







Index Compression

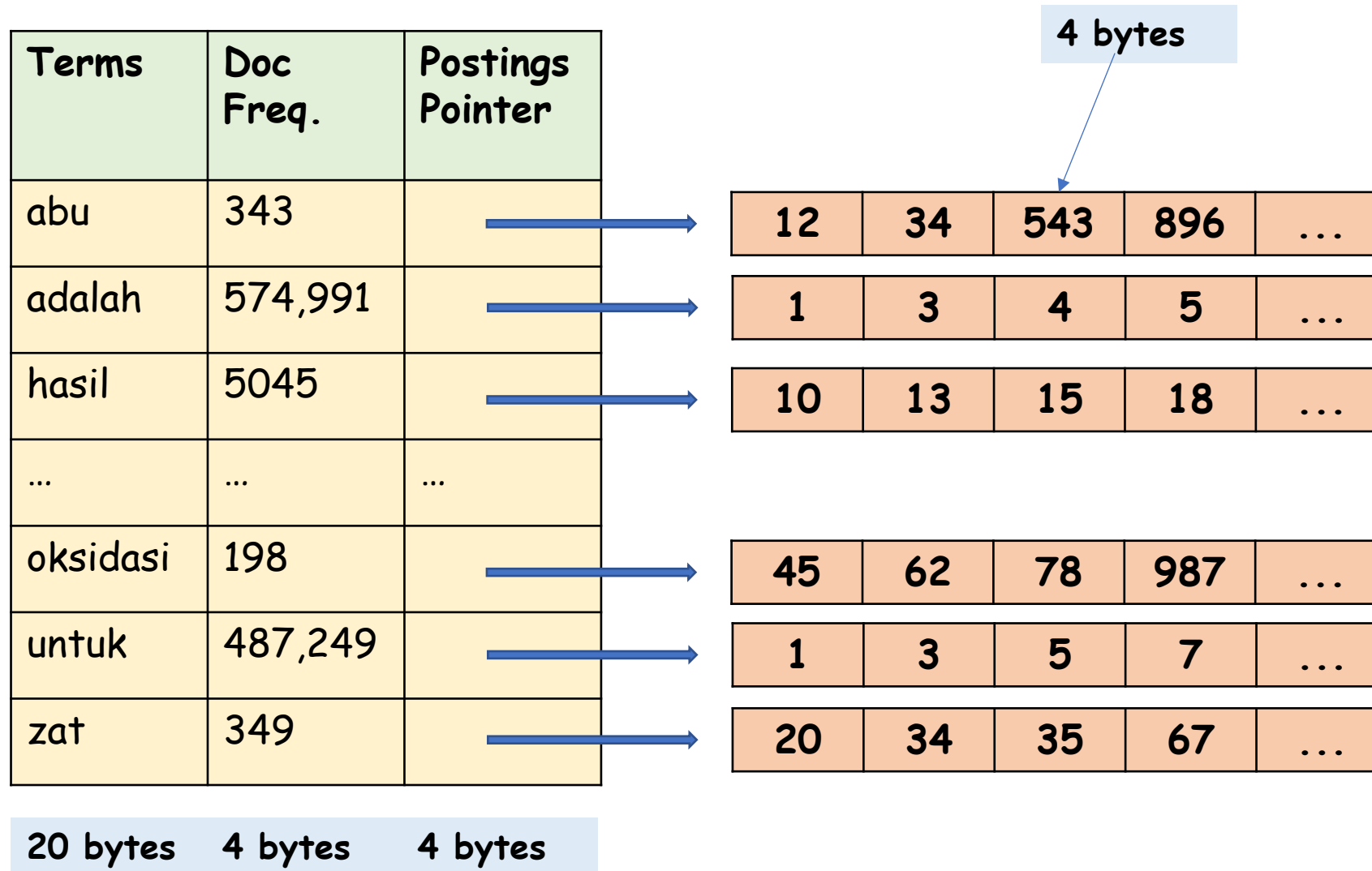
Alfan Farizki Wicaksono

Fakultas Ilmu Komputer, Universitas Indonesia

Recap - Overview of Our Inverted Index

Terms	Doc Freq.	Postings Pointer						
abu	343		<table><tr><td>12</td><td>34</td><td>543</td><td>896</td><td>...</td></tr></table>	12	34	543	896	...
12	34	543	896	...				
adalah	574,991		<table><tr><td>1</td><td>3</td><td>4</td><td>5</td><td>...</td></tr></table>	1	3	4	5	...
1	3	4	5	...				
hasil	5045		<table><tr><td>10</td><td>13</td><td>15</td><td>18</td><td>...</td></tr></table>	10	13	15	18	...
10	13	15	18	...				
...						
oksidasi	198		<table><tr><td>45</td><td>62</td><td>78</td><td>987</td><td>...</td></tr></table>	45	62	78	987	...
45	62	78	987	...				
untuk	487,249		<table><tr><td>1</td><td>3</td><td>5</td><td>7</td><td>...</td></tr></table>	1	3	5	7	...
1	3	5	7	...				
zat	349		<table><tr><td>20</td><td>34</td><td>35</td><td>67</td><td>...</td></tr></table>	20	34	35	67	...
20	34	35	67	...				

Recap - Overview of Our Inverted Index



Mengapa Compression?

Untuk sebuah data Web sebesar **420GB** yang mengandung:

# Documents	25,000,000
# Terms	35,000,000
# Postings	6,000,000,000
Uncompressed Storage Cost	~25 GB

- Jika memungkinkan, sebagian besar inverted index sebenarnya sebaiknya disimpan di memori (query performance).
- Misal, sebuah perusahaan search engine menggunakan 1000 komputer untuk menjawab sebuah query.
- **Reduksi storage sebesar 5% proporsional dengan mematikan 50 dari 1000 mesin!**

Mengapa Compression?

- Reduce storage requirements
- Keep larger parts of the index in memory
- Faster query processing

Contoh:

Ada sebuah state-of-the-art method yang mampu membuat index dari 25 juta dokumen web (420GB) hanya dengan storage berukuran 5GB dan mampu menjawab query dalam rata-rata 10 ms.

25GB --> 5GB berarti tereduksi 80%

Apakah ukuran term vocabulary ada batasnya? Atau akan terus bertambah seiring bertambahnya dokumen?

Prediksi Ukuran Term Vocabulary

Heap's Law memprediksi: $M = k T^b$

M adalah banyaknya term di vocab (dictionary) dan T adalah banyaknya token di collection. Jika dengan log-log plot, akan menghasilkan hubungan linier antara M dan T .

