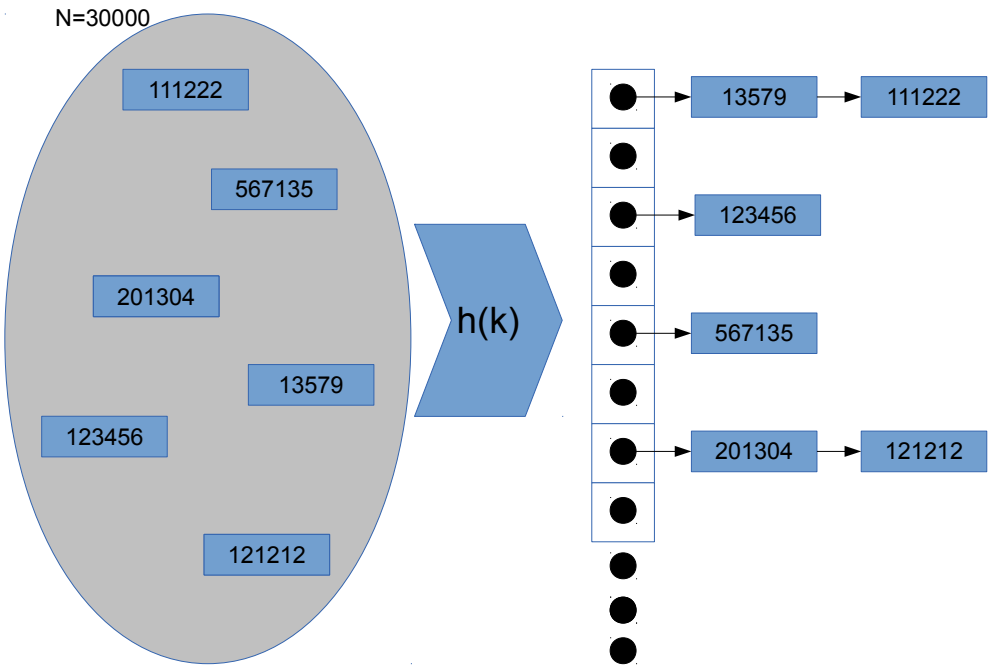
11.1 Hajautustaulu

Hajautustaulun ideana on tiivistää dynaamisen joukon avainten arvoalue pienemmäksi *hajautusfunktion* (*hash function*) h avulla, siten että ne voidaan tallettaa taulukkoon.

- taulukon etuna on sen tarjoama tehokas vakioaikainen indeksointi



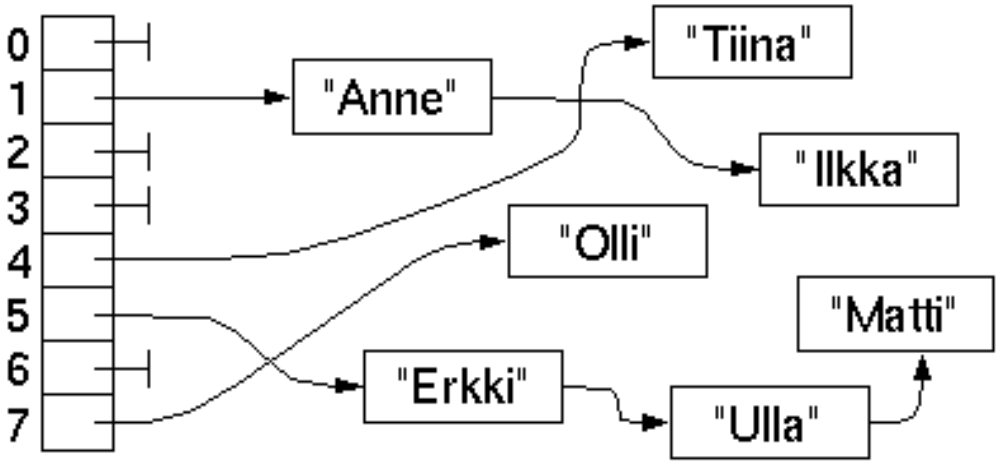
Avainten arvoalueen tiivistämisestä seuraa kuitenkin ongelma: *törmäykset*.

- useampi kuin yksi alkio voi hajautua samaan hajautustaulun lokeroon

Tavallisin tapa ratkaista ongelma on *ketjuttaminen (chaining)*.

