# 4. Hajautus

- tarkastellaan edelleen sivulla 22 määritellyn joukkotietotyypin toteuttamista
- useissa sovelluksissa riittää että operaatiot *insert*, *delete* ja *search* toimivat nopeasti
  - esim. sivun 101 opiskelijarekisteriesimerkissä on epätodennäköistä että opiskelijanumeron mukaan järjestetyssä indeksirakenteessa on mitään käyttöä operaatioille min, max, succ ja prev
  - myöskään puhelinluettelon numeroiden suhteen järjestetty indeksi ei näitä operaatioita tarvinne
  - sensijaan (opiskelijarekisterissä tai puhelinluettelossa) nimen mukaan järjestetyssä indeksissä operaatioille min ja succ voi olla käyttöä jos on tarvetta esim. listata kaikki nimet aakkosjärjestyksessä
- tapauksissa joissa insert, delete ja find operaatiot riittävät *hajautusrakenne* on varteenotettava tapa joukon toteutukselle

- merkitään kaikkien mahdollisten avainten joukkoa $U$:lla, esim. opiskelijarekisterissä tämä olisi kaikki mahdolliset opiskelijanumerot tai puhelinluettelossa kaikki mahdolliset puhelinnumerot
- otetaan käyttöön $m$-paikkainen *hajautustaulukko* $T$, joka on indeksoitu alkaen nollasta

  koska yleensä $U$:n koko on erittäin suuri (tai ääretön), on $m < |U|$

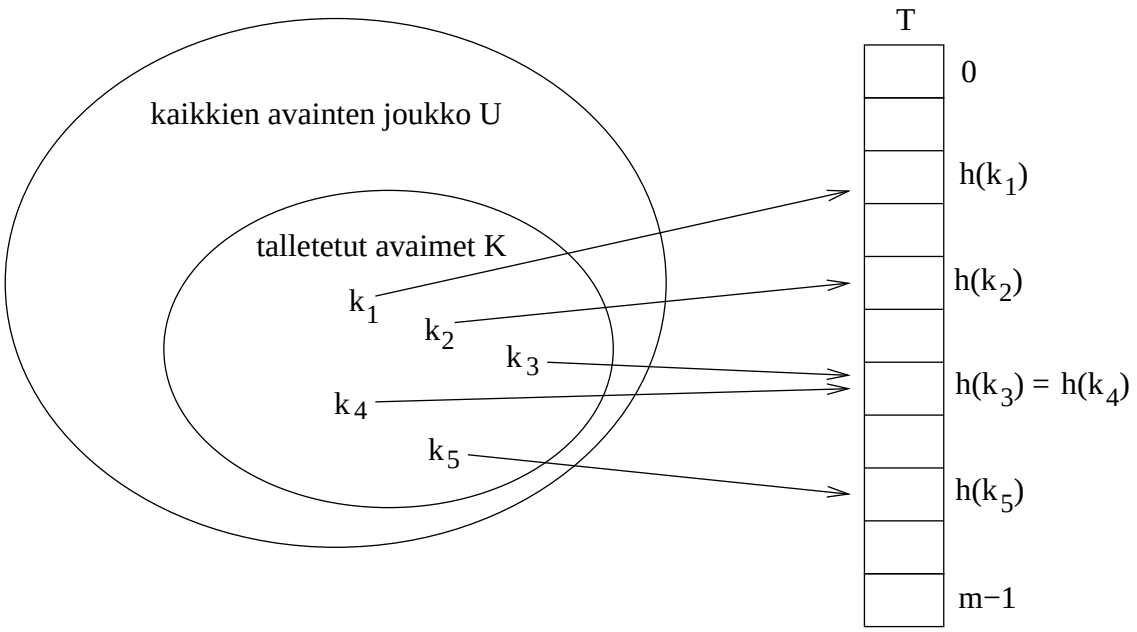
- määritellään *hajautusfunktio* $h : U \to \{0, \ldots, m-1\}$

  eli hajautusfunktio kuvaa jokaisen avaimen jollekin kokonaisluvulle väliltä $0, \ldots, m-1$

- toisin sanoen, hajautusfunktio kuvaa jokaisen avaimen jollekin taulukon $T$ indeksille

- sanotaan että $h(k_i)$ on avaimen $k_i$ *osoite* tai *kotiosoite*

- ideana on nyt että avain $k_i$ talletetaan taulukon $T$ paikkaan $h(k_i)$




- kuten kuva yllä kertoo, voi käydä niin että kaksi avainta saavat saman osoitteen eli joillekin avaimille $k_3, k_4$ voi olla $h(k_3) = h(k_4)$