$$\log(M) = \log(k) + b \cdot \log(T)$$

Biasanya $30 \leq k \leq 100$ dan $b \approx 0.5$

Contoh Empirical Evidence Heap's Law

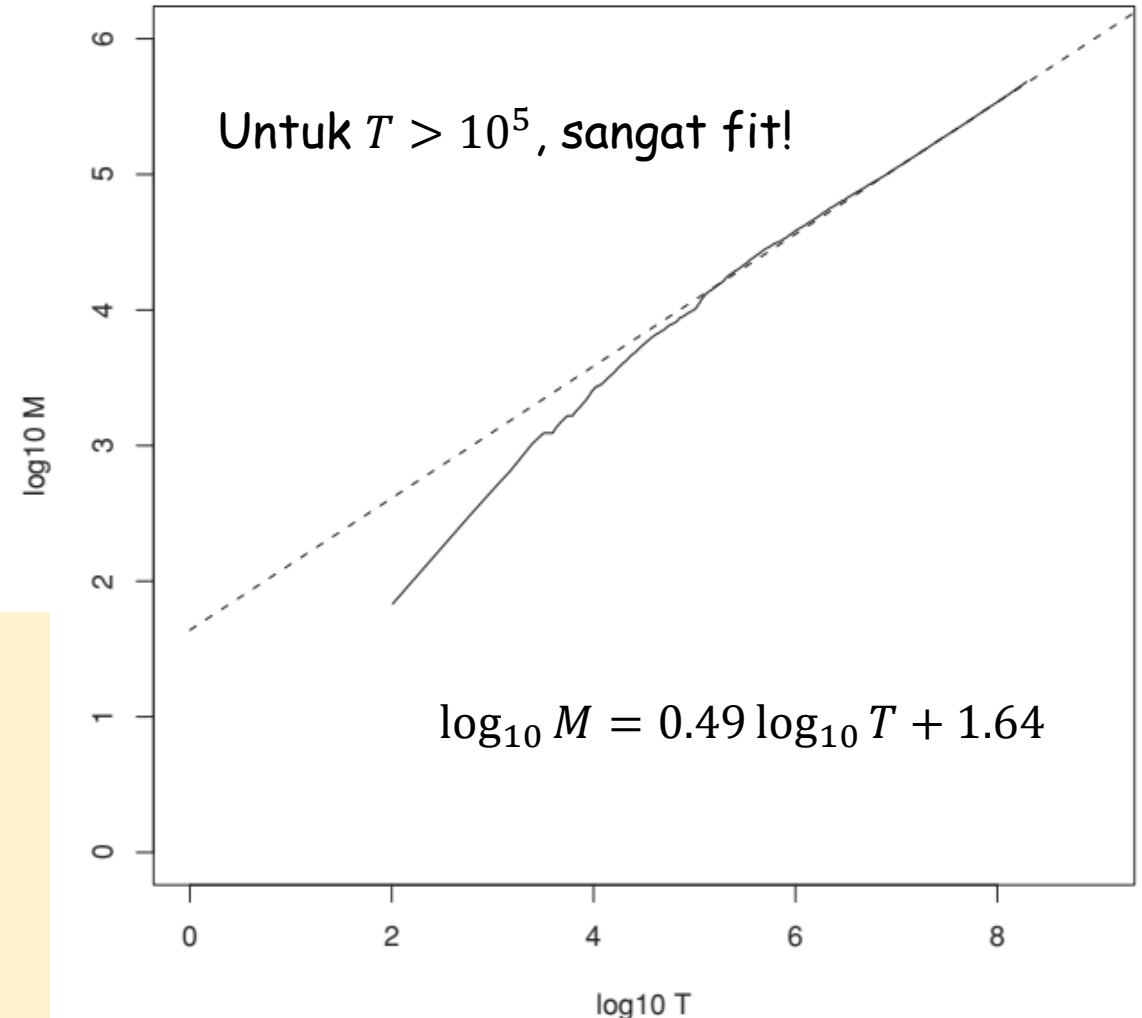
Reuters RCV1 Collection

- 800,000 documents
- ~400,000 terms
- On average 200 tokens / doc

Kesimpulan:

Heap's law mengindikasikan bahwa ukuran term vocab (dictionary) akan terus bertambah seiring bertambahnya dokumen di koleksi (**tidak ada batas atasnya**).

Jadi, **Dictionary Compression** adalah hal yang penting!



