

## 7.4 Sormenjälkitekniikka

Tarkastellaan ensimmäisenä esimerkkinä *pitkien merkkijonojen vertailua*.

**Ongelma:** Ajatellaan, että kaksi  $n$ -bittistä ( $n \gg 1$ ) tiedostoa  $x$  ja  $y$  sijaitsee eri tietokoneilla. Halutaan tarkistaa, päteekö  $x = y$ . Ei kuitenkaan haluta lähettää isoja tiedostoja verkon yli.

### Menetelmä:

1. Valitaan satunnainen  $m$ -bittinen alkuluku  $p$ .
2. Tiedostolle  $x$  lasketaan *sormenjälki*

$$h_p(x) = \ell(x) \bmod p$$

missä  $\ell(x)$  on  $x$  (bittijonoksi ja siten) luonnolliseksi luvuksi tulkittuna; vastaavasti  $h_p(y)$ .

3. Verrataan sormenjälkiä; vaatii  $O(m)$  bitin kommunikoimista. Jos  $h_p(x) \neq h_p(y)$ , tiedetään varmasti  $x \neq y$ . Jos  $h_p(x) = h_p(y)$ , uskotaan  $x = y$  (mutta tässä voi tulla virhe).

Kuinka suuri  $m$  tarvitaan, että virhetodennäköisyys on siedettävä?

Olkoon  $\pi(n)$  niiden alkulukujen määrä, jotka ovat korkeintaan  $n$ . Siis esim.

$$\pi(10) = |\{2, 3, 5, 7\}| = 4.$$

*Alkulukulauseen* mukaan

$$\pi(n) \sim \frac{n}{\ln n}.$$

Esitetty menettely johtaa virheeseen, jos  $\ell(x) \neq \ell(y)$  mutta  $\ell(x) - \ell(y)$  on jaollinen luvulla  $p$ .

Koska  $|\ell(x) - \ell(y)| \leq 2^n$ , luvulla  $\ell(x) - \ell(y)$  ei mitenkään voi olla enempää kuin  $n$  alkutekijää. Virheeseen johtavia lukuja  $p$  on siis korkeintaan  $n$  kappaletta. Siis

$$\text{virhetn.} \leq \frac{n}{\pi(2^m)} \sim \frac{n}{2^m/m \ln 2} = \frac{nm \ln 2}{2^m}.$$

Siis jos esim. tiedostot ovat 100 Mb ja  $m = 64$ , saadaan virhetodennäköisyydeksi noin  $2,0 \cdot 10^{-9}$ . (Alkulukulauseen arvio on jo melko tarkka 64-bittisillä luvuilla.)

Toisena esimerkkinä **joukkojen yhtäsuuruusvertailu**.

Tehtävänä on ylläpitää jonkin suuren perusjoukon  $U$  osajoukkoja  $S_1, \dots, S_n$ .

Aluksi joukot ovat tyhjiä. Sallittuja operaatioita ovat

**insert( $i, x$ )**:  $S_i := S_i \cup \{x\}$  (tässä siis  $x \in U$ )

**equals( $i, j$ )**: palauttaa tosi joss  $S_i = S_j$

Näyttäisi, että deterministisesti equals( $i, j$ ) vaatisi ajan  $O(|S_i| + |S_j|)$  millä tahansa mieleen tulevalla talletusrakenteella.

Esitetään satunnaisratkaisu, joka tekee  $m$  insert/equals-operaatiota odotusarvoisesti ajassa  $O(m \log(m/\varepsilon))$  missä  $\varepsilon$  on equals-operaatioille sallittu virhetodennäköisyys. (Parametrit  $m$  ja  $\varepsilon$  pitää antaa ennakoita.)

Valitaan  $k = \lceil \log(m/\varepsilon) \rceil$ . Ideana on käyttää kullekin joukolle  $S_i$  sormenjälkenä  $k$ -bittistä lukua

$$s[i] = \bigoplus \{ r(x) \mid x \in S_i \}$$

missä  $r(x)$  on alkiolle  $x$  valittu satunnainen  $k$ -bittinen koodi ja  $\oplus$  on bittikohtainen XOR.

**insert**( $i, x$ ):

**if**  $r(x)$  ei määritelty

**then**  $r(x) := \text{random}(0 \dots 2^k - 1)$

$s[i] := s[i] \oplus r(x)$

**equals**( $i, j$ ):

**return**  $s[i] = s[j]$

Aluksi  $s[i] = 0$  kaikilla  $i$ .