- tällöin sanotaan että avainten $k_3$ ja $k_4$ välillä sattuu *yhteentörmäys*

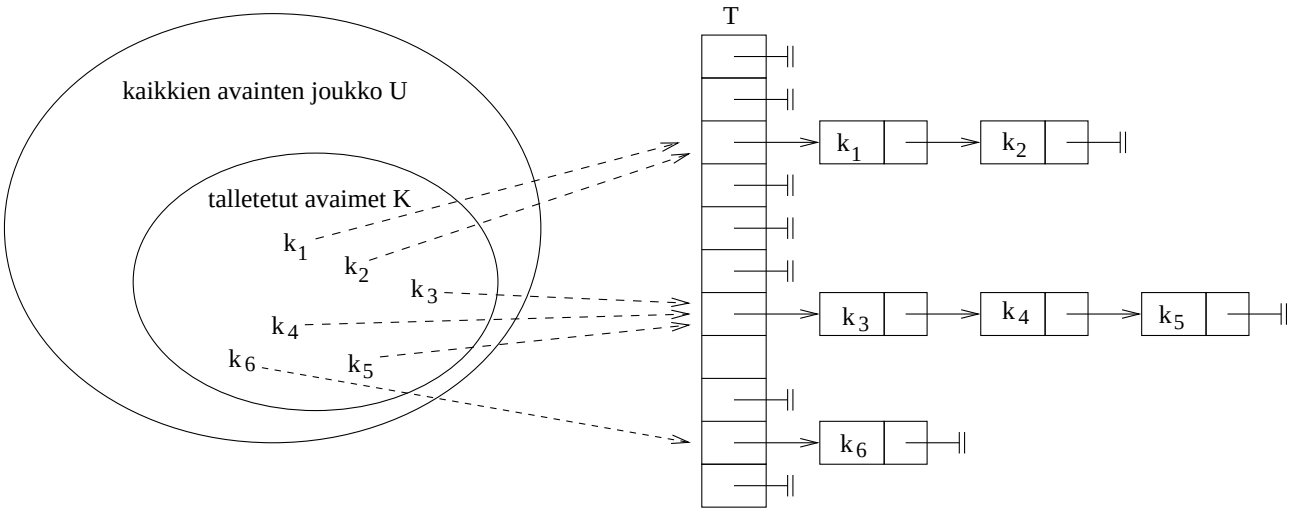
- oleellista onkin että hajautusfunktio $h$ suunnitellaan sellaiseksi että yhteentörmäyksiä tapahtuu mahdollisimman vähän

palataan hieman myöhemmin hajautusfunktion suunniteluun

- oli hajautusfunktio miten hyvä tahansa ei jokaiselle paikalle yleensä riitä omaa osoitetta jos talletettavien alkioiden määrä kasvaa tarpeeksi suureksi

## 4.1 Yhteentörmäykset ylivuotoketjuun

- yksi (ja ehkä paras) tapa ratkaista yhteentörmäykset on tallettaa yhteentörmäävät alkiot linkitettyihin listoihin
- kirjallisuudessa tätä yhteentörmäysten ratkaisustrategiaa sanotaan *ketjutukseksi* (eng. chaining) tosin terminologia vaihtelee kirjasta toiseen (esim. Arto Wiklan kurssimateriaalissa terminologia ei ole sama kun tällä kurssilla)
- alla olevassa kuvassa hajautustaulukon alkio $i$ sisältää linkin osoitteen $i$ saavien avainten sisältämään listan




- eli paikkaan $T[i]$ liittyvään listaan talletetaankin oikeastaan kaikki ositteen $i$ saavat alkiot, ei pelkästään yhteentörmäyksiä

- esitetään nyt toteutus ketjutuksella varustettuun hajautukseen käyttäen luvussa 2.4.(alkaen sivulta 32) esiteltyjä linkitettyjä listoja
- $T$ on nyt taulukko joka on määritelty välille $[0, \ldots, m-1]$, ja jokainen $T[i]$ sisältää viitteen linkitetyn listaan ensimmäiseen alkioon
- alussa hajautusalue on tyhjä eli kaikkien $T[i]$ arvo on NIL
- alkion etsiminen hajautusrakenteesta:

```
hash-search(T,k)
1   L ←T[h(k)]
2   x ←list-search(L,k)
3   return x
```

  - rivillä 1 selvitetään missä taulun indeksiin liittyvässä listassa on alkion paikka
  - rivillä 2 kutsutaan sivulla 38 määriteltyä listan etsintäoperaatiota selvittämään löytyykö etsittävää alkiota

- alkion lisääminen hajautusrakenteeseen

```
hash-insert(T,k)
1   L ←T[h(k)]
2   list-insert(L,k)
```

  - rivillä 1 selvitetään missä taulun indeksiin liittyvässä listassa on alkion paikka
  - rivillä 2 kutsutaan sivulla 39 määriteltyä listan lisäysoperaatiota laittamaan alkio paikoilleen

- alkion poistaminen hajautusrakenteesta

  hash-delete(T,x)
  1 L ←T[h(key[x])]
  2 list-delete(L,x)

  - rivillä 1 selvitetään missä taulun indeksiin liittyvässä listassa poistettava alkio on
  - rivillä 2 kutsutaan sivulla 34 määriteltyä listan poisto-operaatiota joka poistaa alkion listalta

- mikä on operaatioiden aikavaativuus?
- oletetaan että hajautusfunktion arvon laskeminen vie vakioajan
- luvusta 2.4. muistamme että lisäys ja poisto vievät vakioajan (jos kyseessä on kahteen suuntaan linkitetty lista)
- lisäys ja poisto vievät aikaa $\mathcal{O}(1)$, sillä vakioajassa voimme selvittää minkä taulukon indeksin päästä löytyy oikea ylivuotolista ja sekä alkion lisäys että poisto listalta ovat vakioaikaisia operaatioita
- luvusta 2.4 muistamme että $j$ alkiota sisältävältä listalta etsintä vie aikaa $\mathcal{O}(j)$
- hajautustaulusta etsinnän aikavaativuus siis riippuu ylivuotolistojen pituudesta
- pahimmassa tapauksessa kaikki avaimet saavat saman osoitteen ja ne on talletettu samalle listalle eli pahimmassa tapauksessa haku $n$ alkiota sisältävästä hajautusrakenteesta vie ajan $\mathcal{O}(n)$

- pahimman tapauksen tilanne on siis sama kuin alkiot olisi talletettu yhteen linkitettyyn listaan, selvästikään hajautusta ei ole tarkoitettu tällaisiin tilanteisiin, eli pahimman tapauksen analyysi ei anna oikeaa kuvaa menetelmästä
- analysoidaan hakua *keskimääräisessä tapauksessa*

  oletetaan että hajautustauluun on talletettu $n$ alkiota, ja että alkiot ovat jakautuneet tasaisesti eri ylivuotolistoille
- hajautustaulun tämänhetkinen *täyttösuhde* on $\alpha = n/m$, joka on samalla *keksinmääräinen ylivuotoketjun pituus*
- **Lause 4**

