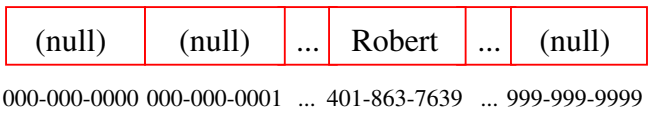
Hashing

Til bruk for ordbok

God FORVENTET kjøretid, men VERSTE FALL er dårlig

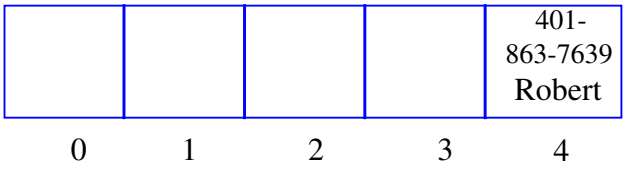
Problemstilling

- Et internasjonalt tele-selskap ønsker å gi sine telefonkunder mulighet til:
  - gitt et telefon-nummer, returner eier
  - telefon-nummere mellom 0 og $R = 10^{10}-1$
  - Anta n er antallet tele-nummere lagret
  - dette må gjøres effektivt
- Bruk ordbok!
  - et ***balansert søketre*** (AVL, red-black) med tele-nummere som nøkkel har O(log n) søketid og bruker O(n) minne --- dette er bra, men kan vi få konstant søketid?
  - en ***bucket array*** indeksert ved tele-nummere har i prinsippet O(1) søketid, men minnebruken er enorm: O($n + R$)



Ny løsning

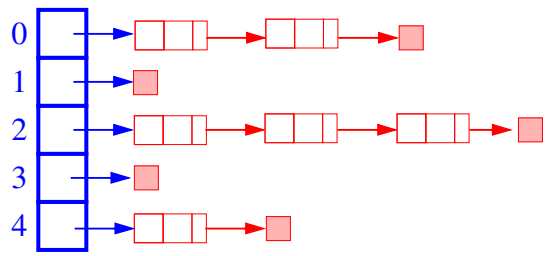
- En ***HashTabell*** er en alternativ løsning med O(1) forventet søketid og O($n + N$) minnebruk, hvor N er valgt størrelse på hashtabell.
- Som array, med tillegg av en funksjon som sender tele-nummere til indeks i tabell
  - f.eks. indeks for tele-nummer X er (X MOD N)
- InsertItem (401-863-7639, Robert) i tabell av størrelse 5:
  - 4018637639 mod 5 = 4, dvs (401-863-7639, Robert) lagres i indeks 4 i hash-tabellen



- Et søk bruker samme prossess: send tele-nummer til indeks, og sjekk hash-tabell i den indeks.
- InsertItem (401-863-9350, Arne). OK
- InsertItem (401-863-2234, Ole). Vi får ***kollisjon***!

Kollisjons-håndtering

- Hva gjør vi med to nøkler som sendes til samme indeks av hash-funksjonen?
- Anvend ***kjeding***
  - Lagre en lenket ***liste*** for nøkler med samme indeks



- Forventet, søk/innsetting/fjerning blir O(n/N), dersom de anvendte indeksene er uniformt distribuert
- Dvs, forventet kjøretid for hash-tabellen avhenger av valg av hash-funksjon og av tabellstørrelsen N.
- N velges noe større enn forventet n, f.eks. n/N=0.75. Re-hashing om nødvendig.

Fra nøkkel til indeks

- ***hash funksjon*** angir avbildning av nøkler til indeks
- NB: Nøkler er ikke nødvendigvis heltall!
- Hash-funksjon består ofte av 2 deler:
  - ***hash-kode***: key → integer
  - ***kompresjon***: integer → $[0, N-1]$
- Dvs indeks(key)=kompresjon(hash-kode(key))
- Krav at dette faktisk er en funksjon, dvs to like nøkler må få samme indeks.
- En "god" hash funksjon minimerer sannsynligheten for kollisjoner

