Hajautus

- eräs (osittainen) toteutus joukko-tietotyypille
- operaatiot **insert** ja **search** pyritään tekemään erittäin nopeiksi
- tärkeä tekniikka käytännön ohjelmoinnissa
- valmiita toteutuksia on, mutta väärät valinnat voivat johtaa huonoon lopputulokseen
- emme tällä kurssilla mene kovin syvälle taustalla olevaan teoriaan, mutta on tärkeä ymmärtää keskeiset asiat, jotka voivat mennä pieleen

Esimerkki: shakkiasemien arvioiminen

- shakkiohjelmat kokeilevat runsaasti erilaisia siirtovaihtoehtoja ja arvioivat syntyviä asemia (eli tilanteita)
- arvio voi olla esim. ”+0.5” jolloin koneen mielestä valkealla on noin puolen sotilaan arvoinen etu
- asemien arvioimiseen kuluvan ajan minimoiminen on olennaista suorituskyvylle
- sama asema voi syntyä useilla eri tavoilla (pelaamalla samat siirrot eri järjestyksessä), mutta sitä ei haluta arvioida kuin kerran
- siis tarvitaan tietorakenne, joka tukee seuraavia:
  - Onko asema X jo arvioitu?
  - Jos on, niin mikä oli arvion lopputulos?
  - Merkitse muistiin asemalle X arvio v.

Havainto: ainakin periaatteessa tähän kelpaa abstrakti tietotyyppi *joukko*:

- avaimena asema shakkipelissä
- datana arvio asemasta

Verrattuna aiempiin esimerkkeihin

- potentiaalisten avainten joukko todella suuri, avaimet monimutkaisia
- **search**-operaation tehokkuusvaatimus ankara
- myös **insert** ja ehkä **delete** tarvitaan, muista ei niin väliä

Hajautuksen perusajatus

- mahdollisten avainten joukko U hyvin suuri
- mikä tahansa yksittäinen sovellus käyttää vain häviävän pientä osaa joukon U alkioista
- talletetaan pareja $\langle avain, data \rangle$, missä $avain \in U$ ja $data$ voi olla mitä tahansa
- avaimet ja data talletetaan hajautustauluun, jonka indeksit ovat väliltä $0 \ldots m-1$
- hajautustaulun koko m on paljon pienempi kuin avainten lukumäärä $|U|$
- hajautusfunktio $h\colon U \to \{0, \ldots, m-1\}$ kertoo, mihin kohtaan hajautustaulua kukin avain pitäisi sijoittaa

- hajautusfunktio siis h liittää jokaiseen avaimeen $x \in U$ indeksin $h(x) \in \{\, 0, \ldots, m-1 \,\}$
- voidaan ajatella, että hajautustaulussa on m lokeroa joihin varsinainen tieto talletetaan
- jos $h(x) = j$, se tarkoittaa, että avain x kuuluu lokeroon j
- koska lokeroita on m, mahdollisia avaimia on $|\, U \,|$, ja $m \ll U$, tulee pakostakin yhteentörmäyksiä eli tilanteita, joissa kaksi eri avainta kuuluu samaan lokeroon
- muodollisemmin yhteentörmäys on tilanne, jossa $x \neq y$ mutta $h(x) = h(y)$
- **Kysymys 1:** miten valitaan h niin, että yhteentörmäyksiä ei tule liikaa
- **Kysymys 2:** kun yhteentörmäyksiä kuitenkin tulee, miten ne käsitellään

Ennen kysymysten 1 ja 2 tarkempaa pohtimista katsotaan konkreettisuuden vuoksi esimerkkinä yksi usein toimiva perusratkaisu:

- oletetaan, että avaimet ovat luonnollisia lukuja: $U = \{ 0, \ldots, |U| - 1 \}$
- hajautusfunktiona jakojäännösmenetelmä (eli jakolaskumenetelmä): $h(x) = x \bmod m$ missä $x \bmod m$ tarkoittaa jakojäännöstä, kun kokonaisluku x jaetaan kokonaisluvulla m
- siis erityisesti $x \bmod m$ on joukossa $\{ 0, \ldots, m - 1 \}$ kaikilla x
- lokeroon j tulee näin ollen avaimet j, $j + m$, $j + 2m$, $\ldots$
- yhteentörmäykset käsitellään ketjuttamalla: lokeroon k tulevat avaimet (ja niihin liittyvä data) talletetaan linkitettyyn listaan
- varsinainen hajautustaulu $T[0 \ldots m - 1]$ sisältää m osoitinta
- $T[j]$ osoittaa lokeroa j vastaavan linkitetyn listan alkuun