Contoh Empirical Evidence Heap's Law

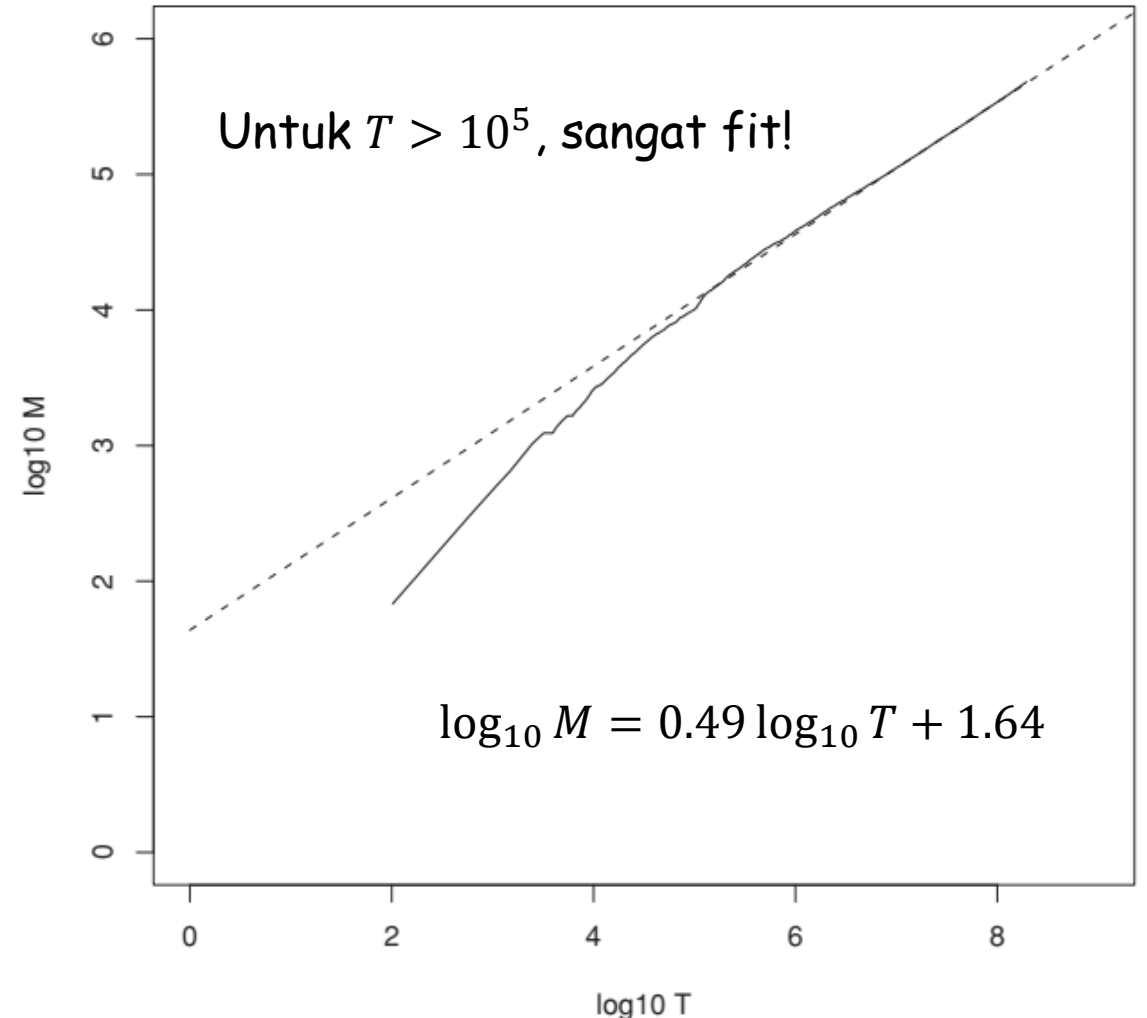
Contoh:

Untuk 1.000.020 token pertama,
Heap's law memprediksi ada
38.323 term unik.

$$44 \times 1.000.020^{0,49} \approx 38,323$$

Angka aslinya adalah 38.365.

Prediksi yang sangat baik!

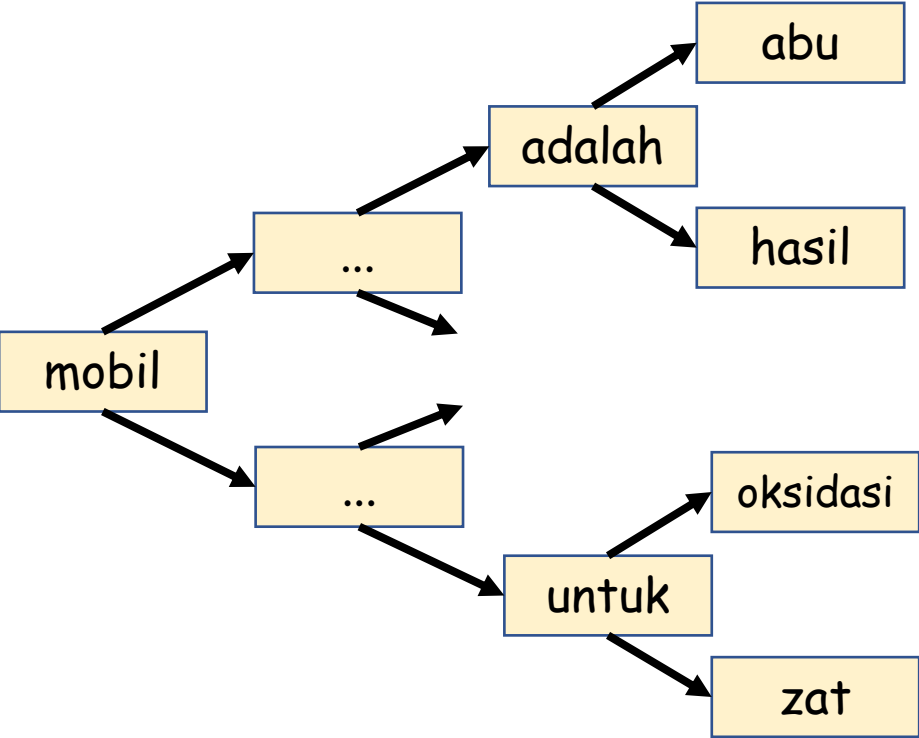


Dictionary Compression







Selain karena Heap's Law, kita juga ingin tetap menjaga agar Dictionary "sekecil mungkin" dan muat di memori.

