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 $2^{64} - 2^{34}$-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 harminccal 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 Java program futtatásához JRE8-ra van szükség, a fordításához JDK8-ra. A C program fordításához C99 szabványú fordítóprogram kell. A program fejlesztése és tesztelése OpenJDK8-cal és GCC 7.3-mal történt Linux operációs rendszeren.

A tarball a Java programot lefordítva is tartalmazza, kicsomagolás után azonnal futtatható. A C programot a generator könyvtárban található Makefile segítségével lehet lefordítani:

```
generator$ make clean build
```

A Java program a NetBeans 8-as verziójával készült, parancssorból a project ant fájlának segítségével lehet újrafordítani:

```
gui$ ant clean jar
```

A program futtatásához több előkészített script is rendelkezésre áll, ezeket mind a scripts könyvtárból lehet indítani. Az előző két fordítást egyben is el lehet végezni a recompile scripttel:

```
scripts$ ./recompile
```

Prímek keresése

A prímek keresését két program végzi, az init program 2^{32}-ig keresi meg és tárolja el a prímeket, a generator 2^{32}-től $2^{64}-2^{43}$-ig. Az init készítette szitattáblák a generator futásához szükségesek.

Mindkét program a számokat 2^{30} hosszú táblánkként szitálja, minden tábla első száma $2^{30}k+1$ alakú. A szitatáblákat a programok a fájlrendszerben bittérképként tárolják, egy tábla kb. 64Mb.

Az első négymilliárd szám szitálása a következő képpen indítható el:

```
generator$ ./init.bin ../db
szegmens 0 - kezdet              1 - felkészülés 159 906 063 ns - szitálás 1 122 878 403 ns
szegmens 1 - kezdet 1 073 741 825 - felkészülés   2 353 698 ns - szitálás 1 171 101 336 ns
szegmens 2 - kezdet 2 147 483 649 - felkészülés   2 344 515 ns - szitálás 1 188 330 478 ns
szegmens 3 - kezdet 3 221 225 473 - felkészülés   2 391 660 ns - szitálás 1 199 900 603 ns
```

Ennek eredmény 4 bittérképfájl, amit a db mappába ment.

A generator programot több féle képpen is lehet indítani. Mindegyik esetben két paramétert kell megadni az indításhoz, a számot, ahol szitálást kezdje, és a szegmensek számát, amit ebben a futásban végig kell szitálni. A kezdő számot meg lehet adni decimálisan is, vagy hexadecimálisan is, "0x" prefixszel. A szegmensek számát is kétféle képpen lehet szabályozni. Az egyik lehetőség fix számú szegmens megadása, decimálisan. A másik mód a háttértáron fenntartandó szabad hely átadása, decimálisan, vagy "0x" prefixszel hexadecimálisan.

```
generator$ ./generator.bin ../db start 0x100000001 segments 3
szegmens kezdet: 4 294 967 297
szegmensek száma: 3
kis szegmensek mérete: 22
elágazások bitjei: 8
felkészülés 1/4
felkészülés 2/4
felkészülés 3/4
felkészülés 4/4
felkészülés vége
szegmens 4 - kezdet 4 294 967 297 - felkészülés 8 162 031 487 ns - szitálás 729 914 370 ns
szegmens 5 - kezdet 5 368 709 121 - felkészülés           418 ns - szitálás 736 412 117 ns
szegmens 6 - kezdet 6 442 450 945 - felkészülés           385 ns - szitálás 750 022 409 ns
összes szitálás 2 216 348 896 ns

generator$ ./generator.bin ../db start 0x100000001 reserve-space 1000000000
szegmens kezdet: 4 294 967 297
fenntartandó hely: 1 000 000 000 byte
kis szegmensek mérete: 22
elágazások bitjei: 8
```

```
felkészülés 1/4
felkészülés 2/4
felkészülés 3/4
felkészülés 4/4
felkészülés vége
szegmens 4 - kezdet 4 294 967 297 - felkészülés 8 162 031 487 ns - szitálás 729 914 370 ns
szegmens 5 - kezdet 5 368 709 121 - felkészülés           418 ns - szitálás 736 412 117 ns
szegmens 6 - kezdet 6 442 450 945 - felkészülés           385 ns - szitálás 750 022 409 ns
összes szitálás 2 216 348 896 ns
```

Mindkét program kimenetében a start a szegmens kezdőszáma, a segment az indexe, $start = segment \cdot 2^{30} + 1$, az init a szita incializálási ideje a szitálás előtt, és a sieve a szegmens szitálásának ideje. Ezek az információk az elmentett táblafájlokból is kiolvashatóak.

