Hashing


CHOICE
HASH
ASSOCIATIVE
RESIZE
OPERATION
CLUSTERING
RECORD
PERFORMANCE
SMALL
TABLE
DBM
POINTER
LOOKUP
HASH
ENTRY
GOOD
COLLISION
WORST
DELETION
UNIFORM
ARBITRARY
TABLE
LOAD
PROPORTIONAL
MEMORY
SPACE
DISTRIBUTION
NEIGHBORHOOD
AMORTIZED
LINEAR
DEGRADES
AVERAGE
STRUCTURE
ADDRESS
UNORDERED
HOPSCOTCH
PROBE
OPENARRAY
INTERNING
ALLOCATOR
COLLIDED
CUCKOO
INSERTION
IMPLEMENTATION
CONSTANT
REHASH
SLOT
INTERVAL

O(log *n*) - søk

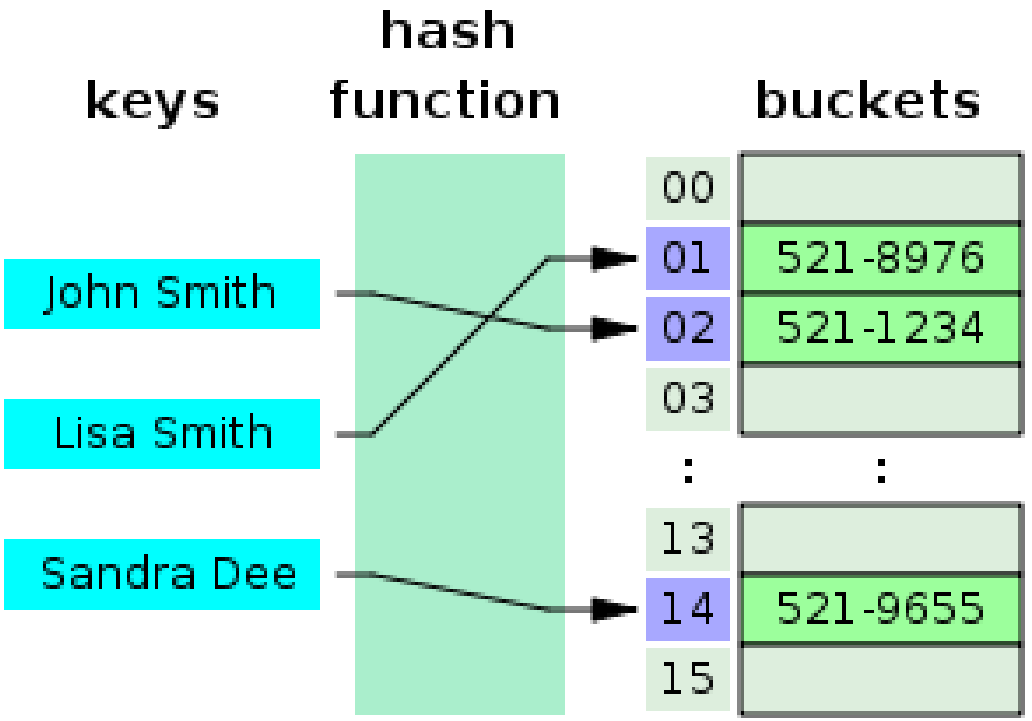
- Søking i et balansert søketre med *n* elementer er alltid *O*(log *n*)
- Søkingen er basert på *parvis*e sammenligninger av to og to verdier
- Er svært raskt uansett hvor stort søketreet er, fordi logaritmen vokser meget langsomt...
- ...men vil allikevel gi lengre søketider for voksende *n*
- Kan vi klare å lage en datastruktur som har samme effektivitet uansett hvor stor *n* er?

O(1) – søking?

- *O*(1) - søking: Uavhengig av antall elementer
- Finn verdien med bare *ett* direkte oppslag i datastrukturen
- Er praktisk mulig bare hvis vi har en *nummerering* av alle mulige elementer som kan lagres i datastrukturen, og vi *vet* hvor hvert element ligger
- Eksempel: Data om alle personer i Norge, lagret i en array der *indeksen* til data om hver person er *personnummeret*

Hashing: Et forsøk på *O*(1) – effektivitet

- Alle dataene lagres i en lang array med «nok plass», en *hashtabell* (aka «buckets»)
- Hvert element som lagres har en *nøkkelverdi*
- Ut i fra nøkkelverdien *beregnes* hvilken *indeks i hashtabellen* som et element skal ligge på
- Indeksen beregnes med en *O*(1) *hashfunksjon*
- Indeksen som beregnes kalles en *hashverdi*
- Hvis alle hashverdier som beregnes er *ulike*, har alle elementer en *unik* indeks – søking blir *O*(1) !


