Effektive algoritmer, tilfældighed og hashfunktioner

Tilfældighed og såkaldte hashfunktioner spiller en vigtig rolle i datalogien, når store mængder data skal håndteres. Fx når man skal udvikle effektive algoritmer til at sammenligne lister over forskellige menneskers yndlingsfilm.

Mængden af data, som computere skal håndtere, eksploderer under øgenavnet *Big Data*. Løsningen kan være at gøre computerne hurtigere eller at få mange af dem til at arbejde sammen. Men tit er det allervigtigste, at man har effektive algoritmer. En algoritme er en smart måde at løse et problem på.

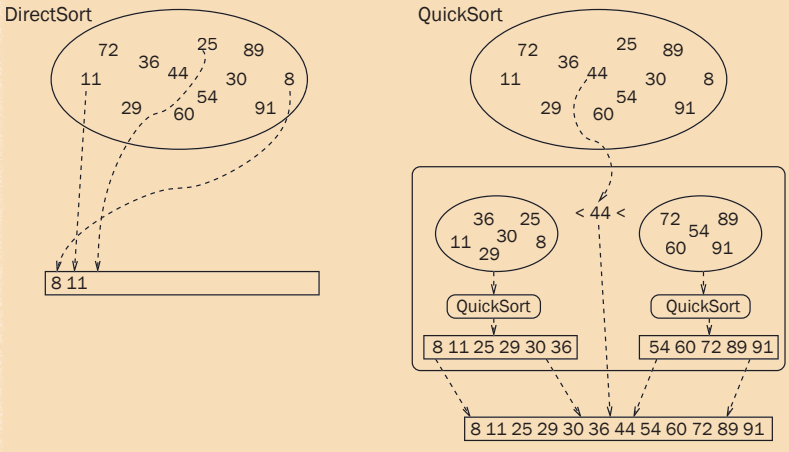
Hvis man fx skal sortere tal, er en algoritme, at man først finder det mindste tal. Det gør man ved at kigge alle tallene igennem, mens man hele tiden husker det mindste tal, man har set. Dernæst kigger man blandt de resterende tal efter det næstmindste osv. Hvis man skal sortere n tal, så ender man med at lave n(n-1)/2 talsammenligninger. Det går nemt nok, hvis der kun er få tal. Men hvis man skal sortere en milliard tal, skal man lave et astronomisk antal talsammenligninger, så det vil tage over tre år på en hurtig processor. Selv hvis arbejdet blev fordelt på tusind processorer, ville det tage en hel dag, og det ville stadig koste en masse energi.

En smartere sorteringsalgoritme er QuickSort, som er udviklet af min specialevejleder på Oxford University Tony Hoare i 1959. Den går ud på først at udtage et tilfældigt tal, og så splitter man resten af tallene op i dem, der er mindre, og dem, der er større. Derefter kan man så sortere de mindre og de større tal hver for sig. Denne algoritme forventes kun at lave (2 n ln n) talsammenligninger, hvad der oversat i tid svarer til otte sekunder i stedet for tre år på en enkelt processor, hvis man skal sortere en milliard tal. Idéen med at bruge tilfældighed her er, at det tilfældige tal, vi splitter over, har en god chance for at ligge et sted i den midterste halvdel.

Om at gemme ting på computeren

En anden vigtig brug af tilfældighed er, når man skal gemme ting i computerens lager. Man kan tænke på lageret som et kæmpestort system af kasser. Man kan godt forestille sig et fast system, hvor alle tøjdyr kommer i kasse 953. Men hvis man nu ikke har andet end tøjdyr, vil de alle ende i samme kasse, som vil blive helt overfyldt og svær at finde rundt i. Det gælder derfor om at få tingene nogenlunde pænt fordelt, så ingen af kasserne bliver overfyldt. Idéen er nu, at vi i stedet placerer alle tingene i tilfældige kasser. Det svarer til, at vi skriver kassenumrene på et lykkehjul og drejer hjulet for hver ting, vi vil putte i kasserne. Når vi drejer hjulet, får vi et helt tilfældigt kassenummer. Hvis lykkehjulet fx siger "10" til tøjkaninen, putter vi den i kasse

Forfatteren
Mikkel Thorup er professor, Københavns Universitet, Datalogisk Institut
Her leder han Center for Efficient Algorithms and Data Structures. Fra 1998 til 2013 var han ledende industriforsker hos AT&T i USA. I 2015 modtog han Villum Kann Rasmussens Årslegat.
mikkel2thorup@gmail.com



Princippet i at sortere tal direkte og at gøre det via en mere smart algoritme som QuickSort. Når der er mange tal, der skal sorteres, bliver forskellen i beregningstid meget stor.