- samaan lokeroon hajautuvat alkiot talletetaan listoihin
- muitakin ratkaisutapoja on
  - *avoimen osoituksen* käsittelytavalla alkio laitetaan sekundääriseen lokeroon, mikäli primäärinen ei ole vapaana
  - joissakin tapauksissa avainten arvoalue on niin pieni, että arvoalueen tiivistämistä ei tarvita, eikä siis synny törmäyksiäkään
    - tällainen *suoraosoitustaulu (direct-access table)* on hyvin yksinkertainen ja tehokas
  - tällä kurssilla kuitenkin käsitellään ainoastaan ketjutettuja hajautustauluja

Alla oleva kuva esittää ketjutettua hajautustaulua, jonka avaimet on hajautettu etukirjaimen mukaan viereisen taulukon avulla.



$h(k)$	alkukirjain			
0		H	P	X
1	A	I	Q	Y
2	B	J	R	Z
3	C	K	S	Ä
4	D	L	T	Ö
5	E	M	U	Å
6	F	N	V	
7	G	O	W	

Onko tämä hyvä hajautus?

- Ei. Katsotaan seuraavaksi, miksei.

Ketjutettu hajautustaulu tarjoaa ainoastaan *sanakirjan* operaatiot, mutta ne ovat hyvin yksinkertaisia:

CHAINED-HASH-SEARCH(T, k)
▷ etsi listasta $T[h(k)]$ alkio, jonka avain on k

CHAINED-HASH-INSERT(T, x)
▷ lisää x listan $T[h(x{\rightarrow}key)]$ alkuun

CHAINED-HASH-DELETE(T, x)
▷ poista x listasta $T[h(x{\rightarrow}key)]$

Suoritusajat:

- lisäys: $\Theta(1)$
- etsintä: hitaimmassa tapauksessa $\Theta(n)$
- poisto: jos lista kaksisuuntainen, niin $\Theta(1)$; yksisuuntaisella hitaimmillaan $\Theta(n)$, koska poistettavan edeltäjä on ensin etsittävä listasta
  - käytännössä ero ei kuitenkaan ole kovin merkittävä, koska yleensä poistettava alkio joudutaan joka tapauksessa etsimään listasta

Ketjutetun hajautustaulun operaatioiden *keskimääräiset* suoritusajat riippuvat listojen pituuksista.

- huonoimmassa tapauksessa kaikki alkiot joutuvat samaan listaan jolloin suoritusajat ovat $\Theta(n)$
- keskimääräisen tapauksen selville saamiseksi käytämme seuraavia merkintöjä:
  - m = hajautustaulun koko
  - n = alkioiden määrä taulussa
  - $\alpha = \frac{n}{m}$ = *täyttöaste* (*load factor*) eli listan keskimääräinen pituus
- lisäksi keskimääräisen suoritusajan arvioimiseksi on tehtävä oletus siitä, miten hyvin h hajauttaa alkiot
  - jos esim. $h(k)$ = nimen alkukirjaimen 3 ylintä bittiä, niin kaikki osuvat samaan listaan
  - usein oletetaan että, jokaisella alkiolla on yhtä suuri todennäköisyys osua mihin tahansa lokeroon
  - *tasainen hajautus* (*simple uniform hashing*)
  - oletetaan myös, että $h(k)$:n laskenta kuluttaa $\Theta(1)$ aikaa

- jos etsitään alkiota, jota ei ole taulussa, niin joudutaan selaamaan koko lista läpi
  $\Rightarrow$ joudutaan tutkimaan keskimäärin α alkiota
  $\Rightarrow$ suoritusaika keskimäärin $\Theta(1+\alpha)$
- jos oletetaan, että listassa oleva avain on mikä tahansa listan alkio samalla todennäköisyydellä, joudutaan listaa selamaan etsinnän yhteydessä keskimäärin puoleen väliin siinäkin tapauksessa, että avain löytyy listasta
  $\Rightarrow$ suoritusaika keskimäärin $\Theta(1+\frac{\alpha}{2}) = \Theta(1+\alpha)$
- jos täyttöaste pidetään alle jonkin kiinteän rajan (esim. $\alpha <$ 50 %), niin $\Theta(1+\alpha) = \Theta(1)$
  $\Rightarrow$ ketjutetun hajautustaulun kaikki operaatiot voi toteuttaa keskimäärin ajassa $\Theta(1)$
  - tämä edellyttää, että hajautustaulun koko on samaa luokkaa kuin sinne talletettavien alkioiden määrä

Laskiessamme keskimääräistä suoritusaikaa oletimme, että hajautusfunktio hajauttaa täydellisesti. Ei kuitenkaan ole mitenkään itsestään selvää, että näin tapahtuu.

