# Lusta szita 2

2018.9.2.

## Contents



## 1 Mertens tétele

Mertens első és második tételének következménye, hogy

$$\sum_{\substack{p \le n \\ p \text{ prím}}} \frac{\ln p}{p} \in \Theta(\log n)$$
$$\sum_{\substack{p \le n \\ p \text{ prím}}} \frac{1}{p} \in \Theta(\log \log n)$$

## 2 Ismételt összeadás

A számok legyenek olyanok, hogy ezek a műveletek konstans idejűek

$$\begin{aligned} n &= 0 \\ n/2 \mapsto & \left\lfloor \frac{n}{2} \right\rfloor \\ n\%2 \mapsto & \; n - 2\left\lfloor \frac{n}{2} \right\rfloor \\ & 2n + 0 \\ & 2n + 1 \end{aligned}$$

Ez láncolt listákkal könnyen teljesíthető. Az összeadást végezzük el így

$$
\begin{aligned}
b(x,y,z) &:= (x\%2) + (y\%2) + (z\%2) \\
r(x,e) &:= x, && \text{ha } (e/2) = 0 \\
&:= r(2x + (e\%2), e/2), && \text{különben} \\
o(x,y,c,e) &:= r(y,e), && \text{ha } c = 0 \wedge x = 0 \\
&:= r(x,e), && \text{ha } c = 0 \wedge y = 0 \\
&:= o(x/2, y/2, d/2, 2e + (d\%2)), && \text{különben, és } d = b(x,y,c) \\
a + b &:= o(a,b,0,1)
\end{aligned}
$$

Azaz tároljuk egy explicit verembenen a részeredményt, majd fejtsük vissza, ha már nincs több összeadandó bit. Pl.

$$
\begin{aligned}
2 + 3 = b10 + b11 &= o(2,3,0,1) = o(b10, b11, b0, b1) \\
&= o(1,1,0,3) = o(b1, b1, b0, b11) \\
&= o(0,0,1,6) = o(b0, b0, b1, b110) \\
&= o(0,0,0,13) = o(b0, b0, b0, b1101) \\
&= r(0,13) = r(b0, b1101) \\
&= r(1,6) = r(b1, b110) \\
&= r(2,3) = r(b10, b11) \\
&= r(5,1) = r(b101, b1) \\
&= 5
\end{aligned}
$$

```
Algoritmus I1(n):
x:=0
for i:=1 to n
  x:=x+1
```

```
Algoritmus I2(k, n):
x:=0
for i:=1 to n
  x:=x+k
```

Legyen BK (x) az x algoritmus bitkomplexitása.

$$
\begin{aligned}
&\mathrm{BK}\,(\mathrm{I1(n)}) \geq 2\,\mathrm{n} \\
&\mathrm{BK}\,(\mathrm{I1(n)}) \leq 2\left(\mathrm{n} + \frac{\mathrm{n}}{2} + \frac{\mathrm{n}}{4} + \frac{\mathrm{n}}{8} + \dots\right) \leq 4\,\mathrm{n} \\
&\mathrm{BK}\,(\mathrm{I1(n)}) \in \Theta(\mathrm{n})
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{h} &:= \lfloor log_2 \mathrm{k} \rfloor + 1 \\
\mathrm{BK}\,(\mathrm{I2(k,n)}) &\geq 2\,\mathrm{h}\,\mathrm{n} \\
\mathrm{BK}\,(\mathrm{I2(k,n)}) &\leq 2(h+1)\left(\mathrm{n} + \frac{\mathrm{n}}{2} + \frac{\mathrm{n}}{4} + \frac{\mathrm{n}}{8} + \dots\right) \leq 4(h+1)\,\mathrm{n} \\
\mathrm{BK}\,(\mathrm{I2(k,n)}) &\in \Theta(\mathrm{n} \log \mathrm{k})
\end{aligned}
$$

# 3 Eratoszthenész szitája

```
Algoritmus E(n):
legyen 2..n semmi se lehuzva
for i:=2 to n
  if i nincs lehuzva
    i prim
    for j:=2*i to n step i
      legyen j lehuzva
```