Ingat search dimulai dari proses pencarian di Dictionary!

Uncompressed Dictionary



Dictionary Search Tree Structure
(Binary Search Tree atau sejenisnya)

Terms	Doc Freq.	Postings Pointer
abu	343	
adalah	574,991	
hasil	5045	
...
oksidasi	198	
untuk	487,249	
zat	349	
20 bytes	4 bytes	4 bytes

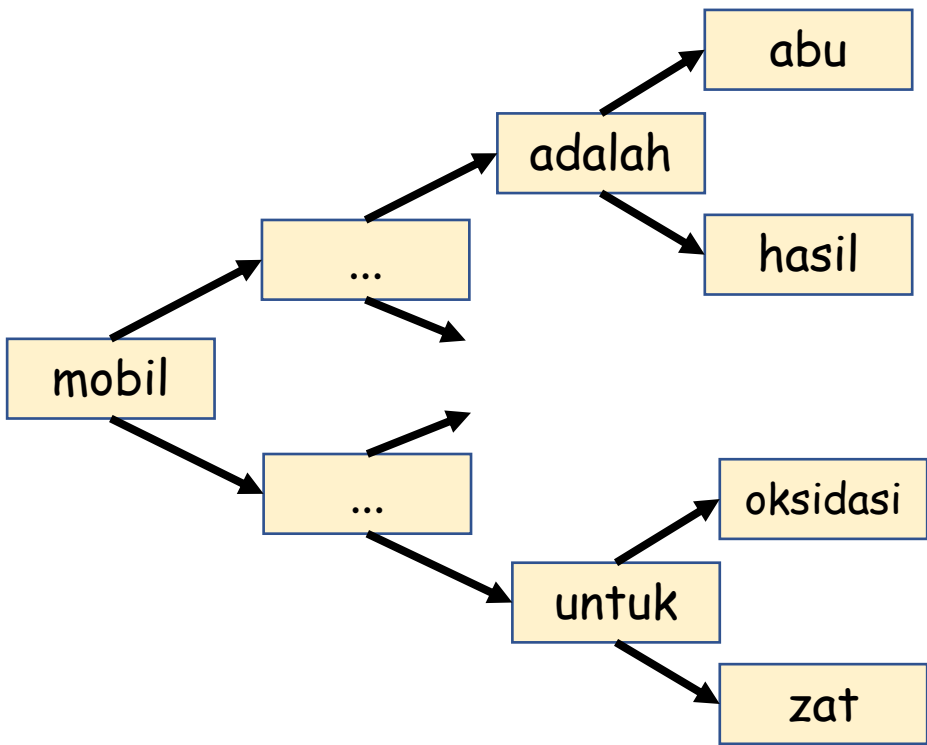
Postings Lists

Uncompressed Dictionary

Koleksi Reuters-RCV1

~400.000 terms; 28 bytes/term

Total = 11,2 MB



Dictionary Search Tree Structure
(Binary Search Tree atau sejenisnya)

- Di koleksi bahasa inggris, ada rata-rata 8 karakter per kata.
- Sebagian besar kata sangat pendek!
- Kata "supercalifragilisticexpialidocious" mungkin hanya muncul di < 10 dokumen.
- **Sebagian besar byte di kolom terms tidak terpakai!**

Terms	Doc Freq.	Postings Pointer
abu	343	
adalah	574,991	
hasil	5045	
...
oksidasi	198	
untuk	487,249	
zat	349	

