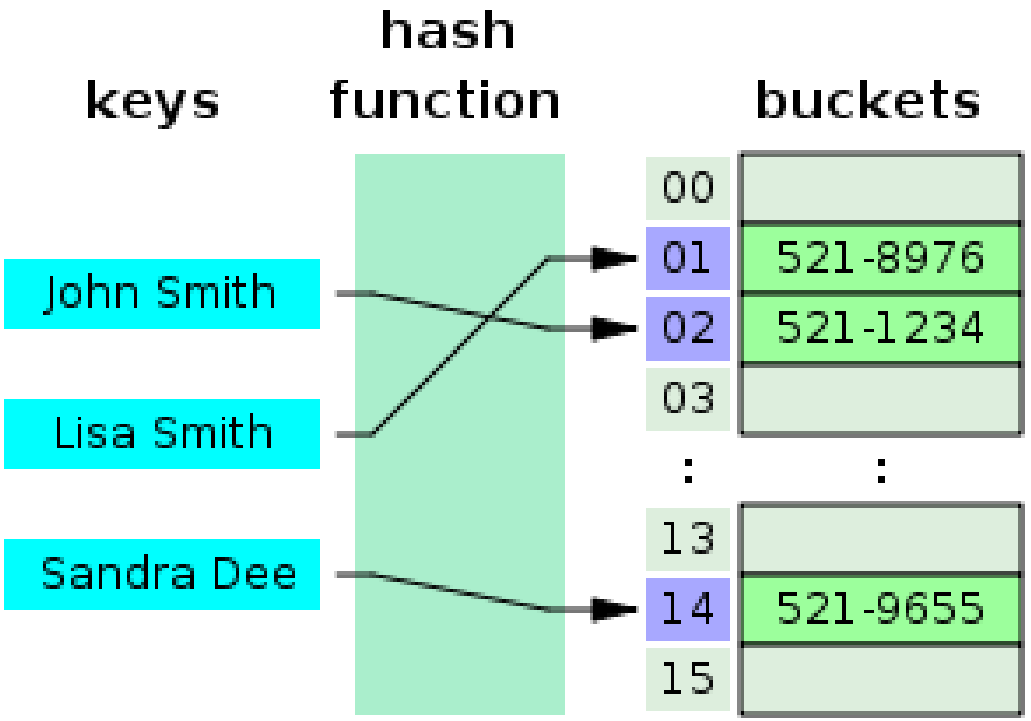
Hashfunksjoner



Hashfunksjoner

- Hashfunksjonen beregner en *indeks* i hashtabellen basert på *nøkkelverdien* som vi søker etter
- Hash: «Kutte opp i biter og blande sammen»
- Perfekt hashfunksjon:
  - Lager aldri kollisjoner
  - Alle elementer får en unik indeks
  - Kan bare lages i tilfeller der alle data er kjent

Effektive hashfunksjoner

- Krav til en effektiv hashfunksjon:
  - Beregning av hashverdien må være rask og $O(1)$
  - Sprer hashverdiene jevnt i hashtabellen
  - Gir et lite antall kollisjoner
- Utvikling av effektive hashfunksjoner er ikke en «eksakt vitenskap»:
  - Antall kollisjoner avhenger både av datasettet *og* av lengden på hashtabellen
  - Baserer seg i stor grad på *heuristikk* og *empiri*

Hashfunksjoner og hashlengde

- Hashlengde: Antall elementer i hashtabellen
- Hashfunksjonen beregner en indeks i hashtabellen:
  - Verdien som returneres må være større eller lik 0 (null) og mindre enn hashlengden
  - Beregner en verdi *h* basert på nøkkelverdien
  - Returnerer resten ved heltallsdivisjon av *h* med hashlengden: *h % hashlengde*
- Vi vil alltid få bedre spredning og færre kollisjoner hvis hashlengden er et primtall:
  - F.eks. er 997 og 1009 bedre hashlengder enn 1000

