Analisis Memori untuk Kompresi *Dictionary*

Definisikan beberapa istilah dan definisi kembali, pada koleksi Reuters-RCV1:

- Terdapat sekitar $400\,000$ terms.
- Panjang rata-rata sebuah terms (bahasa inggris) adalah 8 karakter.

Definisikan nilai ukuran:

- Asumsikan 1MB = 10^3 KB = 10^6 B
- Perhatikan bahwa 1 byte terdiri dari 8 bit, yang bisa merepresentasikan $2^8 = 256$ nilai berbeda secara teori.
- 1 karakter pada sebuah string dapat direpresentasikan dengan nilai 1 byte.

DaaS

Perhatikan bahwa bila terdapat $400\,000$ terms, maka string panjang gabungan setiap terms kira-kira akan sepanjang $8 \frac{\text{karakter}}{terms} \times 400\,000 \text{ terms} \times 1 \frac{\text{bytes}}{\text{karakter}} = 2^5 \times 10^5 \text{ bytes}$.

Perhatikan bahwa untuk string sepanjang itu, akan dibutuhkan kira-kira $\log_2 \left(2^5 \times 10^5\right) = 5 + 5 \log_2(10) \approx \lceil 5 + 16.6 \rceil = 22 \text{ bits}$. Karena 8 bits itu hitungannya 1 byte, maka akan kita sisakan 3 byte untuk pointer si posisi term pada untaian terms yang sudah digabung.

Satu term akan membutuhkan:

- $+3$ bytes untuk alamat
- $+4$ bytes untuk frekuensi
- $+4$ bytes untuk pointer ke posting
- $+8$ bytes (rata-rata) untuk menaruhnya ke dalam untaian terms.

Sehingga total membutuhkan 19 bytes/terms, untuk $400\,000$ terms, maka akan menggunakan 7.6×10^6 bytes, atau setara dengan 7.6MB

DaaS With Blocking (Tidak semua indeks dibikin alamatnya ke untaian terms, lebih tepatnya, hanya menyimpan setiap k-th term string)

Analisis Memori

Perhatikan bahwa kita mula mula perlu menambahkan size dulu ke setiap saat string mulai, alasannya ialah agar bisa kita tokenize lah untaian stringnya, dan kita tahu kapan itu menjadi sebuah entri term lain saat di-*scan*.

Satu term akan membutuhkan:

- $+1$ byte untuk panjang kata
- $+8$ bytes untuk menaruhnya ke dalam untaian terms
- $+4$ untuk frekuensi
- $+4$ untuk pointer ke posting

- $+\frac{3}{k}$ untuk alamat. Perhatikan bahwa karena tidak semua terms akan disimpan pointernya, maka kita hanya akan menggunakan 3 bytes saja setiap k kali.

Untuk $k = 4$:

$(17 + \frac{3}{k}) \times 400\,000 = 7.1 \times 10^6$ bytes, yang berarti sekitar 7.1 MB

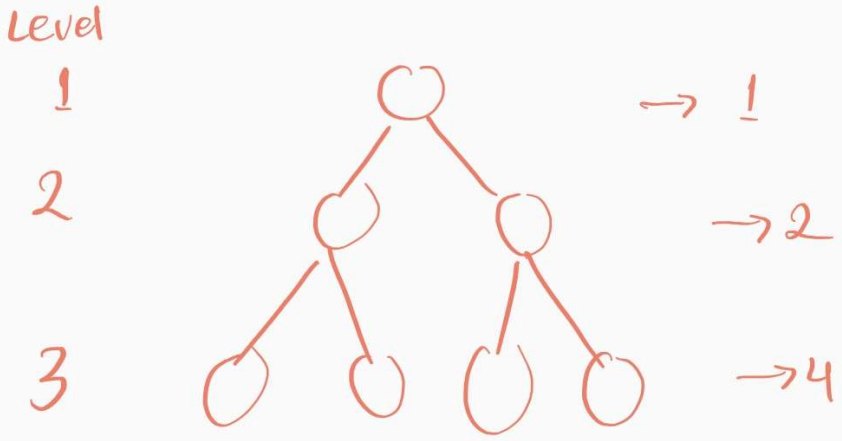
Perhatikan bahwa:

$$\lim_{k \to \text{inf}} (17 + \frac{3}{k}) \times 400\,000 = 6.8\text{MB} \tag{2}$$

Kenapa tidak menggunakan k yang besar? Karena tradeoff dengan waktu dalam mentraverse untuk mencari termsnya, sementara ukurannya tidak berubah banyak. Bahkan untuk $k = 10$, ukurannya masih sekitar 6.92 MB. Begitu pula untuk $k = 10$.