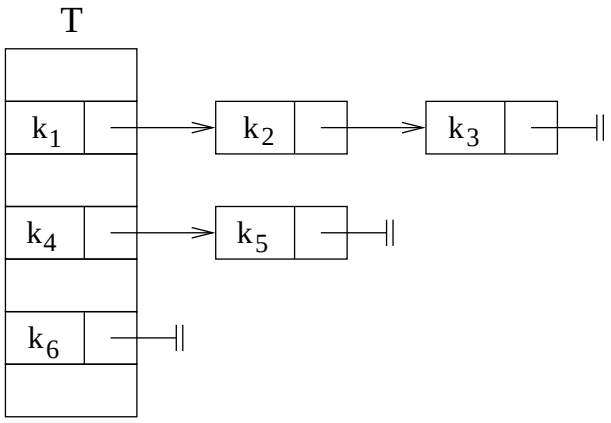
  Avaimen haku hajautusrakenteesta vie $\mathcal{O}(1 + \alpha)$ askelta

  *todistus:*

  > Todistus jakaantuu kahteen osaan: tuloksettomaan ja tuloksekkaaseen hakuun.
  >
  > Haettu avain voi olla yhtä suurella todennäköisyydellä missä tahansa ylivuotoketjussa. Avaimen $k$ tuloksettomassa haussa käydään läpi ylivuotolista T[h(k)], jonka pituus oletuksen mukaan on $\alpha$ alkiota. Kun otetaan vielä huomioon hajautusfunktion laskemiseen kuluva vakioaika saadaan vaatimukseksi $\mathcal{O}(1 + \alpha)$.
  >
  > Tuloksellisen haun analyysi vaatii sen verran todennäköisyyslaskennan osaamista että sivuutamme sen.
- jos hajautustaulun koko $m$ on suoraan verrannollinen talletettujen alkioiden lukumäärän $n$, esim. $m \approx n/3$, silloin on $n = \mathcal{O}(m)$

- nyt $\alpha = n/m = \mathcal{O}(m)/m = \mathcal{O}(1)$ eli tässä tapauksessa haun keskimääräinen aikavaativuus on $\mathcal{O}(1)$
- eli jos tiedämme että talletettavia alkioita on korkeintaan esim. viisinkertainen määrä hajautustaulun kokoon nähden, vie hakuoperaatio keskinmäärin ajan $\mathcal{O}(1)$

- edellä hajautustaulukko sisälsi ainoastaan osoitteet listojen alkuun, toinen mahdollisuus on tallettaa hajautustauluun jo yksi hajautettava alkio joka taas sisältää linkin ylivuotaviin alkioihin




- tätä ratkaisua kutsutaan kirjallisuudessa *suoraksi ketjutukseksi* ja aiempaa menetelmää *erilliseksi ketjutukseksi*

## 4.2 Hajautusfunktion valinta

- tavoitteena on että hajautusfunktion arvo tulee olla laskettavissa nopeasti ja että funktio jakaa avaimet tasaisesti hajautusalueelle
- hajautusfunktio olettaa yleensä että avain on kokonaisluku
- jos avain on kuitenkin esim. merkkijono, voidaan se helposti muuttaa kokonaisluvuksi

  tapoja on monia, olkoon avain $a_1 a_2 \ldots a_n$, oletetaan että funktio $ascii(a)$ palauttaa merkin $a$ ascii-koodin

  voidaan valita esim:

  - $k = ascii(a_1) + ascii(a_2) + \ldots + ascii(a_n)$
  - $k = ascii(a_1) + ascii(a_2) + ascii(a_3)$
  - $k = ascii(a_1) + ascii(a_{n/2}) + ascii(a_n)$
  - $k = ascii(a_1) + 128 \times ascii(a_2) + 128^2 \times ascii(a_3) + \ldots + 128^{n-1} \times ascii(a_n)$
  - $k = ascii(a_1) + 128 \times ascii(a_2) + 128^2 \times ascii(a_3)$
  - $k = ascii(a_1) + 128 \times ascii(a_{n/2}) + 128^2 \times ascii(a_n)$

  kaikilla omat hyvät ja huonot puolensa

- seuraavassa käytännössä toimivaksi osoittautuneita hajautusfunktioita
- *Jakojäännösmenetelmä*
  - $h(k) = k \ mod \ m$

    missä $m$ on alkuluku joka ei ole lähellä mitään 2:n potenssia

– esim. jos haluamme hajautustaulun johon talletetaan noin 2000 alkiota, ja sallimme että ylivuotolistoilla on keskimäärin 3 alkiota, valitaan $m = 701$

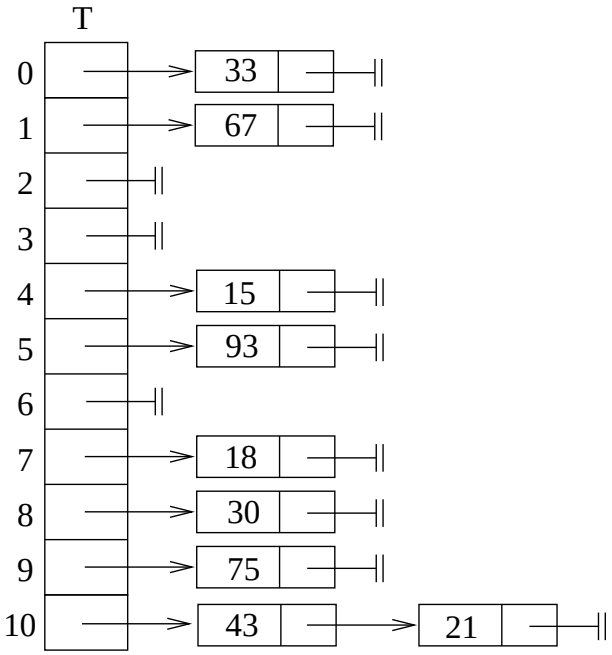
– nyt $m$ on alkuluku joka ei ole lähellä 2:n potenssia ja täyttöasteeksi tulee $2000/701 \approx 2.85$

