FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Gevorderde algoritmen

Bert De Saffel

Master in de Industriële Wetenschappen: Informatica Academiejaar 2018–2019

Gecompileerd op 14 augustus 2019

UNIVERSITEIT
GENT

Inhoudsopgave

Deel I

Strings

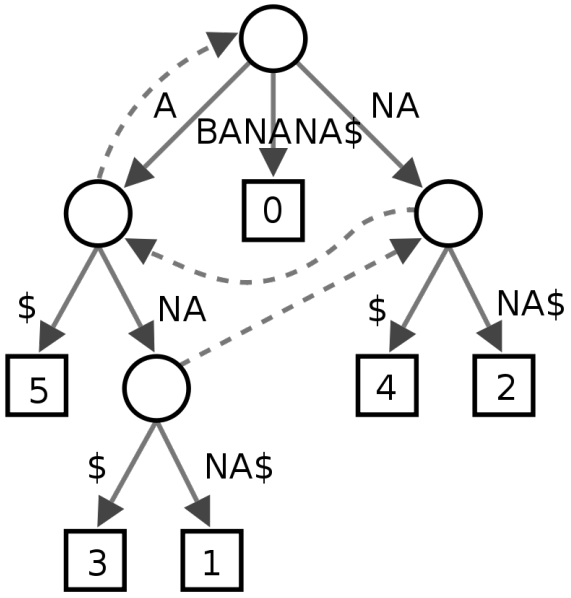
Hoofdstuk 1

Indexeren van vaste tekst

- Sommige zoekoperaties gebeuren op een vaste tekst T waarin frequent gezocht wordt naar een veranderlijk patroon P.
- Voorbereidend werk op de tekst om efficiënter te doorzoeken.
- Alle zoekmethoden in hoofdstuk **??** verrichten voorbereidend werk op het patroon.
  - In het slechtste geval is dit $O(t + p)$.
  - Dit kan gereduceerd worden tot $O(p)$ door eerst $O(t)$ voorbereidend werk te doen op T.
  - Via **suffixen**.
  - Als een patroon in de tekst voorkomt, moet het een prefix zijn van één van de suffixen.
  - Een suffix dat begint op lokatie i wordt aangeduidt met suff_i.

1.1 Suffixbomen

- Gebaseerd op de **Patriciatrie**.
- Het aantal inwendige knopen is $O(t)$ en de vereiste geheugenruimte is $O(|\Sigma|t)$.
- Kan geconstrueerd worden in $O(t)$.
- Er zijn een aantal **wijzigingen** ten opzichte van een originele Patriciatrie:
  1. Een patriciatrie slaat strings op bij de bladeren. Hier volstaat de index i van suff_i.
  2. De testindex wordt vervangen door een begin- en eindindex, die een substring aangeeft van T in elke knoop.
  3. In elke inwendige knoop kan een staartpointer opgenomen worden.
     - De **staart(s)** van een string s is de string bekomen door het eerste karakter te verwijderen.
     - Er is een staartpointer van een inwendige knoop x naar een andere inwendige knoop y als de padstring van y hetzelfde is als staart(s).
     - Op figuur 1.1 is er bijvoorbeeld een staartpointer van de rechtse inwendige knoop met als padstring `NA` naar de linkse inwendige knoop met als padstring `A` omdat staart(`NA`) = `A`.
- De voorwaarde dat een string geen prefix mag zijn van een ander werd vroeger opgelost door een extra afsluitend karakter te introduceren, maar dat is hier moeilijker.



Figuur 1.1: Een suffixboom voor het woord `BANANA$`. Elk van de suffixen `BANANA$`, `ANANA$`, `NANA$`, `ANA$`, `NA$` en `A$` kan gevonden worden in deze boom. Het suffix `NANA$` wordt gevonden door twee keer de rechterdeelboom te nemen vanuit de wortel. De index 2 wijst erop dat de suffix begint bij $T[2]$. De gestreepte verbindingen zijn staartpointers.