Az egészrészek mágikus elhagyásával, és Mertens tételének felhasználásával

$$\mathrm{BK}\left(\mathrm{E(n)}\right) = \Theta(\mathrm{n}) + \Theta(\mathrm{n}) + \sum_{\substack{\mathrm{p}\leq \mathrm{n}\\ \mathrm{p\ prím}}} \mathrm{BK}\left(\mathrm{I2}(\mathrm{p}, \left\lfloor \frac{\mathrm{n}}{\mathrm{p}} \right\rfloor)\right)$$

$$\sum_{\substack{\mathrm{p}\leq \mathrm{n}\\ \mathrm{p\ prím}}} \mathrm{BK}\left(\mathrm{I2}(\mathrm{p}, \left\lfloor \frac{\mathrm{n}}{\mathrm{p}} \right\rfloor)\right) \geq \sum_{\substack{\mathrm{p}\leq \mathrm{n}\\ \mathrm{p\ prím}}} \frac{2\,\mathrm{n}(1+\log_2 \mathrm{p})}{p} = \frac{4n}{\ln 2} \sum_{\substack{\mathrm{p}\leq \mathrm{n}\\ \mathrm{p\ prím}}} \frac{\ln \mathrm{p}}{p} \in \Theta(\mathrm{n} \log \mathrm{n})$$

$$\sum_{\substack{\mathrm{p}\leq \mathrm{n}\\ \mathrm{p\ prím}}} \mathrm{BK}\left(\mathrm{I2}(\mathrm{p}, \left\lfloor \frac{\mathrm{n}}{\mathrm{p}} \right\rfloor)\right) \leq \sum_{\substack{\mathrm{p}\leq \mathrm{n}\\ \mathrm{p\ prím}}} \frac{4\,\mathrm{n}(2+\log_2 \mathrm{p})}{p} \leq \frac{12n}{\ln 2} \sum_{\substack{\mathrm{p}\leq \mathrm{n}\\ \mathrm{p\ prím}}} \frac{\ln \mathrm{p}}{p} \in \Theta(\mathrm{n} \log \mathrm{n})$$

$$\mathrm{BK}\left(\mathrm{E(n)}\right) \in \Theta(\mathrm{n} \log \mathrm{n})$$

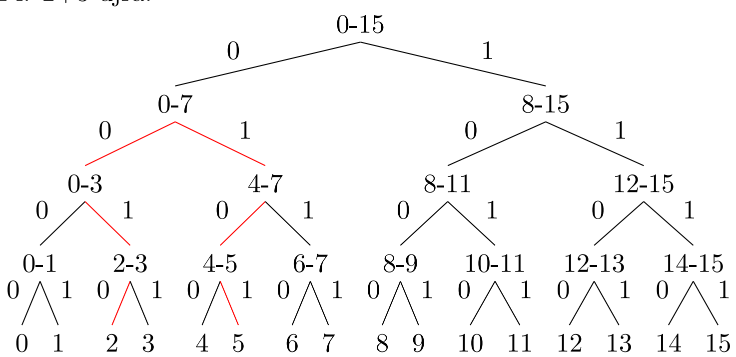
# 4 A természetes számok prefix fája

Képzeljük el, hogy n-ig szitálnánk, és valahonnan már van egy binárs, teljes, kiegyensúlyozott fa, aminek legalább $2\,\mathrm{n}+1$ levele van. A fa élei legyenek 0-val címkézve, ha bal gyerekhez vezetnek, 1-gyel, ha jobb gyerekhez. Így ha h a fa magassága, a levelek ábrázolhatják a természetes számokat 0-tól $2^h - 1$-ig.

Ha még azt is elképzeljük, hogy minden csúcsban azt is eltároljuk, hogy csúcsból a gyökérig vezető úton van-e még 1-es bit, akkor bármelyik csúcsból a gyökérbe vezető út teljesíti az ismételt összeadásnál megkövetelt feltételeket.

Az összeadást lehet módosítani úgy, hogy az egyik argumentuma mindig a fa egyik levele, és az eredmény is a fa levele lesz.