Ketjuttamalla toteutetun hajautustaulun kaikki operaatiot (**insert**, **search**, **delete**) noudattavat seuraavaa kaavaa:

1. Etsi annettua avainta k vastaava lista (siis käytännössä osoitin listan alkuun) $L = T[h(k)]$.

2. Suorita haluttu operaatio listalle L.

Aikavaativuus on siis $O(1 + |\, L\,|)$, missä $|\, L\,|$ on listan pituus ja lisävakio tulee hajautusfunktion laskemisesta.

Miten hajautustaulu eroaa siitä, että pareja $\langle avain, data \rangle$ talletettaisiin tavalliseen taulukkoon:

tavallisen taulukon etuja:

- tavallinen taulukko käyttää muistia mahdollisimman tehokkaasti (jos koko on suunnilleen tiedossa etukäteen)
- tavallinen taulukko voidaan järjestää jne.

hajautustaulun keskeinen etu:

- avaimesta nähdään vakioajassa sen oikea paikka taulukossa
- tosin avainta voidaan vielä joutua etsimään listasta

Hajautustaulun tehokkuuden kannalta keskeinen suure on täyttösuhde (load factor)

- oletetaan lokeroiden lukumääräksi m
- oletetaan taulukkoon talletetuksi n avainta
- täyttösuhde on nyt $\alpha = n/m$

Tästä seuraa

- α on myös ylivuotoketjujen pituuksien keskiarvo
- jos $h(x)$ saa kunkin arvoista $0, \ldots, m-1$ samalla todennäköisyydellä, niin operaatioiden aikavaativuudet ovat odotusarvoisesti $O(1+\alpha)$
- Huom. edellä x on satunnainen, h ei

- yleensä pyritään pitämään täyttösuhde melko pienenä vakiona
- esim. Javan `HashMap` pitää oletusarvoisesti $\alpha \leq 0{,}75$
- täyttösuhteen pienentäminen nopeuttaa hakuja mutta lisää muistinkulutusta
- sopivan täyttösuhteen löytäminen voi vaatia kokeilua

Ei ole realistista yrittää valita niin pieni täyttösuhde, että yhteentörmäyksiltä kokonaan vältyttäisiin

- syntymäpäiväparadoksi: jos $n \geq \sqrt{2m}$ ja avaimet sijoitetaan tauluun tasaisen satunnaisjakauman mukaan, niin ainakin todennäköisyydellä $1/2$ tulee ainakin yksi yhteentörmäys
- tässä siis $\alpha = \sqrt{2/m} \ll 1$

- jos alkioiden lukumäärä ei ollut etukäteen tiedossa ja täyttösuhde näyttää kasvavan liian suureksi, voidaan tehdä uudelleenhajauttaminen (rehash)
  - varataan uusi kaksi kertaa suurempi taulu
  - siirretään alkiot vanhasta taulusta uuteen yksi kerrallaan
- tämä ei ole niin tehotonta kuin ensi silmäyksellä voisi luulla
- voidaan osoittaa, että operaatiot vievät edelleen keskimäärin $O(1 + \alpha)$ (samoin oletuksin kuin edellä)
- kuitenkin jotkin yksittäiset operaatiot (ne jotka laukaisevat uudelleenhajauttamisen) voivat olla hyvin hitaita

- jos hajautettavat avaimet eivät ole kokonaislukuja, ne pitää jotenkin muuntaa sellaisiksi esim. jakojäännösmenetelmän soveltamiseksi
- periaatteessa kaikki avaimet ovat tietokoneen muistissa bitteinä, ja bittiesitys voidaan tulkita kokonaisluvuksi
- tämä kuitenkin tuottaa helposti lukuja, jotka ovat liian suuria käsiteltäväksi
- tavat, jotka tarkastelevat vain osaa biteistä tai esim. ASCII-koodien summaa ovat usein huonoja
- hyväksi havaittu tapa merkkijonoille on

$$ascii(a_1) + c \cdot ascii(a_2) + c^2 \cdot ascii(a_3) + \cdot$$

  - c on sopivasti valittu parametri joka ei ole kakkosen potenssi
  - $ascii(a_i)$ on i:nnen merkin ASCII-koodi
  - laskutoimituksissa huomioidaan vain p vähiten merkitsevää bittiä sopivalla p