– esim 2: oletetaan että käytössä hajautustaulukko jonka koko 11, hajautusfunktiona siis $h(k) = k \ mod \ 11$

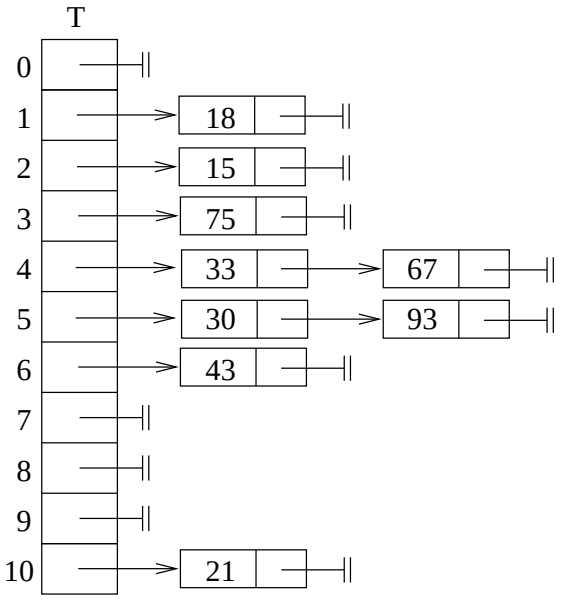
– hajautetaan avaimet 75, 43, 21, 15, 18, 33, 30, 67 ja 93, eli nyt

$h(75) = 9, \quad h(43) = 10, \quad h(21) = 10$
$h(15) = 4, \quad h(18) = 7, \quad h(33) = 0,$
$h(30) = 8, \quad h(67) = 1, \quad h(93) = 5$

– tuloksena

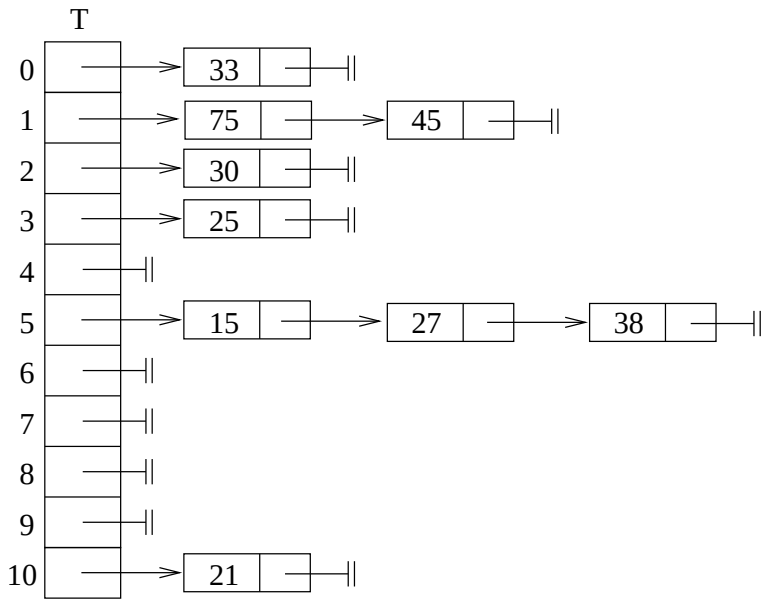


- *Kertolaskumenetelmä*
  - olkoon $A$ jokin luku $0 < A < 1$
  - merkitän $\lfloor x \rfloor$:llä luvun $x$ kokonaislukuosaa ja $fr(x)$:lla luvun $x$ desimaaliosaa
  - eli esim $\lfloor 3,14159 \rfloor = 3$ ja $fr(3.14159) = 0.14159$
  - $h(k) = \lfloor m \times fr(A \times k) \rfloor$
  - etuna jakolaskumenetelmään se että taulukon koon $m$ saa nyt valita vapaasti
  - D.Knuth: paras valinta $A = \frac{\sqrt{5}-1}{2} = 0.6180339887\ldots$
  - esim: oletetaan että käytössä hajautustaulukko jonka koko 11, ja hajautusfunktiona $h(k) = \lfloor 11 \times fr(0.618 \times k) \rfloor$
  - avaimet 75, 43, 21, 15, 18, 33, 30, 67 ja 93, nyt
    $h(75) = 3, \quad h(43) = 6, \quad h(21) = 10$
    $h(15) = 2, \quad h(18) = 1, \quad h(33) = 4,$
    $h(30) = 5, \quad h(67) = 4, \quad h(93) = 5$
  - tuloksena



- *Universaalihajautus*
  - valitaan alkuluku $p$ joka on vähintään yhtä suuri kuin talletettavien avainten lukumäärä $n$
  - valitaan *satunnaisesti* kaksi kokonaislukua $a$ ja $b$ väliltä $1 \leq a < p$ ja $0 \leq b < p$
  - muodostetaan hajautusfunktio seuraavasti:
    $h(k) = ((a \times k + b) \; mod \; p) \; mod \; m$
  - perusteluja näin muodostetun universaalihajautusfunktion hyvyydestä löytyy Karvin monisteesta tai Cormenin luvusta 11.3.3
  - esim: oletetaan jälleen että käytössä hajautustaulukko jonka koko 11
  - oletetaan että talletettavia avaimia korkeintaan 53 eli valitaan $p = 53$, ja valitaan satunnaisesti $a = 31$, $b = 17$ hajautusfunktiona siis $h(k) = ((31 \times k + 17) \; mod \; 53) \; mod \; 11$
  - avaimet 75, 43, 21, 15, 18, 33, 30, 67 ja 93, nyt
    $h(75) = 10 \; mod \; 11 = 1, \quad h(43) = 25 \; mod \; 11 = 3$
    $h(21) = 32 \; mod \; 11 = 10, \quad h(15) = 5 \; mod \; 11 = 5$
    $h(18) = 45 \; mod \; 11 = 1, \quad h(33) = 33 \; mod \; 11 = 0$
    $h(30) = 46 \; mod \; 11 = 2, \quad h(67) = 27 \; mod \; 11 = 5$ ja
    $h(93) = 38 \; mod \; 11 = 5$