Pl. 2+3 újra:



# 5 Eratoszthenész szitája fával

Eratoszthenész szitáját így is implementálhatjuk

```
Algoritmus E2(n):
for i:=2 to n
  if letezik p i-hez rendelve
    for minden p i-hez rendelve
      rendeljuk p-t (i+p)-hez
  else
    i prim
    rendeljuk i-t 2i-hez
```

Ha számokhoz konstans időben rendelhetünk prímeket, és olvashatjuk azokat ki, akkor

$$\mathrm{BK}\,(\mathrm{E2(n)}) = \mathrm{BK}\,(\mathrm{E(n)})$$

Legyen n a fa egy levele, k egy listával ábrázolt szám, és $CS(\mathrm{n},\mathrm{k})$ azon csúcsok halmaza a fában, amiket az $n+k$ elvégzése érint. Legyen

$$CS^*(\mathrm{n},\mathrm{k}) = CS(\mathrm{n},1) \cup CS(\mathrm{n}+1,1) \cup \ldots \cup CS(\mathrm{n}+\mathrm{k}-1,1)$$

Ekkor

$$CS(\mathrm{n},\mathrm{k}) \subseteq CS^*(\mathrm{n},\mathrm{k})$$

Ha az E2 algoritmust úgy implementáljuk, hogy egy $i+p$-hez rendelésnél nem végzi el az összeg teljes kiszímítását, hanem egy közbülső csúcshoz rendeli az addig elvégzett részleges számítást, akkor az $i := i+1$ számítások egyike érinti a félbehagyott számítást, és hamarabb érinti, mint hogy $i+p$ -ig érne.

# 6 Lustán elképzelt fa

Eddig azt fejtegettük, hogy ha el tudunk képzelni egy elég nagy fát elég gyors műveletekkel, akkor Eratoszthenész szitáját implementálhatjuk úgy, hogy a lehúzásokkal sorban haladhatunk, és az aszimptotikus futási ideje megegyezik a szokásos, tömbben össze-vissza lehúzásos implementációval.

A előző fejezet részleges számításainál nem nyilatkoztunk arról, hogy melyik csúcsban érdemes abbahagyni az összeadásokat, és hogy mikor érdemes folytatni, és meddig érdemes folytatni.

Minden összeadás útja a fában úgy néz ki, hogy valahány lépést halad a csúcs felé, egy csúcsban, ahol a bal gyerek felől érkezik, nem a szülő felé folytatja a leszámolást, hanem a jobb gyerek felé folytatja az utat, majd az eredmény függvényében a gyerekek felé haladva kiválaszt egy levelet. Ez a forduló mindig rajta van az aktuális pozíciót a gyökérrel összekötő úton.

Tároljuk a részleges számításokat a forduló utáni közvetlen, jobb gyerekben. A félbehagyott számításokat mindig akkor folytassunk, amikor az aktuális pozíció eggyel növelésének elvégzésénél egy csúcsba a szülője felöl érkezünk, és olyankor egyetlen lépést végezzünk el a félbehagyott számításokban, azaz a fában egyetlen szinttel mozgassuk lejjebb. Ilyenkor a fában felfele haladva nem is fogunk találkozni részleges számítással.

Az is megfigyelhető hogy az új aktuális pozícióban azok a prímek szitálnak, amik az $i := i+1$ forduló jobb gyerekében lettek félbehagyva, és a részeredmény minden bitje 0, azaz a legbalabb levelébe tartoznak, ami éppen az új aktuális pozíció a szitatáblában.

Ha így járunk el, akkor a nagy elképzelt fa helyett elég egy láncolt listát tárolnunk, a lista hossza arányos a fa magasságával, és a lista minden elemében el kell tárolni az ahhoz a magassághoz tartozó részleges számítások vödrét, és egy bitet, amiben az aktuális pozíció számjegyei vannak.

Nézzük kicsit részletesebben az algoritmust.

Ha a fában az aktuális pozíció $i$, és $p$-t $i+p$-hez akarjuk rendelni, akkor az összeadás részleges elvégzésével keressük meg az első csúcsot, ami nincs rajta az $i \to$ gyökér úton, és ahhoz kell rendelni a részeredményt.

Ha szitálás közben az $i$ az új aktuális pozíció a szitattáblában, akkor az $(i-1)+1$ részleges elvégzésével keressük meg, hogy melyik az első csúcs, ami nincs rajta az $(i-1) \to$ gyökér úton. Ilyenkor ezalatt a szint alatt nincs félbehagyott számítás. Viszgáljuk meg az összes részeredményt, amit ebben a csúcsban találunk. A részeredmény legelső bitjét vizsgálva, amíg az 0, addig a fában haladjunk a bal gyerek felé, majd ha még nem üres a részeredmény, akkor egyet a jobb gyerek felé.