Hajautusfunktion valinta

Yleisiä huomioita:

1. oletetaan nyt avaimet luonnollisiksi luvuiksi: $U = \{\, 0, \ldots, |\, U \,| - 1 \,\}$

2. talletusalueena on taas taulu $T[0 \ldots m]$

3. jos avaimet jakautuvat tasaisesti joukkoon U, meille kelpaa mikä tahansa hajautusfunktio, joka sijoittaa suunnilleen $|\, U \,|/m$ avainta kuhunkin lokeroon

4. hyvän hajautusfunktion pitäisi kuitenkin sietää myös tyypillisiä tapoja, joilla avaimet voivat jakautua joukkoon U epätasaisesti:
   - esiintyy vain pieniä avaimia
   - kaikki avaimet ovat parillisia
   - ...

Hajautusfunktion valinta (2)

Jakojäännösmenetelmässä suositellaan, että hajautustaulun koko m on

- alkuluku
- ei lähellä mitään kakkosen potenssia

Miksi alkuluku:

- oletetaan että $m = pq$, missä p, q ykköstä suurempia kokonaislukuja
- jos hajautetaan avaimet $q, 2q, 3q, \ldots$, ne sijoittuvat vain lokeroihin $p, 2p, 3p, \ldots$ (ei tässä järjestyksessä)
- avainten jaollisuus yhteisellä tekijällä q on tilanne johon halutaan varautua

Miksi ei lähellä kakkosen potenssia:

- oletetaan esim. $m = 2^8 - 1$
- hajautetaan 32-bittinen luonnollinen luku x, jonka binääriesitys on $x_1 x_2 \ldots x_{32}$
- on melko helppo nähdä, että $h(x) = x \bmod m$ määräytyy summasta $x_1 \ldots x_8 + x_9 \ldots x_{16} + x_{17} \ldots x_{24} + x_{25} \ldots x_{32}$ (missä siis 32-bittisestä luvusta on lohkottu neljä 8-bittistä ja laskettu ne yhteen)

Hajautusfunktion valinta (3)

Vaihtoehto jakojäännösmenetelmälle on kertolaskumenetelmä

$$h(k) = \lfloor m \cdot \mathrm{fr}(Ak) \rfloor$$

- m on hajautustaulun koko
- $0 < A < 1$ on sopivasti valittu parametri
- eräs suositeltu arvo on $A = (\sqrt{5} - 1)/2 \approx 0{,}618$ (Knuth)
- idea on, että $\mathrm{fr}(Ak)$ jakautuisi suunnilleen tasaisesti välille $[0, 1)$
- tällöin $\lfloor m \cdot \mathrm{fr}(Ak) \rfloor$ jakautuu suunnilleen tasaisesti joukkoon $\{ 0, \ldots, m - 1 \}$ oli m mikä hyvänsä
- voidaan valita $m = 2^p$ sopivalla p laskujen helpottamiseksi

Universaalihajautus

- mille tahansa kiinteälle hajautusfunktiolle on olemassa avainjoukkoja, joilla se toimii erittäin huonosti

- tarkemmin: jos $|U| \geq (n-1)m + 1$, niin millä tahansa hajautusfunktiolla on olemassa n avainta, jotka kaikki menevät samaan lokeroon

  Todistus: Kiinnitä hajautusfunktio h ja hajauta kaikki joukon U avaimet. Ainakin yhteen lokeroon tulee ainakin $|U|/m \geq n$ avainta.