- Elk karakter van T wordt één per één toegevoegd in de suffixboom.
- Na k iteraties zitten er suffixen van $T[0] \cdots T[k-1]$ in de boom zonder afsluitteken.
- _ToDo: ...
- Dus om ervoor te zorgen dat deze voorwaarde geldig is, moet T eindigen op een karakter dat nergens anders voorkomt in de tekst. Op figuur 1.1 is dit het karakter `$`.

1.2 Suffixtabellen

- Eenvoudiger alternatief voor een suffixboom, maar vereist minder geheugen.
- Een tabel met de gerangschikte suffixen (hun startindices) van T.

! Een suffixtabel bevat geen informatie over het gebruikte alfabet.

- Een suffixtabel construeren kan door eerst de suffixboom op te stellen in $O(t)$ en daarna deze in inorder te overlopen, ook in $O(t)$.
  - De suffixtabel, geconstrueerd uit de suffixboom uit figuur 1.1.

$$A = \begin{bmatrix} 6 & 5 & 3 & 1 & 0 & 4 & 2 \end{bmatrix}$$

    Het eerste element ($A[0] = 6$) is een verwijzing naar het eindkarakter, maar zit niet in de boom.

- Er is echter nog een belangrijke hulpstructuur nodig, de LGP-tabel.
  - Langste Gemeenschappelijke Prefix - tabel.
  - Voor suff_i is $LGP[i]$ de lengte van het langste gemeenschappelijke prefix van suff_i.

- De alfabetische opvolger van suff_i wordt gegeven door $\textbf{opvolger}(\textbf{suff}_{SA_{|j|}}) = \textbf{suff}_{SA_{|j+1|}}$.

- De LGP-tabel wordt opgesteld via de suffixtabel:
  - Start met suff_0.
  - Zoek j zodat $A[j] = 0$.
  - Bepaal het langste gemeenschappelijke suffix:
    - Start met $l = 0$.
    - Verhoog l tot $T[i + l]$ niet meer overeenkomt.

1.3 Tekstzoekmachines

1.3.1 Inleiding

- Tekstzoekmachines zijn in eerste instantie gelijkaardig aan databanksystemen.
  - Documenten worden bewaard in een repository.
  - Er worden indexen bijgehouden om snel documenten te doorlopen.
  - Er kunnen queries uitgevoerd worden relevante documenten te zoeken.
- Maar ze verschillen ook van databanksystemen.
  - Een query voor een tekstzoekmachine bestaat enkel uit woorden of zinnen.
  - In een databanksysteem zal de query resultaten geven die voldoen aan een logische uitspraak, maar bij een tekstzoekmachine is dit vager.
  - Een tekstzoekmachine geeft niet alle resultaten terug, maar enkel de meest relevante. Het begrip relevantie is ook niet exact, aangezien dit afhangt van de gebruiker.
- Het gebruik van **indices** om tekst te indexeren is onmisbaar.

1.3.2 Zoeken van tekst en informatie verzamelen

Queries

- In een traditionele databank hebben gegevens een unieke sleutel, wat niet het geval is bij tekstdocumenten op het internet.
- Soms hebben tekstdocumenten *metadata* zoals de auteur, het onderwerp en het aantal pagina's, maar deze zijn slechts occasioneel nuttig.
- De meest voorkomende manier om in tekst te zoeken is het zoeken naar **inhoud** aan de hand van een **query**.
- Aangezien dat een tekstzoekmachine probeert relevante documenten weer te geven, moet gemeten kunnen worden hoe goed deze documenten zijn.
- Een tekstzoekmachine heeft een bepaalde **effectiveness** voor een getal r waarbij de meeste van de eerste r resultaten relevant zijn.
  - De *effectiveness* wordt vaak bepaald door de **precision** en **recall**.
  - De *precision* is de verhouding van documenten dat relevant zijn.
  - De *recall* is de verhouding van relevante documenten die gekozen zijn.

- Voorbeeld:
  - Een tekstdatabank bevat 20 documenten.
  - Een gebruiker zoekt in deze databank met een query en er worden 8 resultaten teruggegeven.
  - De gebruiker vindt dat 5 van deze resultaten relevant zijn voor hem, en dat er nog 2 andere documenten in de tekstdatabank zitten die niet door de tekstzoekmachine gegeven worden.
  - De textitprecision is 5/8.
  - De *recall* is 5/7.
- Veel van de technieken zorgen ervoor dat *effectiveness* vrij hoog blijft.