Amikor végeztünk az összes talált részszámítással, akkor nézzük meg, mi került a legbalabb gyerekbe, azaz az $i$ aktuális pozíció levelébe. Ha üres, akkor $i$ prím, és fel kell venni a fába $2i$ pozícióba. Ha nem üres, akkor minden talált $p$ prímet fel kell venni a fába $i+p$ pozícióba.

Igazából amikor az $i+x$ fordulóját keressük, arra vagyunk kiváncsiak, hogy melyik a legnagyobb helyiértékű bit, ahol $i$ és $i+x$ különbözik, de ha ezt a fa gyökere felől néznénk, akkor a műveletigényt a fa magasságával lehetne csak becsülni, még alulról összeadva a műveletigény becsülhető $\log x$-szel.

Példaként szitáljunk 2-től 15-ig a már lerajzolt fában. A példában $(p : i)$ jelentse azt, hogy a $p$ prímet az $i$ pozícióba kéne tenni. Vegyük sorra, hogy mi lenne ebben fát ábrázoló listában, ahogy az aktuális pozíció 2-től 15-ig lép. A piros számokjegyek a legmagasabb változott helyiértékek.

$i = 2$, prím

| lista-index | számjegy | prímek utána |
|---|---|---|
| 2 | 0 | (2: 4+0) |
| 1 | 1 | |
| 0 | 0 | |

$i = 3$, prím

| lista-index | számjegy | prímek utána |
|---|---|---|
| 2 | 0 | (2: 4+0), (3: 4+2) |
| 1 | 1 | |
| 0 | 1 | |

$i = 4$

| lista-index | számjegy | prímek utána |
|---|---|---|
| 2 | 1 | |
| 1 | 0 | (3: 6+0), (2: 6+0) |
| 0 | 0 | |

$i = 5$, prím

| lista-index | számjegy | prímek utána |
|---|---|---|
| 3 | 0 | (5: 8+2) |
| 2 | 1 | |
| 1 | 0 | (3: 6+0), (2: 6+0) |
| 0 | 1 | |

$i = 6$

| lista-index | számjegy | prímek utána |
|---|---|---|
| 3 | 0 | (5: 8+2), (3: 8+1), (2: 8+0) |
| 2 | 1 | |
| 1 | 1 | |
| 0 | 0 | |

$i = 7$ prím

| lista-index | számjegy | prímek utána |
|---|---|---|
| 3 | 0 | (5: 8+2), (3: 8+1), (2: 8+0), (7: 8+6) |
| 2 | 1 | |
| 1 | 1 | |
| 0 | 1 | |

$i = 8$

| lista-index | számjegy | prímek utána |
|---|---|---|
| 3 | 1 | |
| 2 | 0 | (7: 12+2) |
| 1 | 0 | (5: 10+0), (2: 10+0) |
| 0 | 0 | (3: 9+0) |

$i = 9$

| lista-index | számjegy | prímek utána |
|---|---|---|
| 3 | 1 | |
| 2 | 0 | (7: 12+2), (3: 12+0) |
| 1 | 0 | (5: 10+0), (2: 10+0) |
| 0 | 1 | |

$i = 10$

| lista-index | számjegy | prímek utána |
|---|---|---|
| 3 | 1 | |
| 2 | 0 | (7: 12+2), (3: 12+0), (5: 12+3), (2: 12+0) |
| 1 | 1 | |
| 0 | 0 | |

$i = 11$ prím

| lista-index | számjegy | prímek utána |
|---|---|---|
| 4 | 1 | (11: 16+6) |
| 3 | 1 | |
| 2 | 0 | (7: 12+2), (3: 12+0), (5: 12+3), (2: 12+0) |
| 1 | 1 | |
| 0 | 1 | |

$i = 12$

| lista-index | számjegy | prímek utána |
|---|---|---|
| 4 | 1 | (11: 16+6) |
| 3 | 1 | |
| 2 | 1 | |
| 1 | 0 | (2: 14+0), (3: 14+1), (5: 14+1), (7: 14+0) |
| 0 | 0 | |