At gemme ting på computeren via tilfældighed

Princippet i at gemme ting på computeren via tilfældighed her repræsenteret ved 19 ting, der skal gemmes i 12 kasser via et lykkehjul, der tildeler hver ting et tilfældigt kassenummer. Med denne løsning forventes hver ting rent matematisk at blive lagt sammen med (n-1)/m andre ting.
I dette tilfælde
19-1/12=1,5 ting.

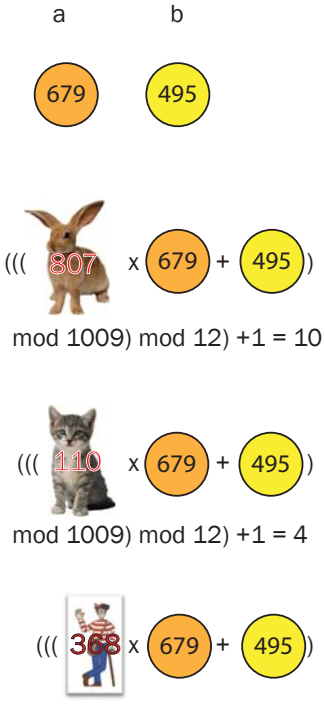
Hvor er Holger?

Princippet i at genfinde ting i computeren ved hjælp af en hashfunktion

På en computer er alle ting repræsenteret som heltal, der her står med røde tal ovenpå den ting, de repræsenterer (fx 807 for kaninen). Vi har valgt et primtal, p=1009, der er større end alle de tal, der skal gemmes. Dertil har vi to tilfældige tal, a=679 og b=495, og nu er hashfunktionen h beskrevet som h(x)=(((a·x+b) mod p) mod m)+1.
I denne formel er "mod" en operator kaldet modulus, der blot betyder "resten ved division". Fx er
8 mod 3 = 2, fordi 3·2+2 = 8.

Hashfunktionen tildeler et vilkårligt x<p en hashværdi h(x) mellem 1 og m (hvor m er antallet af kasser). For Holger er h(368)=4, så han er i kasse 4.

Pointen er, at hvis a og b vælges tilfældigt med a positiv, så er sandsynligheden for, at to givne ting hører til samme kasse, højst 1/m.



Samme smag for film?

Peter	hash
The Shawshank Redemption	83
The Godfather	21
The Godfather II	44
Star Wars Episode V	07
The Dark Knight	26
Apocalypse Now	86
Pulp Fiction	78
Goodfellas	68
The Lord of the Rings III	20
Fight Club	95
Mindste hashværdi	07

Hans	hash
A Hard Day's Night	75
The Godfather	21
Singin' in the Rain	63
Finding Nemo	40
Repulsion	77
Inside Out	39
Boyhood	88
King Kong	64
Toy Story 2	73
The Seven Samurai	15
Mindste hashværdi	15

Lars	hash
The Godfather	21
Raiders Of The Lost Ark	45
Star Wars Episode V	07
The Shawshank Redemption	83
Jaws	15
Goodfellas	68
Apocalypse Now	86
Singin' in the Rain	63
Pulp Fiction	78
Fight Club	95
Mindste hashværdi	07

Personer med den samme mindste hashværdi har en lignende smag for film. Sandsynligheden for, at to brugere har samme mindste hashværdi, er præcis antallet af film, de begge foreslår, divideret med antallet af film, de foreslår tilsammen.

10. Denne løsning forventes at give en pæn fordeling. Hvis vi har "n" ting og "m" bokse, forventes hver ting rent matematisk at blive lagt sammen med (n-1)/m andre ting.