– tuloksena




– valitsemalla satunnaiset $a$ ja $b$ toisin, olisi päädytty eri arvot antavaan hajautusfunktioon jonka käyttö olisi saattanut johtaa parempaan tulokseen

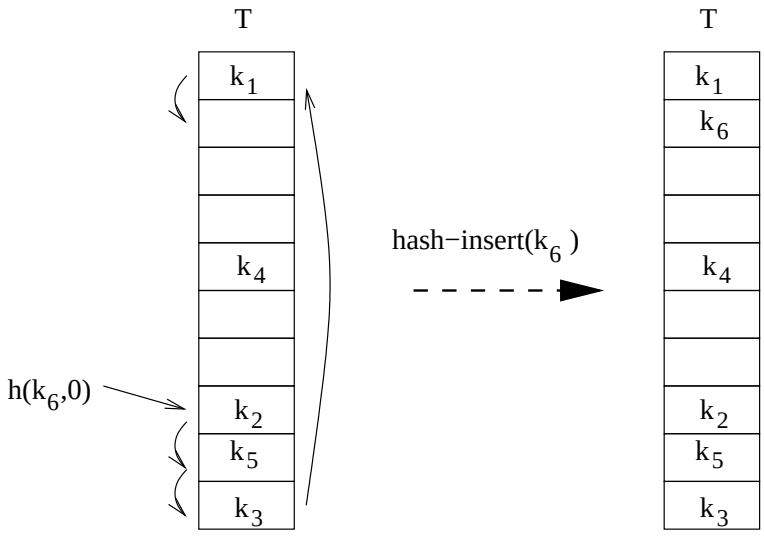
## 4.3 Avoin hajautus

- Ketjutuksen rinnalla toinen tapa ratkaista yhteentörmäävien solmujen ongelma on *avoin hajautus* (eng. open addressing)
- Avoimessa hajautuksessa *kaikki* alkiot talletetaan hajautustauluun
- jos paikka $T[h(k)]$ on varattu, etsitään alkiolle $k$ jokin muu paikka taulusta
- uusikin paikka voi olla varattu, tällöin jatketaan etsimistä edelleen
- ne paikat joihin avainta $k$ yritetään laittaa muodostavat $k$:n *kokeilujonon* (engl. probe sequence)
- järjestyksessään $j$:s kokeilu kohdistuu paikkaan
  $h(k, j) = (h'(k) + s(j, k)) \; mod \; m$
  missä $h'$ on "normaali" hajautusfunktio ja $s$ on kokeilufunktio
- vaaditaan että jokaiselle avaimelle $k$ kokeilujono
  $h(k, 0), h(k, 1), \ldots, h(k, m-1)$ muodostaa jonon $0, 1, \ldots, (m-1)$ permutaation, eli toisin sanoen kokeilujonon on kokeiltava jokaista hajautustaulun indeksiä kertaalleen
- avaimen $k$ lisääminen tapahtuu siten että ensin yritetään laittaa avain paikkaan $h(k, 0)$, jos tämä paikka on täysi, yritetään paikkaa $h(k, 1)$, jne
- koska kaikki avaimet talletetaan hajautustauluun, on talletettavien avaimien maksimimäärä $m$
- oletetaan että jos hajautustaulun paikka $T[i]$ on tyhjä, sillä on erityisarvo NIL

- ennen kuin katsotaan miten operaatiot toteutetaan avoimessa hajautuksessa, tutustumme yksinkertaisimpaan kokeilustrategiaan, eli *lineaariseen kokeiluun*
  - kokeilujono on nyt $h(k, j) = (h'(k) + j)\ mod\ m$
  - eli ensimmäinen paikka on $h'(k)\ mod\ m$, toinen $(h'(k) + 1)\ mod\ m$, kolmas $(h'(k) + 2)\ mod\ m$
  - esim: jos $m = 10$ ja $h'(k) = 7$ niin kokeilujono olisi $7, 8, 9, 0, 1, 2, 3, 4, 5, 6$
- avaimen lisääminen hajautustauluun

```
hash-insert(T,k)
1  i ←0
2  repeat
3      j ←h(k,i)
4      if T[j] = NIL then
5          T[j] ←k
6          return TRUE
7      i ←i+1
8  until i = m
9  return FALSE
```

- jos paikka alkiolle löytyy palauttaa operaatio TRUE, mutta jos mikään kokeilujonon paikka ei ole vapaana, eli hajautustaulu on jo täysi, palauttaa operaatio FALSE