Voorbeelddatabanken

- De **Keeper databank**.

1	The old night keeper keeps the keep in the town.
2	In the big old house in the bog old gown.
3	The house in the town had the big old keep.
4	Where the old night keeper never did sleep.
5	The night keeper keeps the keep in the night.
6	And keeps in the dark and sleeps in the night.

  - Bevat 6 documenten elk met 1 lijn.
  - Verschillende eenvoudige technieken om in deze databank te zoeken.
    - De query `big old house` waarbij de query als één enkele string beschouwd wordt zal enkel document 2 geven.
    - De query `big old house` waarbij elk woord in een verzameling van woorden komt (**bag-of-word**, {`big`, `old`, `house`}) zal documenten 2 en 3 teruggeven. De volgorde van de woorden in deze verzameling spelen geen rol en elk woord wordt afzonderlijk bekeken of ze voorkomt in het document of niet.
  - Meerdere technieken om de **woordenschat** van een tekstdatabank te reduceren:
    - **Zonder aanpassingen**
      `And and big dark did gown had house In in keep keeper keeps light never night old sleep sleeps The the town Where`
    - **Hoofdletter-invariantie**
      `and big dark did gown had house in keep keeper keeps light never night old sleep sleeps the town where`
    - **Verwijderen meerdere varianten van hetzelfde woord**
      `and big dark did gown had house in keep light never night old sleep the town where`
    - **Verwijderen van vaak voorkomende woorden**
      `big dark did gown house keep light night old sleep town`
- Twee hypothetische databanken om efficiëntie te bespreken:
- Elke tekstzoekmachine moet aan een aantal voorwaarden voldoen:
  - De queries moeten goed geanalyseerd worden.
  - De queries moeten snel geanalyseerd worden.

	NewsWire	Web
Grootte in gigabytes	1	100
Aantal Documenten	400 000	12 000 000
Aantal woorden	180 000 000	11 000 000 000
Aantal unieke woorden	400 000	16 000 000
Aantal unieke woorden per document, opgesomd	70 000 000	3 500 000 000

- Minimaal gebruik van resources zoals geheugen en bandbreedte.
- Schaalbaar naar grote volumes van data.
- Resistent tegen het wijzigen van documenten.

Gelijkaardigheidsfuncties

- Elke tekstzoekmachine maakt gebruik van een rankingsysteem om documenten te ordenen.
- Om documenten te ordenen wordt er gebruik gemaakt van een gelijkaardigheidsfunctie.
- Hoe hoger de waarde van deze functie, hoe hoger de kans dat de gebruiker dit document als relevant zal beschouwen.
- De r meest relevante documenten worden dan gegeven aan de gebruiker.
- In **bag-of-words** queries wordt de gelijkaardigheidsfunctie samengesteld door een aantal statistische variabelen:
  - $f_{d,t}$ is de frequentie van het woord t in document d.
  - $f_{q,t}$ is de frequentie van het woord t in de query q.
  - f_t is het aantal documenten dat één of meer keer het woord t bevat.
  - F_t is het aantal keer dat t voorkomt in de hele tekstdatabank.
  - N is het aantal documenten in de tekstdatabank.
  - n het aantal geïndexeerde woorden in de tekstdatabank.
- Deze waarden kunnen gecombineerd worden om drie vaststellingen te maken:
  1. Een woord dat in veel documenten voorkomt krijgt een kleiner gewicht.
  2. Een woord dat veel in één document voorkomt krijgt een groter gewicht.
  3. Een document dat veel woorden bevat krijgt een kleiner gewicht.
- Er is een **query vector** $\vec{w}_q$ en een **document vector** $\vec{w}_d$, waarbij elk component in deze vector gedefinieerd wordt als

$$w_{q,t} = \ln\left(\frac{N}{f_t}\right) \qquad w_{d,t} = f_{d,t}$$

- De maat van gelijkheid $S_{q,d}$, de maat in hoeverre het document d relevant is voor query q, kan bekomen worden door de cosinus van de hoek tussen deze twee vectoren te nemen.