Noen typer hashfunksjoner

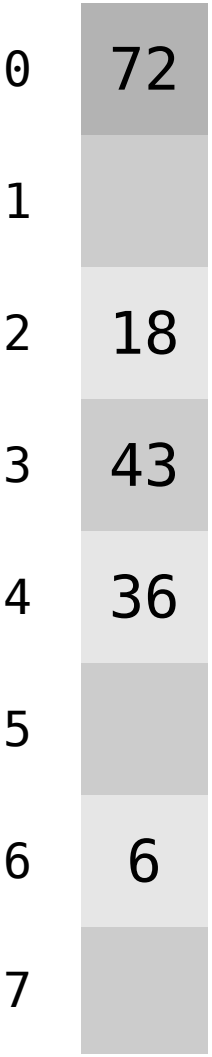
- Avkorting
- Sammenslåing / Folding
- Midten-av-kvadratet
- Bytte av tallsystem
- Utplukk og ombytting
- Basert på lengde av nøkkelverdi

Hashfunksjon: Avkorting (truncation)

- «Klipper» bare ut en del av nøkkelverdien
- Eksempel, nøkkelverdi er en streng:
  - Bruk de *k* første bokstavene, tolket som siffer
- Eksempel, nøkkelverdi er et heltall:
  - Bruk de *k* siste sifferne
  - Finnes med heltallsdivisjon: $\textit{nøkkelverdi} \% 10^k$
- Fordel: Rask beregning av hashverdi
- Ulempe: Fordeler ujevnt, gir mye kollisjoner i tabellen

Eksempel: Hashing med enkel avkorting

```
hashlengde = 8
hash(key)  = key % hashlengde
hash(36)   = 36 % 8 = 4
hash(18)   = 18 % 8 = 2
hash(72)   = 72 % 8 = 0
hash(43)   = 43 % 8 = 3
hash(6)    =  6 % 8 = 6
```



Hashfunksjon: Sammenslåing / Folding

- Del opp nøkkelverdien i flere «småbiter»
- Slå sammen «småbitene», med f.eks. aritmetiske operasjoner, til en hashverdi
- Eksempel:
  - Nøkkelverdi: 625381194
  - Lengde av hashtabell (hashlengde) 1000
  - Hashverdi: (625 + 381 + 194) % 1000 = 100
- Folding sprer bedre enn avkorting

Kode: Enkel folding av tekststrenger

```
int hash(String S)
{
  int h = 0;
  for (int i = 0; i < S.length(); i++)
    h += (int)(S.charAt(i));
  return h % hashLengde;
}
```

Noen verdier med `hashLengde = 1009`:

```
hash("VW Karmann Ghia")   = 317
hash("Porsche 356A")      = 979
hash("Alfa Romeo Spider") = 556
hash("Renault Floride")   = 463
```

Problem: Enkel folding sprer dårlig

- I eksemplet på forrige side vil de fleste hashverdiene havne i et relativt lite intervall – noen hundre opp til et par tusen for tekst-strenger av begrenset lengde
- Ubrukelig for store datasett med millioner av data
- For å spre bedre kan *vektet folding* brukes:
  - Gi tegnene en *vekt* basert på posisjon
  - Tolker strengen noe tilsvarende som et desimalt tall, der hvert tegn er et siffer som angir antallet av en potens av 10

Eksempel: Vektet folding av tekststrenger

- Tolk f.eks. 4 og 4 tegn i strengen som «desimale tall»:

```
VW_Karmann_Ghia
```

- Beregn:

```
((V + W*10 + _*100 + K*1000) +
 (a + r*10 + m*100 + a*1000) +
 (n + n*10 + _*100 + G*1000) +
 (h + i*10 + a*100)) % hashLengde
```

- Gir mye større spenn i verdiene som beregnes

Kode: Vektet folding av tekststrenger

```
int hash(String S)
{
  int h = 0, i = 0;
  while (i < S.length())
  {
    int potens = 1;
    for (int j = 0; (j < 4 && i < S.length()); j++)
    {
       h += (int)(S.charAt(i)) * potens;
       potens *= 10;
       i++;
    }
  }
  return h % hashLengde;
}
```

Javas egen hashfunksjon for strenger

hashCode

`public int hashCode()`

Returns a hash code for this string. The hash code for a `String` object is computed as

```
s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]
```

using `int` arithmetic, where `s[i]` is the *i*th character of the string, `n` is the lengt of the string, and ^ indicates exponentiation. (The hash value of the empty string is zero.)

- Merk: `String.hashCode()` ignorerer overflow og kan returnere negative verdier(!)