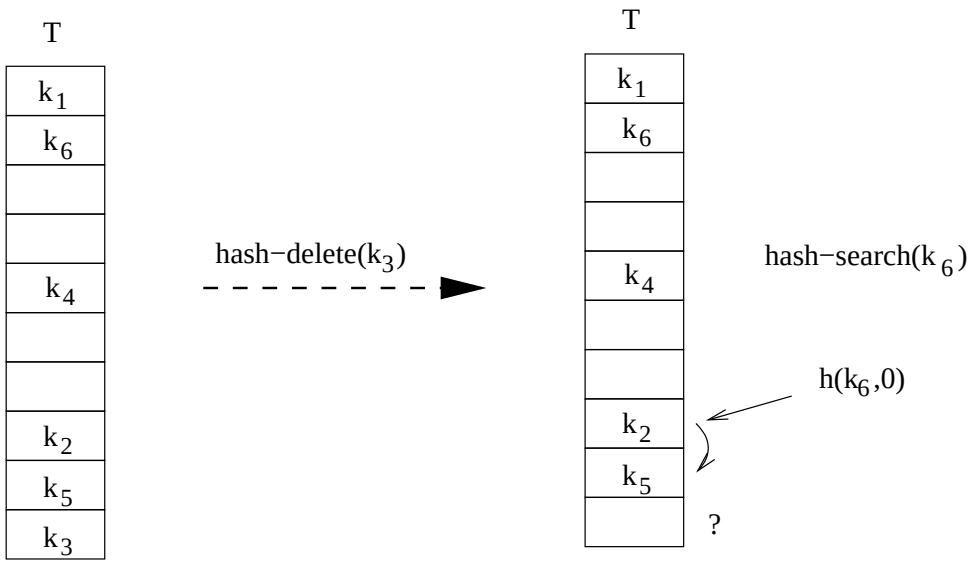



- avaimen etsiminen hajautustaulusta

```
hash-search(T,k)
1   i ←0
2   repeat
3       j ←h(k,i)
4       if T[j] = k then return j
5       i ←i+1
6   until T[j]=NIL or i=m
7   return NIL
```

- etsitään kokeilujonoa $h(k,0), h(k,1), \ldots, h(k,m-1)$ läpi niin kauan kun
  - etsitty avain löytyy,
  - törmätään tyhjään hajautustaulun paikkaan, tai
  - koko kokeilujono on käyty läpi
- avaimen löytyessä palautetaan avaimen tallettavan taulukon paikan indeksi, muuten NIL

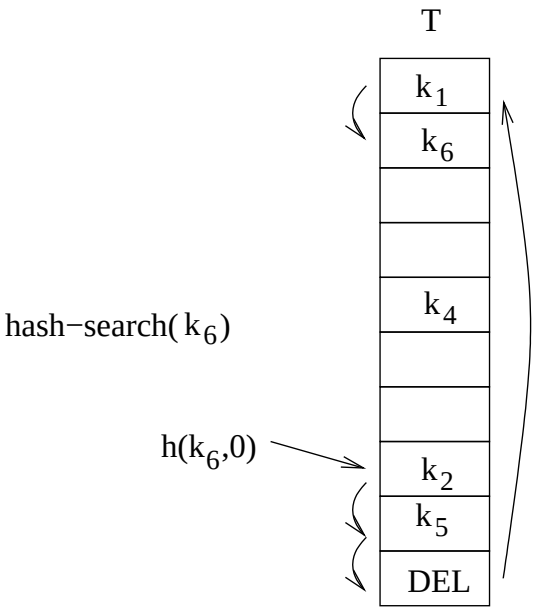
- avaimen poistamisen yhteydessä ei paikkaa voida jättää vapaaksi (NIL-arvoisiksi), muuten reitti joihinkin avaimiin saattaa katketa:




- merkitään poiston yhteydessä poistettavan alkion paikalle erityisarvo DEL

```
hash-delete(T,k)
1   i ←0
2   repeat
3       j ←h(k,i)
4       if T[j] = k then T[j] = DEL
5       i ←i+1
6   until T[j]=NIL or i=m
```

- nyt avaimen etsiminen toimii taas:




- muutetaan vielä lisäysoperaation riviä 5 siten että lisäys voidaan suorittaa taulukon paikkaan jonka arvo on NIL tai DEL

```
hash-insert(T,k)
1   i ←0
2   repeat
3       j ←h(k,i)
4       if T[j] = NIL or T[j]=DEL then
5           T[j] ←k
6           return TRUE
7       i ←i+1
8   until i = m
9   return FALSE
```

- avoimen hajautuksen poisto-operaatio on *laiska*, se ei vapauta muistialuetta, vaan jättää taulukkoon DEL arvoisia kohtia jotka täyttävät hajautustaulukkoa

- DEL kohdat voivat korvautua normaaleilla alkioilla insertin yhteydessä

- esim: oletetaan että käytössä hajautustaulukko jonka koko 11
- käytetään kokeilujonofunktiota $h(k,i) = ((k \; mod \; 11)+i) \; mod \; 11$ käytössä siis lineaarinen kokeilujono
- hajautetaan avaimet 75, 43, 21, 15, 18, 33, 30, 66 ja 92
- $h(75,0) = 9$ ja $h(43,0) = 10$, eli kahden ensimmäisen lisäyksen jälkeen

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | 75 | 43 |

- $h(21,0) = 10$ eli joudumme etsimään 21:lle uuden paikan $h(21,1) = ((21 \; mod \; 11) + 1) \; mod \; 11 = 0$ joka on vapaa.
- $h(15,0) = 4$ ja $h(18,0) = 7$, tilanne viiden ensimmäisen lisäyksen jälkeen:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 21 | | | | 15 | | | 18 | | 75 | 43 |

- $h(33,0) = 0$ joka jo varattu, eli etsintä jatkuu $h(33,1) = 1$. Seuraava avain voidaan sijoittaa heti kokeilujonon ensimmäiseen paikkaan $h(30,0) = 8$. Tilanne tässä vaiheessa:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 21 | 33 | | | 15 | | | 18 | 30 | 75 | 43 |

- $h(66,0) = 0$ ei käy, $h(66,1) = 1$ ei vieläkään, mutta $h(66,2) = 2$ onnistuu!
- Lopuksi $h(92,0) = 4$, ei käy, mutta $h(92,1) = 5$ vapaa

- tuloksena

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 21 | 33 | 66 | | 15 | 92 | | 18 | 30 | 75 | 43 |