Mintaadatbázis karbantartása

A mintaadatbázis a lementett szitatáblák összesített statisztikáit tárolja. A program az adatbázis könyvtárában kétfajta fájlt vesz figyelembe. A "primes\.[0-9a-f]{16}" reguláris kifejezésnek megfelelő nevű fájlok a szegmensek szitatáblái. A fájlnév második fele a szegmens kezdőszáma hexadecimálisan. A fájl a páratlan számok bittérképe mellett tartalmazza a szegmens szitálásának megkezdésére, és a szitálására fordított időt, valamint ellenőrzésképpen a szegmens kezdőszámát is.

A szitatáblák mellett az adatbázis tartalmazhatja szegmensek összesített statisztikáit is, "aggregates" néven. Ez a fájl az eddig összesített szegmensről a következő információkat tartja nyilván:

- a szegmens kezdőszáma
- a szegmens szitálására való felkészülés idejét
- a szegmens szitálásának idejét
- az összesítésre fordított időt
- a szegmensfájl utolsó módosítási idejét
- a szegmensbe eső legkisebb, és legnagyobb prímet
- a szegmensbe eső prímek számát, $12\mathbb{Z}11$, $4\mathbb{Z}1$, $4\mathbb{Z}3$, $6\mathbb{Z}1$ alakok szerinti bontásban
- a szegmensben előforduló prímhézagok első előfordulásának helyét, és az előfordulások számát.

Az adatbázison három művelet végezhető:

- Le lehet kérni, hogy eddig hány szegmens van összesítve, és melyik a következő hiányzó szegmens. Azt is visszaadja, hogy hány szegmensfájl van épp az adatbázis könyvtárában, és hogy ezek között van-e olyan, ami nincs összesítve.
- Összesíteni lehet az új szegmensfájlok.
- Össze lehet fésülni két összesítőfájlt.

Mindhárom művelet elvégezhető grafikus felületen, és automatizáláshoz parancssorból is. A grafikus felületet a gui scripttel lehet indítani, a parancssorból a database script segítségével lehet elérni a műveleteket.

Ezek a scriptek határozzák meg, hogy a program melyik könyvtárat használja adatbázisként, ezt nem kell külön megadni. Szükség esetén ez a könyvtár a scriptek egyszerű módosításával megváltoztatható.

A szegmensinformációkat a grafikus felületen a "DB info" gomb megnyomásával lehet lekérdezni, parancssorból a "database info" kiadásával.

```
scripts$ ./database info
100.0%
szegmensfájlok: szegmensek száma: 7
szegmensfájlok: első szegmens kezdete: 1
szegmensfájlok: utolsó szegmens kezdete: 6,442,450,945
szegmensfájlok: első hiányzó szegmens kezdete: 7,516,192,769
szegmensfájlok: hiányzó szegmensek száma: 18,446,744,073,709,551,593

összesítések: szegmensek száma: 62,881
összesítések: első szegmens kezdete: 1
összesítések: utolsó szegmens kezdete: 67,516,885,893,121
összesítések: első hiányzó szegmens kezdete: 67,517,959,634,945
összesítések: hiányzó szegmensek száma: 18,446,744,073,709,488,719

új szegmensfájlok: 7

scripts$ ./database info crunch 100
100.0%
szegmensfájlok: szegmensek száma: 7
szegmensfájlok: első szegmens kezdete: 1
szegmensfájlok: utolsó szegmens kezdete: 6,442,450,945
szegmensfájlok: első hiányzó szegmens kezdete: 7,516,192,769
szegmensfájlok: hiányzó szegmensek száma: 18,446,744,073,709,551,593

összesítések: szegmensek száma: 62,881
összesítések: első szegmens kezdete: 1
összesítések: utolsó szegmens kezdete: 67,516,885,893,121
összesítések: első hiányzó szegmens kezdete: 67,517,959,634,945
összesítések: hiányzó szegmensek száma: 18,446,744,073,709,488,719

crunch: ../generator/generator.bin ../db start 0x3d6840000001 segments 0x64
```