- universaalihajautuksen idea on valita satunnainen hajautusfunktio niin, että millä tahansa avainjoukolla se toimii todennäköisesti melko hyvin

- tarkemmin: joukko H hajautusfunktioita on universaali, jos mille tahansa avaimille x ja y pätee, että yhteentörmäyksen $h(x) = h(y)$ todennäköisyys on korkeintaan $1/m$, kun h valitaan satunnaisesti joukosta H

- eräs tapa on valita alkuluku $p > n$ ja satunnaiset kokonaisluvut $1 \leq a < p$ ja $0 \leq b < p$, minkä jälkeen

$$h(k) = ((ak + b) \bmod p) \bmod m$$

Avoin hajautus

- ketjutuksessa alkiot talletetaan varsinaisen taulun T ulkopuolelle emmekä etukäteen tarkalleen tiedä, kuinka paljon muistia kuluu
- myös muistiviittausten paikallisuus on huono
- avoimessa hajautuksessa tauluun T talletetaan itse alkiot, ei viitteitä
- yhteentörmäyksen sattuessa etsitään toiselle alkiolle vapaa paikka jostain toisesta kohdasta taulua
- yksinkertaisin menetelmä on lineaarinen kokeilu: avaimen k tallettamiseksi
  - Jos paikka $h(k)$ on tyhjä, talleta sinne.
  - Muuten jos paikka $(h(k) + 1) \bmod m$ on tyhjä, talleta sinne.
  - Muuten jos paikka $(h(k) + 2) \bmod m$ on tyhjä, talleta sinne.
  - ...

Lineaarisen kokeilun ongelma on ensisijainen kasautuminen (primary clustering)

- yhteentörmäykset täyttävät taulusta yhtenäisiä alueita
- mitä suurempi yhtenäinen alue on täytetty, sitä todennäköisemmin sinne tulee lisää yhteentörmäyksiä

Tämä ongelma voidaan välttää neliöisellä kokeilemisella: avaimen k tallettamiseksi

- Jos paikka $h(k)$ on tyhjä, talleta sinne.
- Muuten jos paikka $(h(k) + a + b) \bmod m$ on tyhjä, talleta sinne.
- Muuten jos paikka $(h(k) + 2a + 4b) \bmod m$ on tyhjä, talleta sinne.
- Muuten jos paikka $(h(k) + 3a + 9b) \bmod m$ on tyhjä, talleta sinne.
- ...
- Muuten jos paikka $(h(k) + i \cdot a + i^2 \cdot b) \bmod m$ on tyhjä, talleta sinne.
- tässä a ja b ovat sopivasti valittuja parametreja

Neliöinen kokeileminen välttää ensisijaisen kasautumisen, mutta kärsii silti toissijaisesta kasautumisesta:

- jos $h(x) = h(y)$, niin kaikki muutkin paikat, joihin avaimia x ja y yritetään laittaa, ovat samat

Toissijainenkin kasautuminen voidaan välttää kaksoishajautuksella: avaimen k tallettamiseksi

- Jos paikka $h(k)$ on tyhjä, talleta sinne.
- Muuten laske arvo $h'(k)$ missä h' on jokin toinen hajautusfunktio
- Jos nyt paikka $(h(k) + h'(k)) \bmod m$ on tyhjä, talleta sinne.
- Muuten jos paikka $(h(k) + 2h'(k)) \bmod m$ on tyhjä, talleta sinne.
- ...
- Muuten jos paikka $(h(k) + i \cdot h'(k)) \bmod m$ on tyhjä, talleta sinne.
- ...

Avoimessa hajautuksessa myös **search**-operaatiossa pitää seurata samaa kokeilujonoa

- esim. lineaarisessa kokeilussa jos etsitään alkiota x ja $T[h(x)]$ sisältää jonkin toisen avaimen, x voikin olla paikassa $T[h(x) + 1]$

- etsiminen voidaan lopettaa, kun löytyy joko etsitty avain tai vapaa kohta taulussa

Jotta etsimisen tosiaan voi turvallisesti lopettaa ensimmäiseen vapaaseen kohtaan, **delete** pitää toteuttaa laiskasti:

- käytetään jotain erityistä arvoa merkitsemään, että taulukon tässä kohdassa on ollut avain, joka on poistettu

- hakua *ei* voi lopettaa tällaiseen merkittyyn kohtaan, mutta lisäyksen siihen voi tehdä

Loppuhuomioita hajautuksesta

- riippuu aineistosta, mikä on hyvä hajautusmenetelmä
- jos tämä on tärkeää, kannattaa kokeilla eri menetelmiä
- kirjallisuudesta löytyy jonkin verran teoriaa ja paljon kokeilemalla hyväksi havaittuja menetelmiä
- hajautusfunktioilla ja niiden sukuisilla tekniikoilla on paljon muitakin sovelluksia kuin *joukko*-tietotyypin toteuttaminen: tietoturva, tarkistussummat, sormenjälkitekniikat, . . .