Men hvad gør vi, hvis vi skal finde noget i de mange kasser? Det er fint nok at bruge et lykkehjul til at bestemme, hvor tingene skal smides hen, men så er det jo svært at huske, hvor vi har lagt dem. Det, vi har brug for, er en såkaldt hashfunktion, der som lykkehjulet tildeler alle ting tilfældige kasser, men hvor man altid kan genberegne, hvilken kasse en ting hører hjemme i.

Den egenskab, vi har brug for, er, at to vilkårlige ting ender i samme kasse med sandsynlighed 1/m, hvor m er antallet af kasser. Til det formål viser det sig, at vi kan klare os med blot to tilfældige tal, a og b, der ved hjælp af hashfunktionen i et hug beskriver, hvordan alting skal placeres (se figur side 36).

Når vi bruger en hashfunktion til at gemme ting i computerens lager, kalder vi det en hashtabel. Det er noget af det, som computere bruger allermest.

Om at finde ting, der ligner

En helt anden anvendelse af hashfunktioner er, når man skal finde ud af, hvor meget to mængder af ting ligner hinanden. Den slags bruges overalt i analysen af Big Data. Det kan fx være i forbindelse med lister over yndlingsfilm. Idéen er, at hvis man kan finde en bruger, der har et stort overlap med en selv i yndlingsfilm, så kan man sikkert bruge hinandens anbefalinger. Det er den slags, der ligger bag, når Netflix foreslår film baseret på, hvad man ellers ser.

Man bruger nu en hashfunktion til at tildele forskellige tilfældige tal til alle film. For hver bruger gemmer vi bare den mindste hashværdi. Vi siger, at to brugere ligner hinanden, hvis de har samme mindste hashværdi. Det lyder måske temmelig naivt, men igen er der pæn matematik bag. Sandsynligheden for, at to brugere har samme mindste hashværdi, er præcis antallet af film, de begge foreslår, divideret med antallet af film, de foreslår tilsammen. Hvis man tager eksemplet vist i tabellen herover er det for Peter og Hans 1/19, for Hans og Lars er det 2/18, og for Peter og Lars er det 7/13. Det er derfor ikke overraskende, at netop Peter og Lars fik samme mindste hashværdi. Gentager vi eksperimentet med forskellige hashfunktioner, får vi mere præcision.

Nu er det jo nemt nok, hvis det bare er lister af 10 film, der skal sammenlignes. Det bliver først rigtigt interessant, når man bruger det til at sammenligne meget store lister via nogle få hashfunktioner. Søgemaskiner bruger det til hurtigt at finde lignende hjemmesider på internettet.

Den præcise egenskab, man har brugt, er, at det for en vilkårlig mængde gælder, at alle elementer har samme chance for at få den mindste hashværdi. Sådanne hashfunktioner er ikke så nemme at lave, men de bedste, vi har, publicerede jeg med en af mine ph.d.-studerende, Søren Dahlgaard, i 2014.

Grænser for tilfældighed

Den ideelle hashfunktion tildeler alle ting helt uafhængige tilfældige tal. Men det kan slet ikke lade sig gøre, for så vil man være i en situation, hvor man ikke kan komprimere data. Computeren skal så at sige huske "det hele", og så har vi jo ikke vundet noget i forhold til den!

Dette gør hashfunktioner udfordrende at arbejde med. De fleste "leger" blot, at de har en helt tilfældig hashfunktion, og man ender ofte med at bruge nogle hashfunktioner, der godt nok ser tilfældige ud, men som man slet ikke kan stole på. Fx opdagede jeg under min forskning hos AT&T, at angreb på internettet kan få hashtabeller til at gå helt ned, hvis ikke de er lavet ordentligt. Det var i høj grad det, der gjorde mig interesseret i at forske i hashfunktioner. Tit har man nemlig ikke brug for fuld tilfældighed, men blot for nogle begrænsede sandsynlighedsmæssige egenskaber. Jeg vil gerne forstå grænserne for, hvad der er muligt, og hvor dyrt det er at gøre tingene sikre. ■

Videre læsning
Artiklen bygger på artiklen *Effektive algoritmer, tilfældighed og hashfunktioner* bragt i Årsskrift for Villum Fonden og Velux Fonden 2015.