Postings Lists

20 bytes 4 bytes 4 bytes

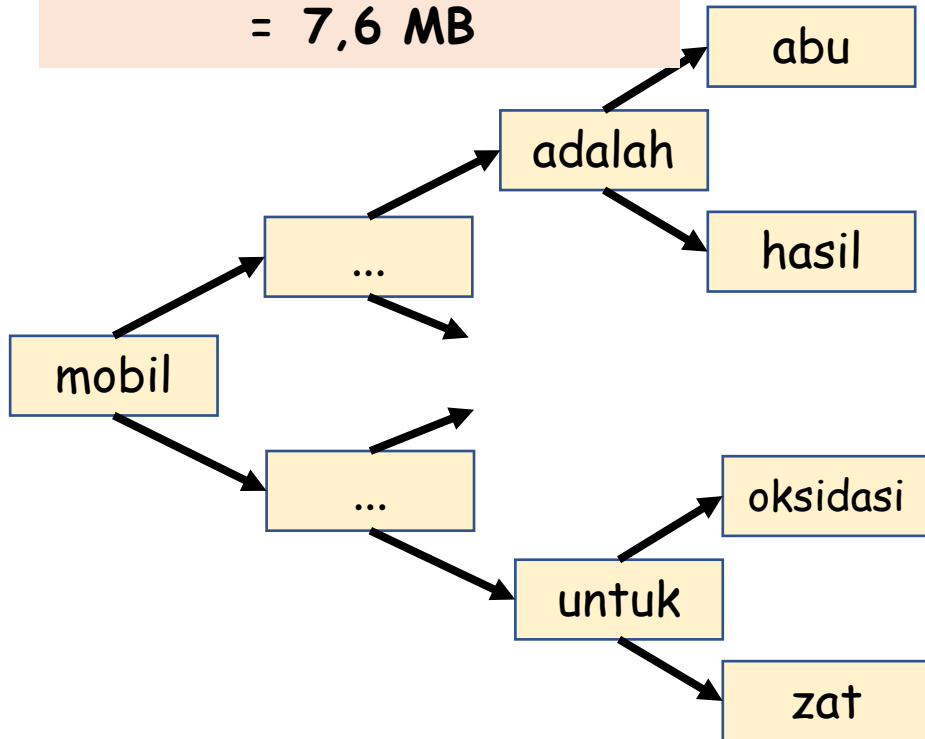
Dictionary-As-A-String (DaaS)

Koleksi Reuters-RCV1

~400.000 terms;

Satu term = 4 + 4 + 3 + 8

Total = 400K x 19 bytes
= **7,6 MB**



abuadalahhasil ... oksidasiuntukzat

Term Pointer	Doc Freq.	Postings Pointer
	343	
	574,991	
	5045	
...
	198	
	487,249	
	349	

3 bytes

4 bytes

4 bytes

Misal, 1 char = 1 byte

Ingat, rata-rata 8 karakter / kata = 8 bytes / kata.

Total string length:
400K x 8B = **3,2 MB**

Jadi, slot term pointer perlu muat 3,2M kemungkinan

$2^{\log 3,2M} = 22 \text{ bits}$
= 3 bytes

Panjang kata bisa disimpan di 1 byte saja. **Mengapa?**

DaaS with Blocking

Store pointers to **every** k-th term string!

Misal k = 4

Koleksi Reuters-RCV1

~400.000 terms;

Total = 7.1 MB

Berkurang ~0.5MB








Kok bisa? :) Ada yang bisa bantu hitung?

Bagaimana jika k = 10 & 20 ?

Mengapa tidak menggunakan k yang besar?

*visualisasi Binary Search Tree di-skip dahulu
(bukan dihilangkan)

3abu6adalah5hasil7halaman8inisiasi3itu ...

Term Pointer	Doc Freq.	Postings Pointer
	343	
	574,991	
	5045	
	835	
	231	
	498	
	67	