Eksempler på Hash-funksjoner

- ***Hash-kode:*** Om nøkkel ikke er heltall, men f.eks en streng, må vi først anvende hash-kode
  - F.eks. hash-kode(key)=sum av ASCII-koden til alle tegn i key
- **Kompresjon: fra heltall til indeks**: F.eks. Multiply, Add and Divide (MAD): Bruk primtall N, og to tall a,b > 0, og la
  - indeks(key)=(a*key+b) mod N, dvs
  - hash-kode(key)=a*key+b og
  - kompresjon(int)=int mod N
  - MAD brukes også for visse random-generatorer.
- I Java har alle Object en metode hashCode(), men denne returnerer ofte kun 32-bits minneaddresse for objektet.
- hashCode() fungerer derfor ikke for Integer eller String objekter. Hvorfor ikke?
- Dvs, hver enkelt klasse må overskrive hashCode() på en fornuftig måte.

Mer om Kollisjoner

- Hva gjør vi om flere nøkler sendes til samme indeks:
- Vi har sett ***Kjeding***
- Kan også bruke 2 løsninger innen det som kalles ***Åpen Addressering:***
  - ***Double Hashing***
  - ***Lineære Forsøk*** (eng: Linear Probing)

Linear Probing

- Om indeksen er opptatt, så forsøk neste indeks

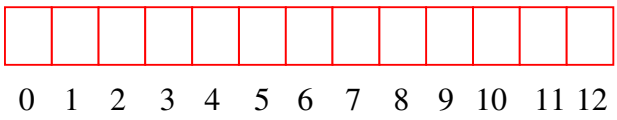
```
linear_probing_insert(K)
  if (table is full) error

  indeks = hash(K)

  while (table[indeks] opptatt)
        indeks = (indeks + 1) mod N

  table[probe] = K
```

- Et søk må gå bortover i tabellen til nøkkelen blir funnet, eller til en tom posisjon finnes
- Mindre minnebruk enn kjeding, ingen pekere
- Saktere enn kjeding
  - kan gå langt i tabellen
- Fjerning mer komplisert
  - kan f.eks. merke posisjon med gitt indeks som fjernet. Hva da med søk?
  - eller fylle denne ved å flytte verdier nedover i tabellen

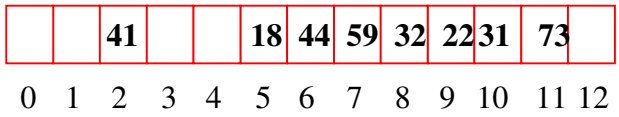
Linear Probing Eksempel

- $h(k) = k \bmod 13$
- Innsett:

18 41 22 44 59 32 31 73



Linear Probing Eksempel



Double Hashing

- Bruk 2 hash-funksjoner h1 og h2
- Om N er et primtall, vil alle indekser forsøkes

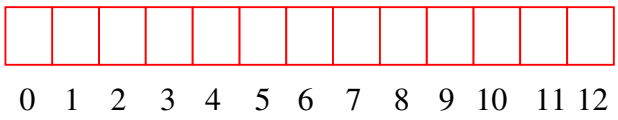
```
double_hash_insert(K)
  if(table is full) error

  indeks = h1(K)
  offset = h2(K)

  while (table[indeks] opptatt)
    indeks = (indeks + offset) mod N

  table[indeks] = K
```

- Analysen likner på linear probing
- Distribuerer nøkler mer uniformt enn linear probing.

Double Hashing Eksempel

- h1(K) = K mod 13
  h2(K) = 8 - K mod 8
  - siden h2 skal adderes

18 41 22 44 59 32 31 73



Double Hashing Eksempel

0	1	2	3	4	5	6	7	8	9	10	11	12
44		**41**	**73**		**18**	**32**	**59**	**31**	**22**			

Teoretisk analyse

- La $\alpha = n/N$
- Dette er gjennomsnittlig antall nøkler lagret i hver indeks
- Analysen er probabilistisk

Forventet antall forsøk

	ikke funnet	*funnet*
Kjeding	$1 + \alpha$	$1 + \frac{\alpha}{2}$
Linear Probing	$\frac{1}{2} + \frac{1}{2(1-\alpha)^2}$	$\frac{1}{2} + \frac{1}{2(1-\alpha)}$
Double Hashing	$\frac{1}{(1-\alpha)}$	$\frac{1}{\alpha} ln \frac{1}{1-\alpha}$

Forventet antall forsøk versus n/N