$i = 13$ prím

| lista-index | számjegy | prímek utána |
|---|---|---|
| 4 | 1 | (11: 16+6), (13: 16+10) |
| 3 | 1 | |
| 2 | 1 | |
| 1 | 0 | (2: 14+0), (3: 14+1), (5: 14+1), (7: 14+0) |
| 0 | 1 | |

$i = 14$

| lista-index | számjegy | prímek utána |
|---|---|---|
| 4 | 1 | (7: 16+5), (2: 16+0), (11: 16+6), (13: 16+10) |
| 3 | 1 | |
| 2 | 1 | |
| 1 | 1 | |
| 0 | 0 | (3: 15+0), (5: 15+0) |

$i = 15$

| lista-index | számjegy | prímek utána |
|---|---|---|
| 4 | 1 | (3: 16+2), (5: 16+4), (7: 16+5),<br>(2: 16+0), (11: 16+6), (13: 16+10) |
| 3 | 1 | |
| 2 | 1 | |
| 1 | 1 | |
| 0 | 1 | |

# 7 Egyebek

Előnyök:

- akár s&m funkcionálisan is implementálható bitekkel és láncolt listákkal
- fixnum implementációban a gyökérbe vezető lista lehet egy tömb, és a forduló csúcs meghatározható xor és bit-scan-right assembly utasítsokkal
- ha a fa bővíthető, nem kell eldönteni előre, hogy meddig szitálunk
- aszimptotikusan optimális
- cache oblivious, talán
- könnyen implementálható fixnum
- talán általánosítható egy cache oblivious queue-ra, ahol a betett elem nagyságától függ mindenféle idő, nem az elemek számától

Hátrányok:

- a memóriaigénye $\Theta$(n)
- a prímek semmilyen tulajdonságát nem használja ki
- macerás implementálni bignum
- macerás formalizálni
- nem tudom, hogy kéne formálisan bizonyítani a helyességét
- nem tudom, hogy kéne formálisan bizonyítani a futási időt

- nem tudom, hogy kéne formálisan bizonyítani, hogy cache oblivious

Nézegethető dolgok:

- fixnum ennyire könnyű szó szerint implementálva
  `https://github.com/zooflavor/lazysieve/blob/master/ls2.c`
- csináltam egy függvényillesztőt
  `http://arrostia.web.elte.hu/lazysieve/ffit/ffit.html`
  ennek a helyességével vannak gondok, a polinomiális regresszióra is csak ráfogtuk nummod2-n, hogy helyes
  a pixelek szemre illeszkednek és színesek...
- összehasonlítottam a futási idejét a legegyszerűbb szitával, és csináltam egy grafikont belőle
  `http://arrostia.web.elte.hu/lazysieve/speed/`
  a csv file-okkal ki lehet próbálni az illesztőt
  az oszlopok
  - n-ig szitálva, a többi oszlop nanosec.
  - array: egy nagy tömbben a szokásos szitálás, algoritmus E
  - ls2-1: a fixnum implementáció, a szitatáblában egyesével halad
  - ls2-2: mint az ls2-1, de a szitatáblátban rövidebb szakaszonként halad, a fa-listába csak azokat a prímeket teszi, amik nagyobbak, mint a szakasz hossza, a kis prímeket egy nagy tömbben gyűjti, és csak mindig végigloop-ol rajtuk
  - ls2-3: mint az ls2-2, de a nagy prímeken használja a 2-3-5 kereket
- megszámoltam, hogy hipotetikus időben mit segít egy gyors cache
  `http://arrostia.web.elte.hu/lazysieve/cache/`
  a cache igyekszik a lista kisebb indexeit tárolni, amik gyakarabban sorra kerülnek
  a hipotetikus idő 1000ns ha sikerül a cache-be írni, vagy onnan olvasni, és 1000000ns, ha csak a lassú tárba sikerült írni, vagy onnan kell olvasni.
  az oszlopok
  - n-ig szitálva, a többi oszlop nanosec.
  - a: ez igazi időben van, a hipotetikus számolás kikapcsolva
  - cx: x prím nagyságú a cache

  említésre méltó, hogy a cache nélküli hipotetikus idő mintára remekül illeszthető az elvárt $n \ln n$