Hajautusfunktion laatu on kriittisin tekijä hajautustaulun suorituskyvyn muodostumisessa.

Hyvän hajautusfunktion ominaisuuksia:

- hajautusfunktion on oltava deterministinen
  - muutoin kerran tauluun pantua ei välttämättä enää koskaan löydetä!
- tästä huolimatta olisi hyvä, että hajautusfunktion arvo olisi mahdollisimman "satunnainen"
  - kuhunkin lokeroon tulisi osua mahdollisimman tarkasti $\frac{1}{m}$ avaimista

- valitettavasti täysin tasaisesti hajottavan hajautusfunktion teko on useinmiten mahdotonta
  - eri arvojen esiintymistodennäköisyydet aineistossa ovat yleensä tuntemattomia
  - aineisto ei yleensä ole tasaisesti jakautunut
    - lähes mikä tahansa järkevä hajautusfunktio jakaa tasaisesti jakautuneen aineiston täydellisesti
- yleensä hajautusfunktio pyritään muodostamaan siten, että se sotkee tehokkaasti kaikki syöteaineistossa luultavasti esiintyvät säännönmukaisuudet
  - esimerkiksi nimien tapauksessa ei katsota yksittäisiä kirjaimia, vaan otetaan jotenkin huomioon nimen kaikki bitit

- esittelemme kaksi yleistä usein hyvin toimivaa hajautusfunktion luontimenetelmää
- oletamme, että avaimet ovat luonnollisia lukuja 0, 1, 2, . . .
  - jollei näin ole, avain voidaan usein tulkita luonnolliseksi luvuksi
  - esim. nimen saa luvuksi muuttamalla kirjaimet numeroiksi ASCII-koodiarvon mukaan, ja laskemalla ne sopivasti painottaen yhteen

Hajautusfunktion luonti *jakomenetelmällä* on yksinkertaista ja nopeaa.

- $h(k) = k \mod m$
- sitä kannattaa kuitenkin käyttää vain, jos m:n arvo on sopiva
- esim. jos $m = 2^b$ jollekin $b \in N = \{0, 1, 2, \ldots\}$, niin

$$h(k) = k\text{:n } b \text{ alinta bittiä}$$

$\Rightarrow$ funktio ei edes katso kaikkia k:n bittejä
$\Rightarrow$ funktio todennäköisesti hajauttaa huonosti, jos avaimet ovat peräisin binäärijärjestelmästä

- samasta syystä tulee välttää m:n arvoja muotoa $m = 10^b$, jos avaimet ovat peräisin kymmenjärjestelmän luvuista
- jos avaimet ovat muodostetut tulkitsemalla merkkijono 128-järjestelmän luvuksi, niin m = 127 on huono valinta, koska silloin saman merkkijonon kaikki permutaatiot osuvat samaan lokeroon
- hyviä m:n arvoja ovat yleensä alkuluvut, jotka eivät ole lähellä 2:n potensseja
  - esim. halutaan $\approx$ 700 listaa $\Rightarrow$ 701 kelpaa
- kannattaa tarkistaa kokeilemalla pienellä "oikealla" aineistolla, hajauttaako funktio avaimet tehokkaasti

Hajautusfunktion luonti *kertomenetelmällä* ei aseta suuria vaatimuksia m:n arvolle.

- valitaan vakio A siten, että $0 < A < 1$
- $h(k) = \lfloor m(kA - \lfloor kA \rfloor) \rfloor$
- jos $m = 2^b$, koneen sanapituus on w, ja k ja $2^w \cdot A$ mahtuvat yhteen sanaan, niin $h(k)$ voidaan laskea helposti seuraavasti:

$$h(k) = \lfloor \frac{(((2^w \cdot A) \cdot k) \mod 2^w)}{2^{w-b}} \rfloor$$

- mikä arvo A:lle tulisi valita?
  - kaikki A:n arvot toimivat ainakin jollain lailla
  - kuulemma $A \approx \frac{\sqrt{5}-1}{2}$ toimii usein aika hyvin