Koodit  $r(x)$  voidaan taulukoida esim. hajauttamalla.

Tarkastellaan nyt operaatiota  $\text{equals}(i, j)$ , jossa sattuu virhe.

Merkitään  $R = S_i \Delta S_j = (S_i - S_j) \cup (S_j - S_i)$ . Ehto  $s[i] = s[j]$  on yhtäpitävää sen kanssa, että

$$\bigoplus \{ r(x) \mid x \in R \} = 0.$$

Kun  $R \neq \emptyset$  on kiinteä joukko alkioita  $x$  ja koodit  $r(x)$  ovat tasaisesti jakautuneita  $k$ -bittisiä, niin myös  $\bigoplus \{ r(x) \mid x \in R \}$  on tasaisesti jakautunut  $k$ -bittinen ja

$$P(\bigoplus \{ r(x) \mid x \in R \} = 0) = \left(\frac{1}{2}\right)^k.$$

Siis yhden nimenomaisen operaation  $\text{equals}(i, j)$  virhetodennäköisyys, kun  $S_i \neq S_j$ , on

$$2^{-k} \leq 2^{-\log(m/\varepsilon)} = \frac{\varepsilon}{m}.$$

Jos  $S_i = S_j$ , virhetodennäköisyys on nolla.

Siis todennäköisyys että  $m$  operaatiossa sattuu ainakin yksi virhe on korkeintaan

$$m \cdot \frac{\varepsilon}{m} = \varepsilon.$$

## 7.5 Symmetrian rikkominen

Tarkastellaan hajautettua järjestelmää, jossa on  $n$  identtistä mutta ei välttämättä synkronista prosessoria.

Jotta hyödyllistä rinnakkaisuutta saadaan aikaan, prosessorien pitää erityä. (Muuten vain suoritetaan samat asiat  $n$  kertaa.) Tämä voidaan tehdä valitsemalla yksi prosessori *johtajaksi*, joka sitten jakaa muille työt.

Oletetaan, että kukin prosessori tietää prosessorien kokonaismäärän  $n$ , voi lähettää viestejä koko verkolle ja osaa generoida satunnaislukuja omalla siemenluvullaan muista prosessoreista riippumatta.

Seuraava algoritmi (jota kaikki prosessorit suorittavat) valitsee johtajan käyttäen odotusarvoisesti alle kolme viestintäkierrosta prosessorien lukumäärästä riippumatta. Yhdellä viestintäkierroksella jokainen prosessori saa lähettää yhden viestin kaikille muille.

## Johtajanvalinta-algoritmi:

1. Aseta  $m := n$ .
2. Valitse  $r := \text{random}(1 \dots m)$
3. Lähetä kaikille muille prosessoreille luku  $r$ .
4. Odota, kunnes olet saanut kaikilta muilta prosessoreilta niiden valitsemat luvut.
5. Olkoon  $k$  niiden prosessorien lukumäärä (sinä itse mukaanlukien) jotka valitsivat luvun 1.
6.
  - Jos  $k = 0$ , palaa kohtaan 2.
  - Jos  $k \geq 1$  ja  $r \neq 0$ , lopeta; sinusta ei tule johtajaa.
  - Jos  $k \geq 2$  ja  $r = 1$ , aseta  $m := k$  ja palaa kohtaan 2.
  - Jos  $k = 1$  ja  $r = 1$ , olet johtaja.

Siis valinnalla  $r = 1$  pääsee mukaan seuraavalle kierrokselle. Jos kukaan ei valinnut  $r = 1$ , kaikki pysyvät kuitenkin mukana.

Muuttujassa  $m$  pidetään kirjaa mukana olevien prosessorien määrästä.

# Yhteenveto

Algoritmia suunnitellessa on hyvä pitää mielessä sille asetettavat vaatimukset

- **oikeellisuus**
- aikavaativuus
- suorituskyky ml. ”vakiokertoimet”

Syötteen koko suhteessa käytettävissä olevaan laskentakapasiteettiin rajoittaa ylipäänsä kyseeseen tulevia ratkaisumenetelmiä (peruuttaminen vs. dynaaminen ohjelmointi vs. ahne)

Paras ja tehokkain ratkaisu on usein palauttaminen tunnettuun ongelmaan (ja olemassaolevaan toteutukseen).

Teoreettisia peruskysymyksiä:

- satunnaisuuden ja epädeterminismin vaikutus laskentavoimaan
- approksimoituvuus
- optimaaliset ratkaisut perusongelmille