Analisis Waktu

Ketika tanpa blocking, pencarian di dictionary itu menggunakan binary search tree biasa. Ilustrasi berikut merepresentasikannya.



untuk banyak node $2^k - 1$ node, terdapat k level.
Misal $n = 2^k - 1$, average seek time ialah :

$$\frac{1}{2^k - 1} \cdot (1.1 + 2.2 + 3.4 + 4.8 + 5.16 + \dots + k.2^{k-1})$$

Level 1 · Level 2 · · · · · Level k

$$= \frac{\sum_{i=1}^{k} i \cdot 2^{i-1}}{n} = \frac{\sum_{i=1}^{\log_2(n+1)} i \cdot 2^{i-1}}{n} \approx \Theta(\log n)$$

$n + 1 = 2^k$
$\log(n+1) = k$

Extended Master Theorem

WolframAlpha's Expansion:

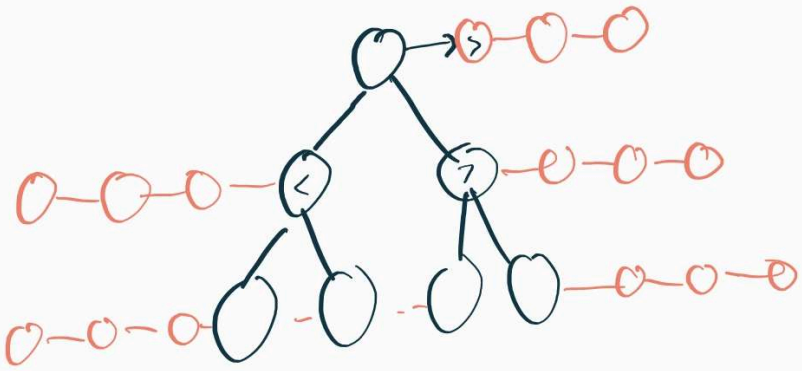
$$\frac{1}{n} \cdot \log_2(n+1) + \log_2(n+1) - 1$$

$$\approx \log_2(n+1) \left(\frac{n+1}{n} \right) - 1$$

$n = 1000 \rightarrow 8,97719$

$n = 10^6 \rightarrow 18,9315$

Dengan blocking:



Saat ada blocking, definisikan blocking b = 4 misalnya:
Dengan asumsi treenya complete binary dengan setiap node ada block node appended tambahan sebanyak b-1:

Kompleksitas waktu aproksimasi; hitung total operasi compare untuk setiap term

- Banyak node = n, ukuran binary search tree (doang) n/b

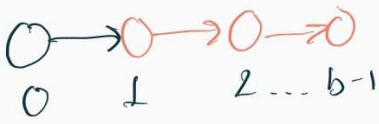
Komponen 1 :

$$\left(\sum_{i=1}^{\log_2(n/b+1)} i \cdot 2^{i-1} \right) \times b$$

BST dengan n/b node

Karena setiap komponen harus ke block yang tepat

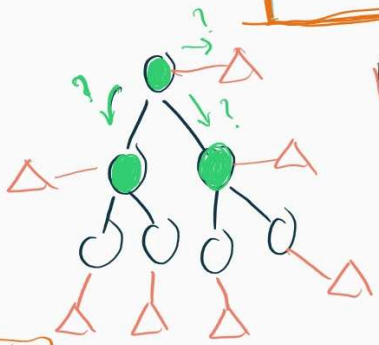
Komponen 2 :



$$\sum_{i=1}^{b-1} i = \frac{(b-1) \times b}{2}$$

comparison untuk setiap node di bloknya

Komponen 3

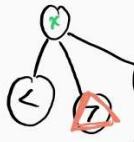


$$\sum_{i=1}^{\log_2(n/b+1)-1} i \cdot 2^{i-1} \times b$$

$$+ \left(\frac{n}{b}+1\right) \times 2^{\left(\log_2(n/b+1)-1\right)} \cdot b$$

← komponen terbawah

Perhatikan bahwa ada node-node internal yang butuh komparasi lebih, misal pada node x, perbandingan pertama bila term bernilai 7, bisa saja kita berada di block ini, atau harus traverse ke anak kanan.



Total Comparison:

$$\left(\sum_{i=1}^{\log_2(n/b+1)} (i \cdot 2^{i-1}) \right) \cdot b + \frac{(b-1) \cdot b}{2} + \sum_{i=1}^{\log_2(n/b+1)-1} (i \cdot 2^{i-1}) \cdot b$$

$$+ \left(\frac{n}{b} + 1\right) + 2b\left(\log_2(n/b + 1) - 1\right)$$

Average seek time untuk $b = 4$:

$$\left[\left(\sum_{i=1}^{\lceil \log_2(\frac{n}{4}+1) \rceil} i \cdot 2^{i-1} \right) \cdot 4 + 6 + \sum_{i=1}^{\lceil \log_2(n/4+1) - 1 \rceil} (i \cdot 2^{i-1}) \cdot 4 \right.$$

$$\left. + \frac{n}{4} + 1 + 8\left(\log_2\left(\frac{n}{4}+1\right) - 1\right) \right] \times \frac{1}{n}$$

