Föreläsning 5 i ADK

Datastrukturer: bloomfilter

Viggo Kann

KTH

Algoritm för stavningskontroll

- Läs in ordlistan i lämplig datastruktur
- För varje ord **w** i texten:
  - Slå upp **w** i ordlistan och säg till om det inte finns

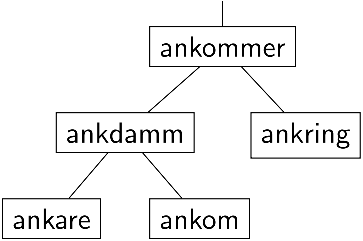
Krav på datastrukturen:

- Uppslagning ska gå snabbt
- Nya ord ska kunna läggas till snabbt
- Minnesutrymmet ska inte vara större än själva ordlistan
- Orden ska inte gå att utvinna

Datastrukturer för ordlistan

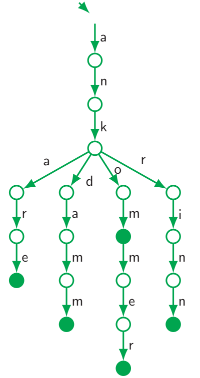
1. Sorterad array

2. Binärt sökträd



Datastrukturer för ordlistan

3. Trie/automat



4. Hashtabell

Idé till datastruktur: Boolesk hashtabell

- Låt f vara en funktion som omvandlar ett ord till ett arrayindex i en boolesk array $\mathtt{v}$ (alltså hashtabellen).
- Ett ord är med i ordlistan om $\mathtt{v}[f(\text{ordet})] = \text{sant}$

Fördelar:

- Snabbt
- Utrymmesbesparande

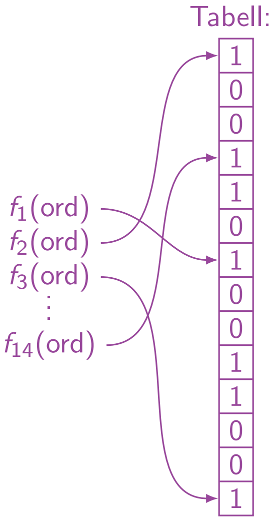
Nackdelar:

- Krockar går inte att klara av, dvs. om $f(\text{ord1}) = f(\text{ord2})$
- Om krockar ska kunna undvikas måste tabellen vara orimligt stor eller f vara onödigt svår att beräkna
- Egentligen behöver bara krockar mellan ett rätt- och ett felstavat ord förhindras

Stava: Bloomfilter

Stava lagrar ordlistan som ett bloomfilter:

- Upprepad hashning i en tabell med endast nollor och ettor
- Använd 14 hashfunktioner
  - f_i: ord $\rightarrow$ plats i hashtabell
  - $1 \leqslant i \leqslant 14$
- Givet ord som ska kontrolleras:
  - Beräkna $f_1(\text{ord}), \ldots, f_{14}(\text{ord})$
  - Acceptera ordet om det står ettor på alla dessa 14 platser



Sannolikhet för fel

- Låt n vara antalet ord i ordlistan
- Låt m vara antalet bitar i hashtabellen
- Låt k vara antalet hashfunktioner
- $P\left[\begin{array}{l}\text{En viss bit sätts till sant}\\ \text{vid en viss hashning}\end{array}\right] = \frac{1}{m}$
- $P\left[\begin{array}{l}\text{En viss bit sätts \underline{inte} till}\\ \text{sant vid en viss hashning}\end{array}\right] = 1 - \frac{1}{m}$
- $P\left[\begin{array}{l}\text{En viss bit är fortfarande}\\ \text{falsk efter } kn \text{ hashningar}\end{array}\right] = \left(1 - \frac{1}{m}\right)^{kn}$
- $P\left[\begin{array}{l}\text{En viss bit är sann}\\ \text{efter } kn \text{ hashningar}\end{array}\right] = 1 - \left(1 - \frac{1}{m}\right)^{kn}$
- $P\left[\begin{array}{l}k \text{ slumpvisa uppslagningar}\\ \text{i tabellen ger alla svaret sant}\end{array}\right] = \left[1 - \left(1 - \frac{1}{m}\right)^{kn}\right]^{k} = f(k)$

$f(k)$ minimeras av $k = -\frac{\ln 2}{n \ln(1-\frac{1}{m})} \approx \ln 2 \cdot \frac{m}{n} \approx 0{,}69\frac{m}{n}$

och ger då $f(k) = 2^{-k}$

Bloomfilter

Datastruktur för snabb koll av mängdtillhörighet i stora mängder.