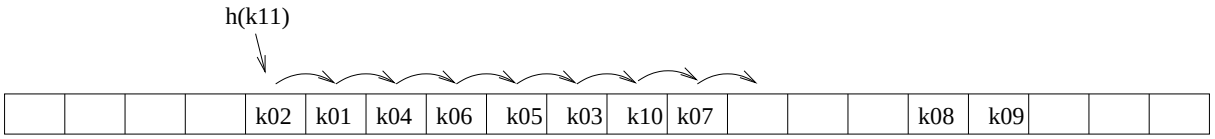
- entä jos haluamme vielä lisätä alkion 29? $h(29, 0) = 7$ varattu ja vapaa löytyy vasta 8. kokeilulla: $h(29, 7) = 3$!
- entä jos etsisimme taulukosta lukua 41? Etsintä etenisi paikasta $h(41, 0) = 7$ aina paikkaan $h(41, 7) = 3$ asti jonka jälkeen voidaan todeta että 41 ei ole taulukossa.
- poistetaan taulukosta alkiot 21, 30 ja 75:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| DEL | 33 | 66 | | 15 | 92 | | 18 | DEL | DEL | 43 |

- Edelleen, luvun 41 etsintä etenisi paikasta $h(41, 0) = 7$ aina paikkaan $h(41, 7) = 3$ asti jonka jälkeen voidaan todeta että 41 ei ole taulukossa
- luvun 29 lisäys löytäisi nyt 29:lle paikan toisella kokeilulla $h(29, 1) = 8$ sillä $T[8]$ =NIL
- taulukko luvun 29 lisäyksen jälkeen

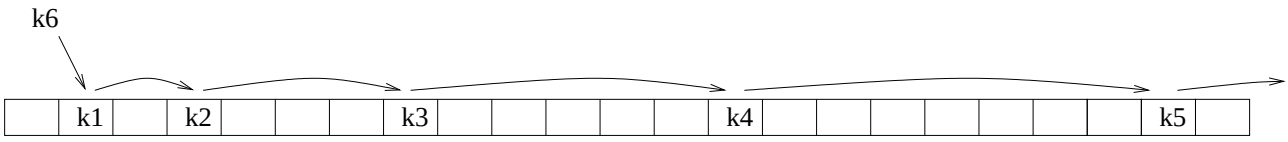
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| DEL | 33 | 66 | | 15 | 92 | | 18 | 29 | DEL | 43 |

- lineaarinen kokeilujono on helppo toteuttaa mutta käytännössä aika huono:
  - tauluun tulee usein pitkiä varattuja alueita
  - pitkät varatut alueet kasvavat suuremmalla todennäköisyydellä kuin lyhyet varatut osat:



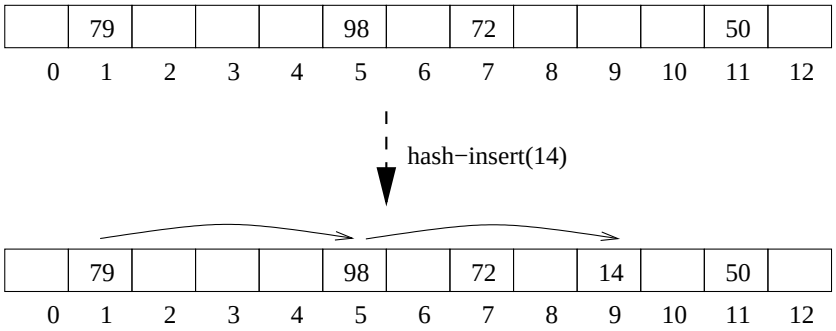

  - ilmiötä nimitetään *primääriseksi kasautumiseksi* (eng. primary clustering)

- *neliöinen kokeilu*
  - kokeilujono on nyt $h(k,i) = (h'(k) + c_1 \times i + c_2 \times i^2)\ mod\ m$, missä $h'$ normaali hajautusfunktio, $c_1$ ja $c_2$ vakioita
  - kokeilujonon ensimmäinen paikka on $h'(k)\ mod\ m$,
    toinen $(h'(k) + c_1 + c_2)\ mod\ m$,
    kolmas $(h'(k) + 2c_i + 4c_2)\ mod\ m$, jne
  - esim jos $c_1 = c_2 = 1$ ja $h'(k_1)$=5 niin kokeilujonon alku olisi $5, 7, 11, 17, 25, \ldots$, jos $m > 25$
  - primääriseltä kasaantumiselta vältytään, mutta nyt vaivana on *sekundäärinen kasautuminen*: jos $h(k_1, 0) = h(k_2, 0)$ niin $k_1$:n ja $k_2$:n kokeilujono on sama

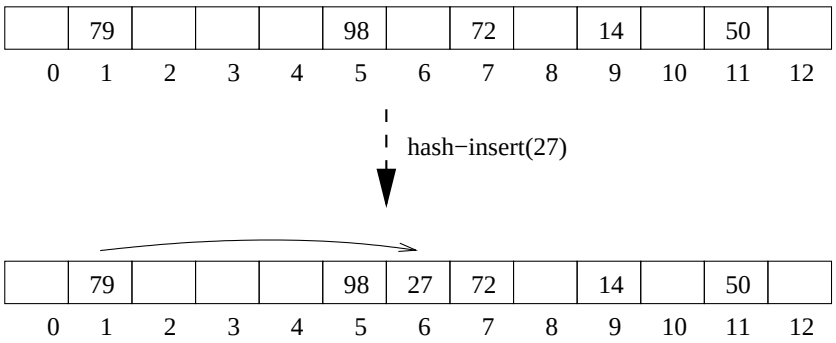



h(k1,0) = h(k2,0) = h(k3,0) = h(h4,0) = h(k5,0) = h(k6,0) = 1

- *kaksoishajautus*
  - kokeilujono on nyt $h(k,i) = (h'(k) + i \times h''(k))\ mod\ m$, missä $h'$ ja $h''$ normaaleja hajautusfunktioita
  - ensimmäinen paikka on $h'(k)\ mod\ m$, toinen $(h'(k)+h''(k))\ mod\ m$, kolmas $(h'(k)+2\times h''(k))\ mod\ m$, jne
  - esim. tarkastellaan 13 paikkaista hajautustaulua ja olkoot $h'(k) = k\ mod\ 13$ ja $h''(k) = 1 + (k\ mod\ 11)$
  - lisätään hajautustauluun avain 14
  - $h'(14) = 14\ mod\ 13 = 1$, $h''(14) = 1 + (14\ mod\ 11) = 4$,
  - kokeilujono on siis $1, 5, 9, 0, 4, 8, 12 \ldots$