3 bytes

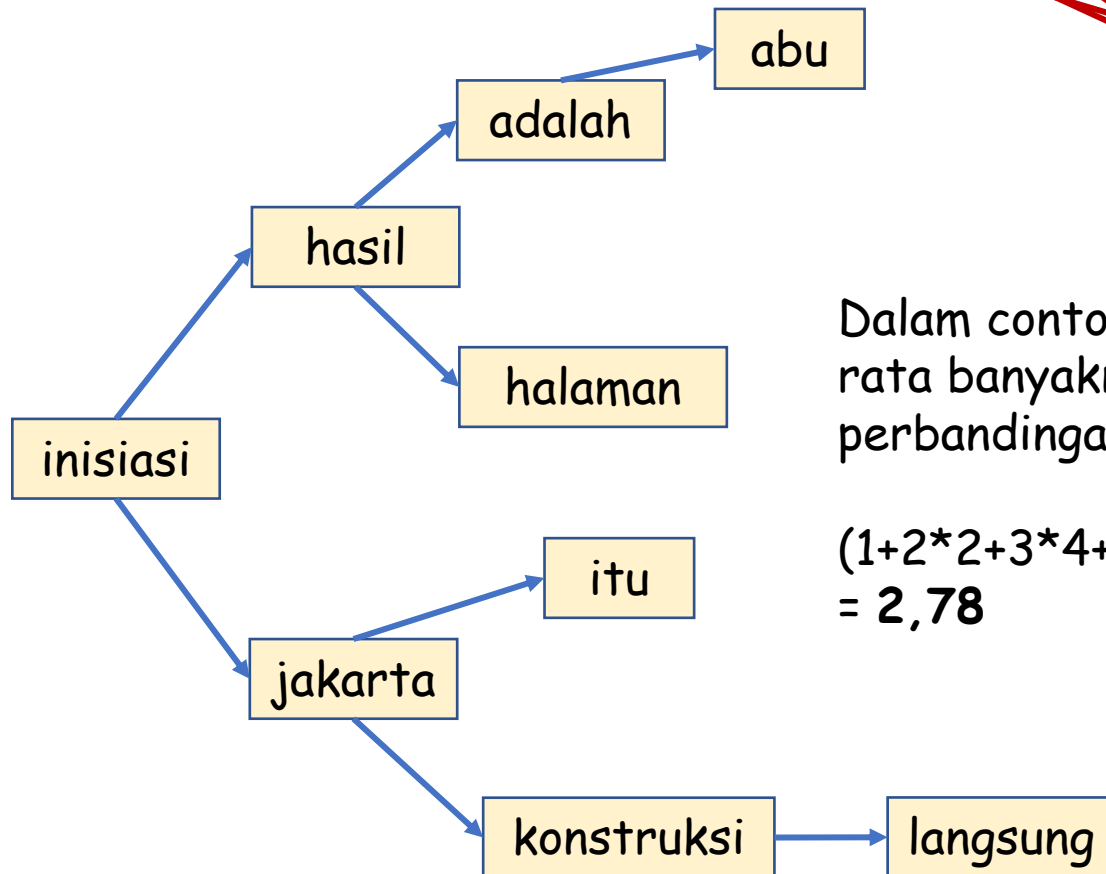
4 bytes

4 bytes

Hemat 3 lokasi
Hemat 9 bytes

Proses Search di Dictionary Ketika Tanpa Blocking

3abu6adalah5hasil7halaman8inisiasi3itu7jakarta10konstruksi8langsung

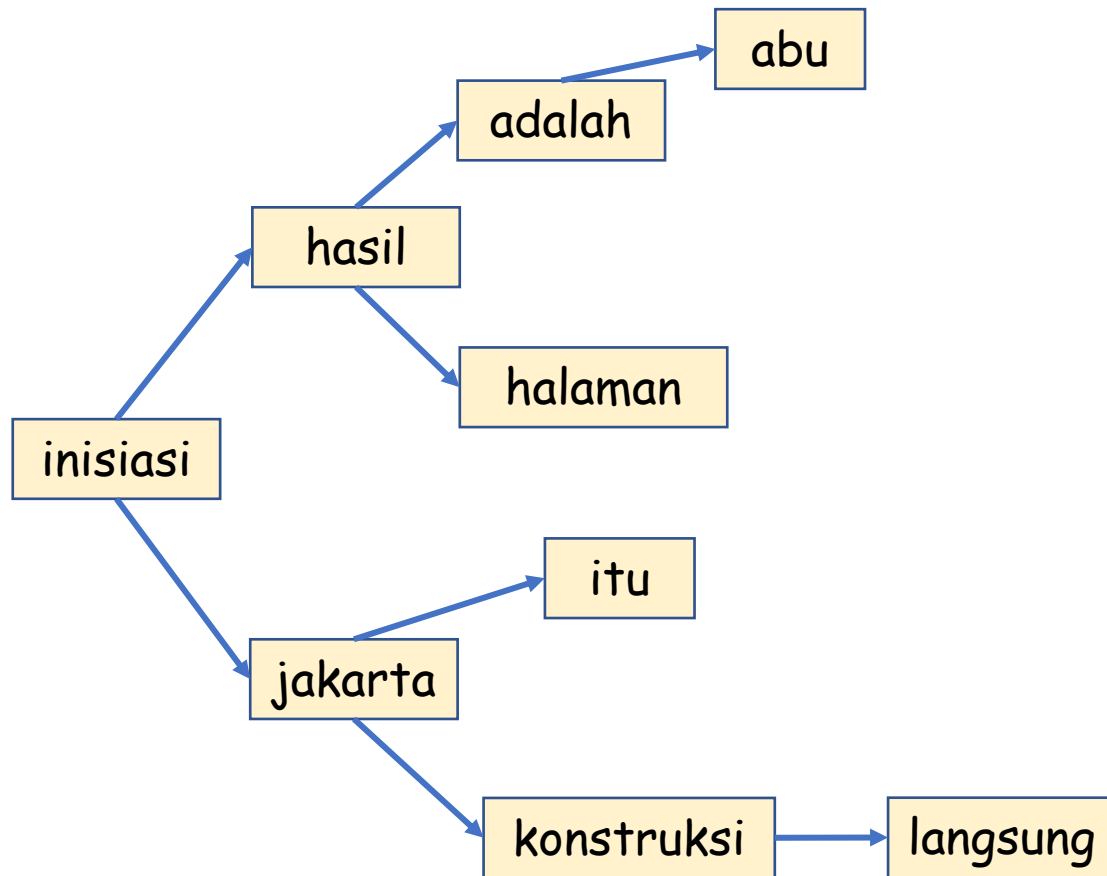


Dalam contoh ini, rata-rata banyaknya operasi perbandingan adalah

$$(1+2*2+3*4+4*2) / 9 = 2,78$$

Term Pointer	Doc Freq.	Postings Pointer
	343	
	574,991	
	5045	
	835	
	231	
	498	
	67	
	92	
	763	

Proses Search di Dictionary Ketika Tanpa Blocking



Ini adalah ketika kita asumsikan semua term mempunyai chance/probability yang sama untuk muncul di sebuah query.

Bagaimana jika tidak sama? Misal, term “adalah” muncul paling sering diantara semuanya di query.

OPTIONAL: Optimal Binary Search Trees

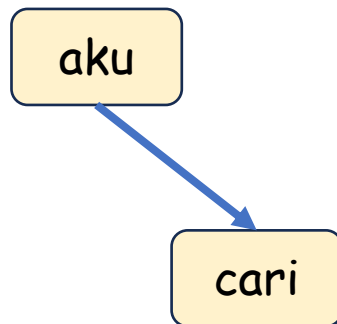