$$S_{q,d} = \frac{\vec{w}_d \cdot \vec{w}_q}{||\vec{w}_d|| \cdot ||\vec{w}_q||} = \frac{\sum_t w_{d,t} \cdot w_{q,t}}{\sqrt{\sum_t w_{d,t}^2} \cdot \sqrt{\sum_t w_{q,t}^2}}$$

- De grootheid $w_{q,t}$ encodeert de **inverse document frequentie** van een woord t.
- De grootheid $w_{d,t}$ encodeert de **woord frequentie** van een woord t.

- Het nadeel aan deze methode is dat elk document in beschouwing genomen moet worden, maar dat slechts r documenten gevonden moeten worden.
- Voor de meeste documenten is de gelijkaardigheidswaarden insignificant.
- Deze **brute-force** methode kan uitgebreidt worden tot betere methoden, via **indices**.

1.3.3 Indexeren en query-evaluatie

- Een **index** in deze context is een datastructuur dat een woord afbeeldt op documenten dat dit woord bevat.
- Het verwerken van een query kan dan enkel uitgevoerd worden op documenten die minstens één van de query woorden bevat.
- Er zijn vele soorten indices, maar de meest gebruikte is een **inverted file index**: een collectie van lijsten, één per woord, dat documenten bevat dat dit woord bevat.
- Een **normale inverted file index** bestaat uit twee componenten.
  1. Voor elk woord t houdt de **zoekstructuur** het volgende bij:
     - een getal f_t van het aantal documenten dat t bevat, en
     - een pointer naar de start van de correspondeerde geïnverteerde lijst.
  2. Een **verzameling van geïnverteerde lijsten**, waarbij elk lijst het volgende bijhoudt voor een woord t:
     - de sleutels van documenten d die t bevatten, en
     - de verzameling van frequencies $f_{d,t}$ van woorden t in document d.
     - $\rightarrow \langle d, f_{d,t} \rangle$ paren.
- Samen met W_d en deze twee componenten zijn geordende queries mogelijk.
- Een *inverted file* voor de *keeper database* is te zien op tabel 1.1.
- Er kan nu een **query evaluatie** algoritme opgesteld worden (gevisualiseerd op figuur 1.2).
  1. Er wordt een accumulator A_d bijgehouden voor elk document d. Initieel is elke $A_d = 0$.
  2. Voor elk woord t in de query worden volgende operaties uitgevoerd:
     (a) Bereken $w_{q,t} = \ln\left(\frac{N}{f_t}\right)$ en vraag de geïnverteerde lijst op van t.
     (b) Voor elk paar $\langle d, f_{d,t} \rangle$ in de geïnverteerde lijst worden volgende operaties uitgevoerd:
        i. Bereken $w_{d,t}$.
        ii. Stel $A_d = A_d + w_{q,t} w_{d,t}$.
  3. Voor elke $A_d > 0$, stel $S_d = A_d / W_d$.
  4. Identificeer de r grootste S_d waarden en geef de correspondeerde documenten terug.
- Het is ook nog mogelijk om **de posities van de woorden in het document te indexeren**.
  - Het paar $\langle d, f_{d,t} \rangle$ kan uitgebreidt worden om de posities p bij te houden waar dat t voorkomt in d.