Bara två operationer:

- `Insert(x)`: Stoppar in `x` i mängden
- `IsIn(x)`: Kollar om `x` är i mängden

Egenskaper:

- Båda operationerna har tidskomplexitet $\mathcal{O}(1)$
- Litet minnesutrymme
- Elementen lagras inte i klartext
- Liten sannolikhet för att `IsIn(x)` är sann om `x` inte ingår i mängden.

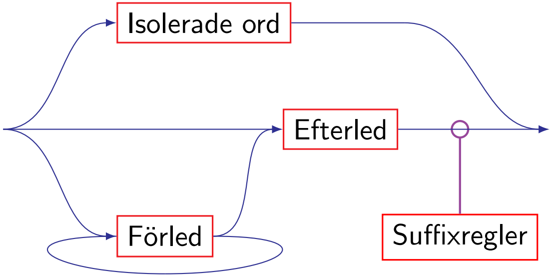
Tillämningsexempel:

- Lagring av ordlistan i stavningskontroll
- Lagring av besökta länkar i webbläsare

Stava: Hantering av sammansättningar

Tre ordlistor:

- **Isolerade ord**, cirka 1300 ord (t.ex. ömsom, eller)
- **Förled**, cirka 23 000 ord (t.ex. fotbolls, medie, stavnings)
- **Efterled**, cirka 100 000 ord (t.ex. kunskap, medium, stora, låta)



Rättstavning utan ordlista i klartext

Över 80 % av alla felstavningar beror på att en av följande saker har hänt:

- Två intilliggande bokstäver har kastats om
- En bokstav har tappats bort
- En extra bokstav har kommit in
- En bokstav har ersatts av en annan

Låt oss generera alla tänkbara ord som kan ha gett upphov till felstavningen och kolla om dom är riktiga med hjälp av bloomfiltret! Ge sedan dom riktiga orden som rättelseförslag.

Exempel på rättstavning

Felstavat ord: strutn

- Kasta om intilliggande bokstäver:

  tsrutn, srtutn, sturtn, strtun, strunt

- Sätt in en extra bokstav:

  . . . , estrutn, setrutn, sterutn, streutn, struetn, struten, strutne, . . .

- Ta bort en bokstav:

  _trutn, s_rutn, st_utn, str_tn, stru_n, strut_

- Ersätt en bokstav:

  . . . , ytrutn, . . . , sxrutn, . . . , stwutn, . . .
  . . . , strvtn, . . . , struun, . . . , struts, . . .

Optimering av hashning

Observation:

- Nästan all tid av rättstavningen tillbringas i hashfunktionen
- Försök optimera hashningen!

Ursprungliga hashfunktioner:

$$f_i(w) = \sum_j k_{i,j} \cdot w[j] \bmod p_i$$

För ett ord med n bokstäver görs vid varje hashfunktionsberäkning:

- n multiplikationer
- n moduloberäkningar
- $n - 1$ additioner
- $3n$ indexeringar

Moduloberäkning kräver division och tar längst tid av dessa operationer

Moduloberäkning med flyttal

$x \bmod p$ Resten då x divideras med p

$= x - \left\lfloor \frac{x}{p} \right\rfloor \cdot p$ Låt $q = \frac{1}{p}$

$= x - \lfloor x \cdot q \rfloor \cdot p$

Förbehandla genom att beräkna $\frac{1}{p_i}$ för alla hashfunktioner

Därefter kan en moduloberäkning ersättas av

- En omvandling heltal $\rightarrow$ flyttal
- En flyttalsmultiplikation
- En om vandling flyttal $\rightarrow$ heltal
- En heltalsmultiplikation
- En heltalssubtraktion

Test av hashningsoptimering

Indata Ordlista med 200 000 ord varav knappt hälften finns i Stavas ordlista

Tid utan optimering	27,3 s
Tid med kompilatorns optimering -O2 påslagen	23,0 s
Tid om dessutom bara en avslutande moduloberäkning utförs	11,9 s
Tid om processorns inbyggda modulofunktion används	6,4 s
Tid om flyttalsmoduloberäkning används	5,2 s
Tid om Bob Jenkins hashfunktion används (dvs ingen moduloberäkning alls)	4,0 s