$$n = 10^6 \rightarrow \frac{17825796 + 6 + 8388612 + 250136}{10^6}$$

$$\rightarrow \approx 26,4645 \rightarrow \Theta(\log n)$$

Perhatikan bahwa saat $b \rightarrow n$, maka komponen

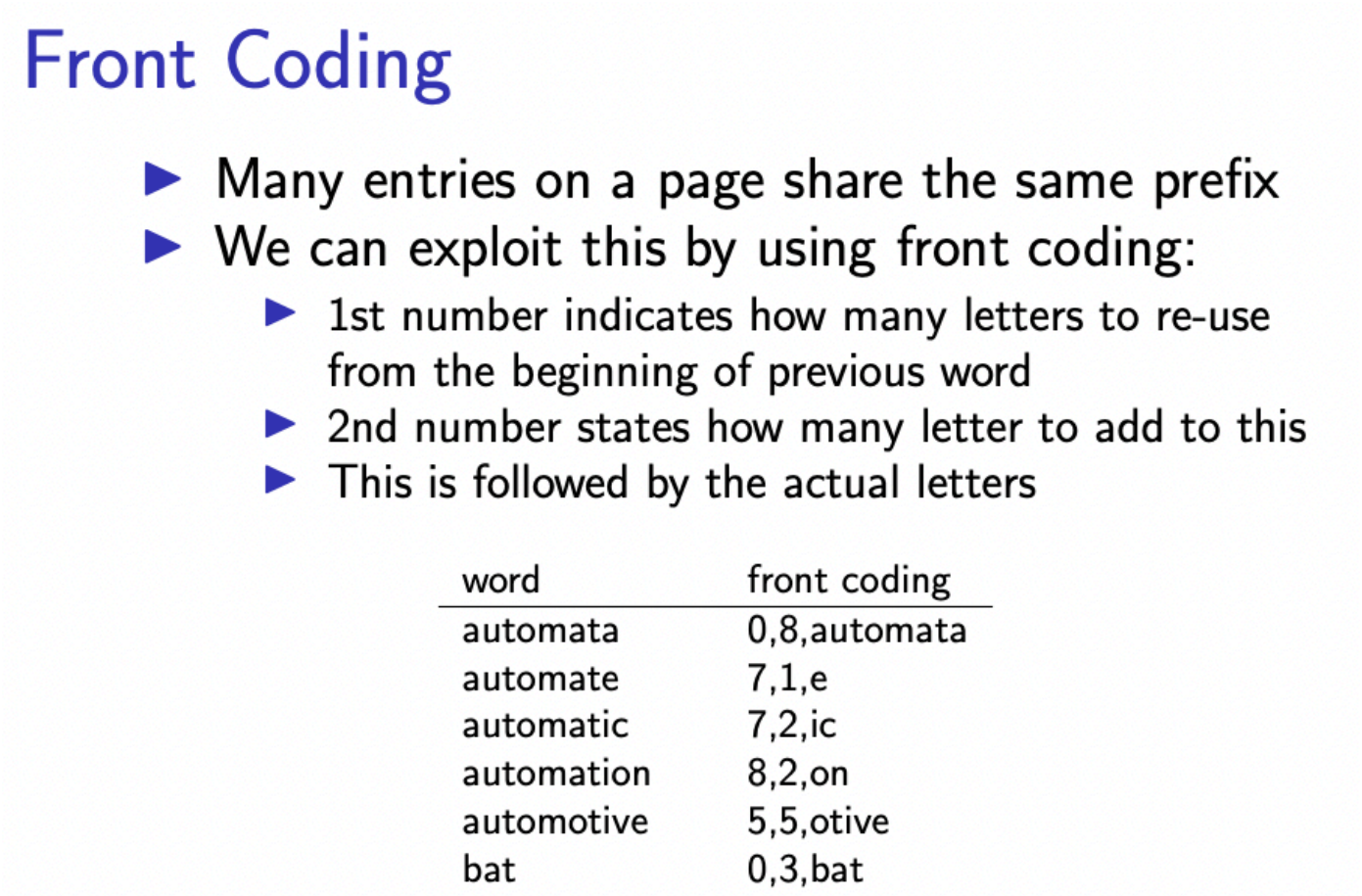
$\frac{b(b-1)}{2} \rightarrow \frac{n(n-1)}{2}$, yang menyebabkan

average seek time menjadi linear

∴ bila b semakin besar, memori ↓ waktu seek ↑

Front Coding

Menurut saya teknik ini keren, implementasinya juga jadi menyerupai struktur data Trie, tapi memorynya in place.



Terdapat beberapa variasi encoding untuk front coding ini, ada yang menggunakan kata sebelumnya menjadi patokan, yang menurut saya menarik, namun tentu saja pasti membutuhkan overhead dari segi waktu untuk mendecode, namun dengan tradeoff memori yang **jauh lebih kecil** bila prefixnya banyak yang dekat. Contoh di atas menurut saya cukup bagus untuk variasi kata yang prefixnya beragam.

Sumber:

- https://www.dcs.bbk.ac.uk/~dell/teaching/nlp/dell_iir_ch05.pdf

Untuk implementasi, saya rasa kode dari repositori ini cukup bagus:

- https://github.com/WikiBox/SD