$$\langle d, f_{d,t}, p_1, \cdots, p_{f_{d,t}} \rangle$$

woord t	f_t	Geïnverteerde lijst voor t
`and`	1	$\langle 6,2\rangle$
`big`	2	$\langle 2,2\rangle$ $\langle 3,1\rangle$
`dark`	1	$\langle 6,1\rangle$
`did`	1	$\langle 4,1\rangle$
`gown`	1	$\langle 2,1\rangle$
`had`	1	$\langle 3,1\rangle$
`house`	2	$\langle 2,1\rangle$ $\langle 3,1\rangle$
`in`	5	$\langle 1,1\rangle$ $\langle 2,2\rangle$ $\langle 3,1\rangle$ $\langle 5,1\rangle$ $\langle 6,2\rangle$
`keep`	3	$\langle 1,1\rangle$ $\langle 3,1\rangle$ $\langle 5,1\rangle$
`keeper`	3	$\langle 1,1\rangle$ $\langle 4,1\rangle$ $\langle 5,1\rangle$
`keeps`	3	$\langle 1,1\rangle$ $\langle 5,1\rangle$ $\langle 6,1\rangle$
`light`	1	$\langle 6,1\rangle$
`never`	1	$\langle 4,1\rangle$
`night`	3	$\langle 1,1\rangle$ $\langle 4,1\rangle$ $\langle 5,1\rangle$
`old`	4	$\langle 1,1\rangle$ $\langle 2,2\rangle$ $\langle 3,1\rangle$ $\langle 4,1\rangle$
`sleep`	1	$\langle 4,1\rangle$
`sleeps`	1	$\langle 6,1\rangle$
`the`	6	$\langle 1,3\rangle$ $\langle 2,2\rangle$ $\langle 3,3\rangle$ $\langle 4,1\rangle$ $\langle 5,3\rangle$ $\langle 6,2\rangle$
`town`	2	$\langle 1,1\rangle$ $\langle 3,1\rangle$
`where`	1	$\langle 4,1\rangle$

d	1	2	3	4	5	6
W_d	4	4.2	4	2.8	4.1	4

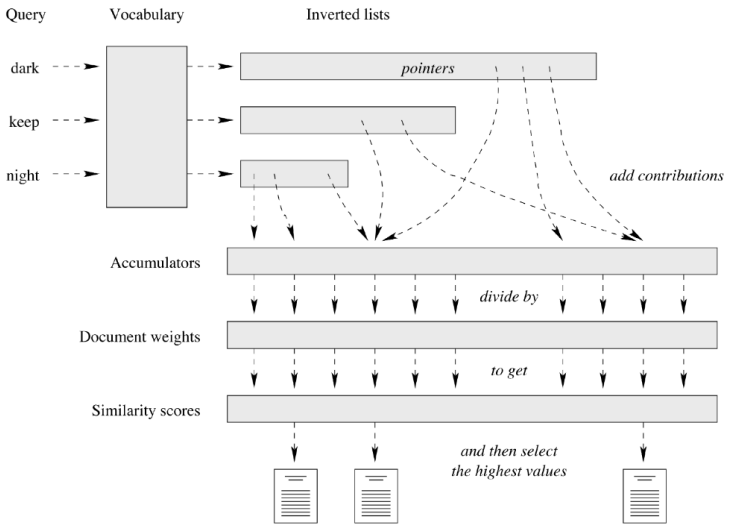
Tabel 1.1: Een op document niveau geïnverteerd bestand voor de *Keeper* databank. Elk woord t bestaat uit f_t en een lijst van paren, waarbij elk paar bestaat uit een sleutel d van een document en de frequentie $f_{d,t}$ van het woord t in d. Ook zijn de waarden van W_d te zien, berekend volgens $W_d = \sqrt{\sum_t w_{d,t}^2} = \sqrt{\sum_t f_{d,t}^2}$.

1.3.4 Queries met zinnen

- Een query kan een expliciete zin bevatten, aangeduid met aanhalingstekens, zoals `"philip glass"` of `"the great flydini"`.
- Soms is het ook impliciet zoals `Albert Einstein` of `San Francisco hotel`.
- _ToDo: idk

1.3.5 Constructie van een index

- Het volume van de data is veel te groot om alles in het geheugen te doen.
- Er zijn drie methoden:
  1. **In-memory Inversion**
     - Alle documenten wordt tweemaal overlopen.
       (a) Een eerste keer telt de frequentie f_t van alle verschillende woorden van alle documenten.
       (b) Een tweede maal plaatst de pointers in de juiste positie.
  2. **Sort-Based Inversion**
  3. **Merge-Based Inversion**



Figuur 1.2: Het gebruik van een geïnverteerd bestand en een verzameling van accumulators om gelijkaardigheidswaarden te berekenen.