A "crunch" utótag és szitálni kívánt szegmensek számának megadásával az utolsó sorban a következő futtatásra ajánlott parancsot is megjeleníti a program. Ezeknek a parancsoknak a követésével a teljes szitálható tartományt fel lehetne dolgozni. A "crunch-numbers" script ezt a feladatot kísérli meg elvégezni.

Az info megjeleníti az új szegmensfájlok számát is, azokat, amikhez vagy nem tartozik összesített adat, vagy tartozik, de az ott tárolt utolsó módosításnál újabb a szegmensfájl. Ezeket a "DB összesítés" gomb megnyomásával, vagy a "database reaggregate" paranccsal lehet összesíteni.

```
scripts$ ./database reaggregate
100.0%
```

Lehetőség van egy másik adatbázisból statisztikák átvételére is. Az összefésülést a grafikus felületen a "DB import" megnyomása után egy file kiválasztásával lehet indítani, parancssorból a "dabatase import aggregates fájlnév" kiadásával. A megadott fájlból azokat a szegmeneseket veszi át a program, amik vagy nincsenek még meg az adatbázisban, vagy megvannak, de az importált statisztikák utolsó módosítási ideje nagyobb az adatbázisban lévőnél.

```
scripts$ ./database import aggregates ../db/aggregates.large
100.0%
```

screenshot

Szitatáblák ellenőrzése

A generator program teszteléséhez a grafikus program képes a szegmensfájlok ellenőrzésére. Ehhez a program újra előállítja az ellenőrzött szegmensfájlokat, és azokat összeveti. A újraelőállításhoz egy egyszerű szegmentált Eratoszthenészi szitát, vagy ez erős pszeudo-prím tesztet lehet választani. A pszeudo-prím teszt rendkívül lassú, a célja, hogy a szitákhoz képest egy alapvetően más módszer is rendelkezésre álljon az ellenőrzéshez.

A grafikus felületen a "Szegmensfájlok ellenőrzése" gomb megnyomása után lehet kiválasztani a szitálni kívánt szegmenseket, és a referencia előállításának módszerét, majd az "Ellenőrzés" gombbal lehet megkezdeni a tényleges folyamatot.

Az ellenőrzést parancssorból is lehet futtatni a check-segments scripttel. A script első paraméterében a referencia módszerét várja, ami lehet "sieve", vagy "test", a paraméterek maradéke az ellenőrizendő szegmensek kezdőszáma kell legyen. Ha egy kezdet sincs megadva, akkor a program az adatbázis könyvtárban található összes szegmensfájlt ellenőrzi.

```
scripts$ ./check-segments sieve
100.0%: A szegmensek helyesek.

scripts$ ./check-segments sieve 0x1 0xc0000001
100.0%: A szegmensek helyesek.

scripts$ ./check-segments test 0x1
100.0%: A szegmensek helyesek.
```

screenshot

Minta megjelenítése

Minta közelítése függvényekkel

Minták generálása scriptekkel

Sziták ellenőrzése scriptekkel

3. fejezet

Fejlesztői dokumentáció

A feladat leírása

A program komponensei

A forráskód felosztása

Adatszerkezetek

Numerikus algoritmusok

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

Sziták

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))
 7:     end while
 8:     if ¬ Megjelölt?(i) then
 9:         Sor-Beszúr(q, (i, 2i))
10:     end if
11: end for
```

Bináris kupac

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

Bigyó

A bigyó egy monoton prioritásos sor. 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

```
1: while q.a < i do
2:     j ← d(q.a, q.a + 1)
3:     q.a ← q.a + 1
4:     for all (p, k) ∈ q.e[j] do
5:         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