjával képes közelíteni, a legkisebb négyzetek módszerével. A minták elemeinek nagy száma, és a mintaértékek nagy terjedelme miatt szükséges volt a numerikus algoritmusok nagyobb pontosságú implementációja, ami a számítási idő növekedésével jár.

A program hosszabb távon egyedül a prímek szegmensenkénti statisztikáit tárolja, amit csak sorban dolgoz fel, és a statisztikák összmérete sem indokolna egy egyszerű bináris fájlnál bonyolultabb megoldást. Az adatvesztés elkerüléséhez elég a statiszikafájl csere alapú felülírása.

A Java program tartalmaz több prímszita implementációt is, ezeknek az algoritmikus bonyolultsága különböző, de az összehasonlíthatósághoz a közös részek közös implementációt kaptak, és az eltérő részek hasonló szinten optimalizáltak. A sziták helyességének ellenőrzéséhez a sziták eredménye egymással összehasonlítható. A programhoz elkészült Atkin szitája, és Eratoszthenész szitájának több variációja: az algoritmus legegyszerűbb implementációja a teljes szitatáblán jelöli meg a prímek többszöröseit, a többi a táblát rövidebb szegmensekre osztja fel, és ezeket sorban szitálja. A szegmentálás szükségessé teszi, hogy eldüntsük, hogy mely szegmensben mely prímek szitálnak. A legegyszerűbb megvalósítás minden szegmensben minden prímmel megpróbál szitálni, a próbaosztáshoz hasonlóan. Ennél hatékonyabb a két elsőbbségi sorral megvalósított szita, és a harmadik, edényekbe csoportosító megoldás.

A sziták között kapott helyet a próbaosztás, és az erős pszeudo-prím teszt is, de ezek a lassúságuk miatt csak igen rövid tartományok ellenőrzésére alkalmasak.

A program telepítése és futtatása

A szitatábla-generátor

Mintaadatbázis karbantartása

Szitatáblák ellenőrzése

Minta megjelenítése

Minta közelítése függvényekkel

3. fejezet

Fejlesztői dokumentáció

A program komponensei

A forráskód felosztása

Adatszerkezetek

Numerikus algoritmusok

Egyenletrendszerek. Összeadás. Körül kéne írni, hogy igazán tudjuk, hogy hipotézisvizsgálatra nem vállalkozunk.

Szináták

Eratosztenész szitája, szegmentáltan is. COLS. Prioritásos sorral. Atkin szitája.

Szegmentált szita inicializálása.

Trial division. Pszeudoprím teszt.

Feltételek. Elméleti sebesség.

Prioritásos sorok

1: $q \leftarrow$ Új-sor
2: **for** $i \leftarrow 2, n$ **do**
3: **while** $\exists (p, k) \in q : k \leq i$ **do**
4: $(p, k) \leftarrow$ Sor-Eltávolít-Min(q)
5: Megjelöl(i)
6: Sor-Beszúr(q, (p, k+p))

```
    end while
    if ¬ MEGJELÖLT?(i) then
        SOR-BESZÚR(q, (i, 2i))
    end if
end for
```

Bináris kupac

A mérések grafikonják pixelei alapján lassú. A beszúrásonkénti elméleti $\mathcal{O}(log|q|)$ ideje se biztató.

Bigyó

A bigyó egy természetesszám-párokat tartalmazó monoton prioritásos sor. A számpárok egy prímet, és a prím egy szitálási pozícióját reprezentálják. A sor monoton, minden állapotához tartozik egy érték, a sor aktuális pozíciója, aminél kisebb vagy egyenlő pozíciójú értéket a sor nem tartalmazhat. A bigyó edények egy végtelen sorozatát is tárolja, a sor elemei ezekbe az edényekbe kerülnek. Egy eltárolt elem helyét a sorozatban az elem pozíciójának és a sor aktuális pozíciójának távolsága határozza meg.

A távolságfüggvény legyen

$$d(x, y) := \lfloor log_2(x \oplus y) \rfloor \qquad (x, y \in \mathbb{N}, y > x \geq 0)$$

ahol $\oplus$ a bitenkénti XOR.

$d(x, y)$ a legnagyobb bit-index, ahol x és y eltér.

Ha q egy bigyó, legyen $q.a$ q aktuális pozíciója, és $q.e[i]$ q $i.$ edénye. AZ edények, és a számpárok struktúrája...

Egy q bigyó invariánsa

$$\begin{aligned}
&\forall (p, k) \in q : \\
&\qquad q.a < k \\
&\qquad \forall i \in \mathbb{N}_0 : (p, k) \in q.e[i] \iff i = d(q.a, k) \\
&\forall (p, k) \notin q : \forall i \in \mathbb{N}_0 : (p, k) \notin q.e[i]
\end{aligned}$$

Új, üres sor létrehozása tetszőleges kezdőpozíciótól, és meglévő sorba elem beszúrása...

A sor elemeinek feldolgozása i-ig

```
while q.a < i do
```

```
    j ← d(q.a, q.a + 1)
    q.a ← q.a + 1
    for all (p, k) ∈ q.e[j] do
        Edény-Kivesz(q.e[j], (p, k))
        if k = i then
            Visszaad((p, k))
        else
            Edény-Beszúr(d(q.a, k), (p, k))
        end if
    end for
end while
```

Helyesség

Idő

Hely

Számrendszer

És számrendszer vs. tisztán funkcionálisban nincs bármekkora tömb, csak valami fával közelítve.

Amúgy sincs bármekkora tömb...

És exponenciálisan kell növelni...

Cache

És exponenciálisan kell növelni...

Memória

Összes prím, és pozíciója 2^32-ig. Primitív típusok és boxing. Garbage collector.

Teszt

Adatszerkezetek

Numerikus pontosság és sebesség

Sziták

Elmélet vs. mért

Források