Hashfunksjon: Midten-av-kvadratet

- Anta nøkkelverdien er et heltall*
- Beregn kvadratet av nøkkelverdien
- Hashverdi: Siffere i midten av kvadratet
- Eksempel:
  - Bruker hashverdier med tre sifre (000 - 999)
  - Nøkkelverdi: 18562
  - 18562 · 18562 = 344547844
  - hash(18562) = 547

*: For ikke-numeriske nøkkelverdier kan vi f.eks. bare *tolke* bitsekvensene som tall

Kode: Midten-av-kvadratet for strenger

```
int hash(String S)
{
   // Kvadrerer Javas hashverdi for strengen
   long h = S.hashCode());
   h *= h;

   // Tar vekk første og siste siffer
   String tall = String.valueOf(h);
   h = Long.parseLong(tall.
                      substring(1,tall.length()-1));

   return (int) (h % hashLengde);
}
```

Hashfunksjon: Bytte av tallsystem*

- Antar nøkkelverdi er et desimalt heltall:
  - Grunntallet er 10, sifferne er 0, 1, 2, 3, ..., 8, 9
  - $1369 = 1369_{10} = 9 + 6 \cdot 10^1 + 3 \cdot 10^2 + 1 \cdot 10^3$
- Skriver om nøkkelverdien til f.eks. 8-tallsystemet:
  - Grunntallet er 8, sifferne er 0, 1, 2, 3, 4, 5, 6, 7
  - $2531_8 = 1 + 3 \cdot 8^1 + 5 \cdot 8^2 + 2 \cdot 8^3 = 1369_{10}$
- Hvis hashlengden er 1009:

  hash(1369) = 2531 % 1009 = 513

*: Det finnes flere metoder for bytte av tallsystemer i Java, se f.eks. Integer.toOctalString

Kode: Bytte av tallsystem for strenger

```
int hash(String S)
{
    // Beregner Javas hashverdi for strengen
    long h = (long) Math.abs(S.hashCode());

    // Konverterer til oktalt tall
    h = Long.parseLong(Long.toOctalString(h));

    return (int) (h % hashLengde);
}
```

Hashfunksjon: Utplukk og ombytting

- Plukker ut noen siffer eller tegn fra nøkkelverdien
- Bytter deretter om på disse sifferne/tegnene
- Eksempel:
  - Vil ha fire-sifrede hashverdier
  - Nøkkelverdier med tolv siffer: 783248695301
  - Tar ut siffer nummer tre, seks, ni og tolv: 3851
  - Hashverdi: En eller annen omstokking av sifferne:

    Reversering: 1583 Høyreskift: 1385

    Venstreskift: 8513 Parvis bytting: 8315

Kode: Utplukk og ombytting for strenger

```
int hash(String S)
{
   // Beregner lengden av "halvparten" av strengen
   int len = S.length()/2;
   if (S.length() % 2 != 0) len += 1;

   // Plukker ut annethvert tegn i S i omvendt rekkefølge
   char C[] = new char[len];
   for (int i = 0, j = len-1; i < S.length(); i+=2, j--)
      C[j] = S.charAt(i);

   // Returnerer verdien fra Javas innebygde hashfunksjon
   // for strenger, brukt på den nye "halve" strengen

   return Math.abs(new String(C).hashCode() % hashLengde);
}
```

Hashfunksjon:
Basert på lengde av nøkkelverdi

- Beregner en hashverdi basert på en nøkkelverdi *og* nøkkelverdiens *lengde*
- Eksempel:
  - Nøkkelverdier med åtte siffer: 46286947
  - Ganger de tre første sifrene med lengden av nøkkelverdiene: 462 · 8 = 3696
  - Deler på det siste sifferet: 3696 / 7 = 528
  - hashverdi = 528 % hashlengde

Kode:
Beregning av hashverdi med lengden av streng

```
int hash(String S)
{
   // Beregner Javas hashverdi for strengen
   long h = (long) Math.abs(S.hashCode());

   // Multipliserer med lengde av strengen
   h = h * S.length();

   return (int) (h % hashLengde);
}
```

Test av ulike hashfunksjoner

- Testprogram: `hashFunctions.java`
- Tester følgende metoder ved å telle antall kollisjoner:
    1. Enkel folding
    2. Vektet folding
    3. Javas innebygde hashfunksjon (vektet folding)
    4. Midten-av-kvadrat
    5. Bytte av tallsystem
    6. Utplukk og bytting
    7. Lengde av dataverdi
- Hasher hele linjer med tekst fra standard input
- Testdatasett: `cars.txt`