  - lisätään hajautustauluun avain 27
  - $h'(27) = 27\ mod\ 13 = 1$, $h''(27) = 1 + (27\ mod\ 11) = 5$,
  - kokeilujono on siis $1, 6, 11, 3, 8, 0 \ldots$



- kaksoishajautus ei aiheuta kasaantumista ja onkin yleensä avoimen hajautuksen menetelmistä toimivin
- funktion $h''$ valinnassa on kuitenkin oltava huolellinen, arvoilla $h''(x)$ ja $m$ ei saa olla yhteisiä tekijöitä, muuten osa hajautustaulusta voi jäädä käymättä läpi
- yksi hyvä valinta on edellisessäkin esimerkissä esiintynyt tilanne missä $m$ on alkuluku ja $h''(x) < m$, eli valitaan esim
  $h'(k) = k \ mod \ m$ ja $h''(k) = 1 + (k \ mod \ m')$
  missä $m' < m - 1$, vaikkapa $m' = m - 2$

- avoimen hajautuksen tapauksessa kaikkien operaatioiden tehokkuus on riippuvainen avaimen löytymisen tehokkuudesta
- seuraavassa taulukossa tuloksettoman ja tuloksellisen haun *keskimääräinen* pituus kullakin kokeilujonotyypillä suhteessa täyttösuhteeseen $\alpha = n/m$

| | tulokseton | tuloksellinen |
|---|---|---|
| kaksoishajautus | $\frac{1}{1-\alpha}$ | $\frac{1}{\alpha}\ln\frac{1}{1-\alpha}$ |
| neliöinen | $\frac{1}{1-\alpha} - \alpha + \ln\frac{1}{1-\alpha}$ | $1 - \frac{2}{\alpha} + \ln\frac{1}{1-\alpha}$ |
| lineaarinen | $\frac{1}{2}(1 + \frac{1}{(1-\alpha)^2})$ | $\frac{1}{2}(1 + \frac{1}{1-\alpha})$ |

- tarkastellaan vielä hakujen keskimääräisiä pituuksia muutamalla konkreettisella täyttösuhteen arvolla

| $\alpha = 0.5$ | tulokseton | tuloksellinen |
|---|---|---|
| kaksoishajautus | 2 | 1.38 |
| neliöinen | 2.19 | 1.44 |
| lineaarinen | 2.5 | 1.5 |

| $\alpha = 0.75$ | tulokseton | tuloksellinen |
|---|---|---|
| kaksoishajautus | 4 | 1.85 |
| neliöinen | 4.63 | 2.01 |
| lineaarinen | 8.5 | 2.5 |

| $\alpha = 0.9$ | tulokseton | tuloksellinen |
|---|---|---|
| kaksoishajautus | 10 | 2.55 |
| neliöinen | 11 | 2.88 |
| lineaarinen | 50.5 | 5.5 |

| $\alpha = 0.95$ | tulokseton | tuloksellinen |
|---|---|---|
| kaksoishajautus | 20 | 3.15 |
| neliöinen | 22 | 3.53 |
| lineaarinen | 200.5 | 10.5 |

- näyttää siltä että täyttöasteeseen 0.5 asti suurta eroa menetelmien välillä ei ole
- tätä täydemmillä hajautustauluilla lineaarista kokeilujonoa ei kannata käyttää

- yhteentörmäysten ketjutusratkaisussa hajautusrakenne toimii hajautusalueen täytyttyäkin
- avoimessa hajautuksessa lisäys täyteen hajautusrakenteeseen on mahdotonta

- hajautusrakenteen täyttymisongelma voidaan ratkaista *uudelleenhajautuksella*
  - täyteen rakenteeseen tehtävän lisäyksen yhteydessä rakenteelle varataan uusi kooltaan kaksinkertainen tila
  - kaikki alkuperäisen rakenteen avaimet talletetaan uuteen hajautusrakenteeseen käyttäen uutta hajautusfunktiota
  - uuden hajautusrakenteen täyttösuhde on 1/2
  - jos poiston jälkeen rakenteen täyttösuhde on alle 1/4 voidaan tilaa vapauttaa varaamalla uusi rakenne, kooltaan puolet alkuperäisestä
  - tässäkin tapauksessa kaikki alkuperäisen rakenteen avaimet talletetaan uuteen hajautusrakenteeseen käyttäen uutta hajautusfunktiota
  - uuden hajautusrakenteen täyttösuhde on 1/2
  - molemmissa tapauksissa uudelleenhajautuksen aikavaatimus on $\mathcal{O}(m)$, missä $m$ alkuperäisen hajautusrakenteen koko

## 4.4 Käytännöllisiä huomautuksia hajautuksesta

- hajautusfunktioista jakojäännösmenetelmä on yleensä hyvä
- usein aineiston kanssa kannattaa tehdä muutamia kokeita ennen kuin käytettävä hajautusfunktio valitaan
- yhteentörmäysstrategioista ketjutus on yleensä paras ratkaisu
  - ketjutus toimii hyvin vaikka hajautusalue onkin täynnä
  - poistot ketjutuksessa ovat aitoja toisin kuin avoimessa hajautuksessa
- avoin hajautus on hyvä jos täyttösuhde säilyy pienenä
- hajautus on yleensä paras menetelmä joukko-operaatioiden toteuttamiseen jos operaatioita min, succ, max ja pred ei tarvita
- operaatioiden min, succ, max ja pred toteuttaminen hajautusrakenteissa on toivottoman hidasta eli jos näitä operaatioita tarvitaan, kannattaa käyttää tasapainoitettua puuta