keys
hash function
buckets
John Smith
Lisa Smith
Sandra Dee
00
01
02
03
13
14
15
521-8976
521-1234
521-9655

Typiske anvendelser av hashing

- Databasesystemer
- Minnehåndtering i operativsystemer
- Håndtering av variabler og metoder i kompilatorer
- Rask gjenfinning av grafikkelementer i 3D-spill
- Stavekontroll i editorer og tekstbehandlere:
  - Riktig stavede ord kan legges i en hashtabell i stedet for å sorteres alfabetisk i et søketre

Hashing: Eksempel

- Skal lage et register for maksimalt 1000 objekter
- Unike nøkkelverdier: Syvsifret tall, f.eks. 4618996
- Kan bruke en array med 10 mill. elementer, men...
- Bruker en array med lengde 1000 (hashlengden)
- Tre siste sifre i nøkkelverdi brukes som hashverdi:

  `hash(key) = key % 1000`

  `hash(4618996) = 996` (lagres på indeks 996)
- Og da er alt i orden og vi har en *O*(1) struktur?

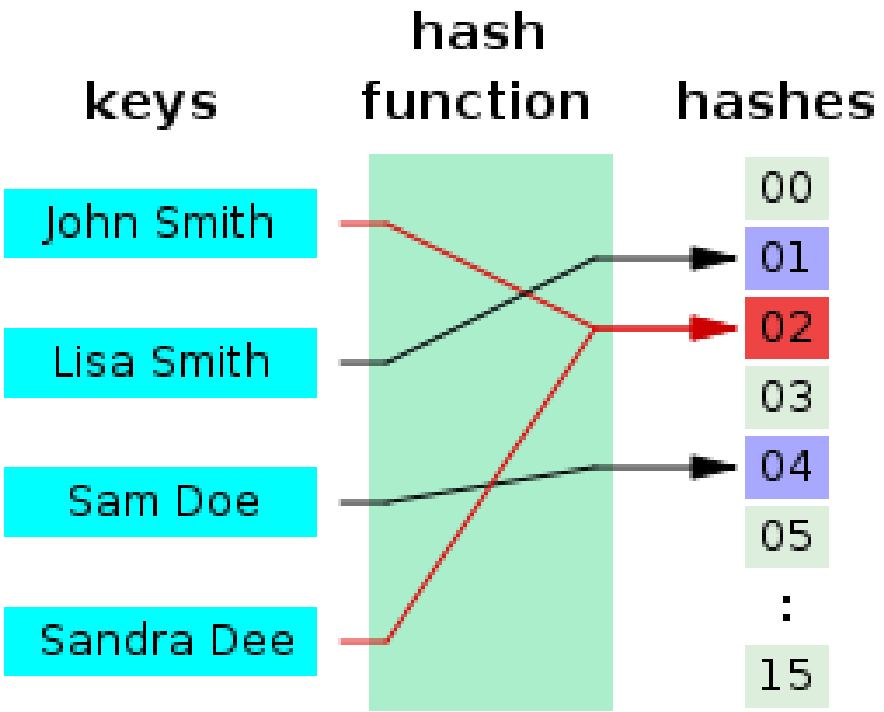
Problem: Kollisjoner

- Kollisjon*:
  - To elementer får samme indeks
- Eksempel:

```
hash(key)      = key % 1000
hash(6894331) = 331
hash(7462331) = 331
```

- Hashing kan bli $O(n)$ hvis det er mange kollisjoner i hashtabellen!

*: Aka «hash-clash»


keys
hash function
hashes
John Smith
Lisa Smith
Sam Doe
Sandra Dee
00
01
02
03
04
05
:
15

Og kollisjoner skjer «hele tiden»

- «En ulykke skjer sjelden alene»
- «Alt» har en tendens til å opptre i klynger/clustere:
  - Byer, bilkøer, industriklynger, sosiale samlinger
  - Fornavn
  - Maurtuer, fiskestimer, gresshoppesvermer
  - Galakser, stjernehoper
- Hvorfor det *alltid* er slik vet vi egentlig ikke, men:
  - Det skal «veldig lite til» før ting begynner å kollidere
  - Klassisk eksempel: Fødselsdagsparadokset

Hashfunksjoner, kollisjoner og effektivitet

- Det er *alltid* mange kollisjoner i hashing med store datamengder
- For at hashing skal være effektivt:
  - Hashfunksjonen som brukes må gi et lite antall kollisjoner og spre dataene godt i hashtabellen
  - Kollisjoner må håndteres så effektivt som mulig
- Hvis disse to kravene tilfredsstilles, kan hashing være mer effektivt enn både søketrær og B-trær for svært store datamengder