Yang mau eksplorasi masalah ini,
akan diberikan nilai 300 point

How to construct a binary search tree that minimizes the **expected search cost (ESC)**?

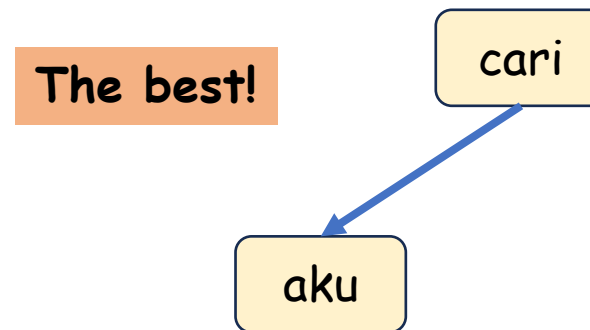
Input: `keys[] = {aku, cari}` `freqs[] = {20, 45}`

"aku" appears 20 times in the query logs, while "cari" is the most frequent keyword with 45 occurrences.

There can be following two possible BSTs:



$$ESC = \frac{20}{65} \times 1 + \frac{45}{65} \times 2 = 1.69$$



$$ESC = \frac{20}{65} \times 2 + \frac{45}{65} \times 1 = 1.30$$

OPTIONAL: Optimal Binary Search Trees

Yang mau eksplorasi masalah ini,
akan diberikan nilai 300 point

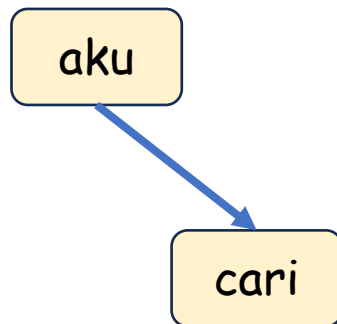
How to construct a binary search tree that minimizes the **expected search cost (ESC)**?

Input: `keys[] = {aku, cari}` `freqs[] = {20, 45}`

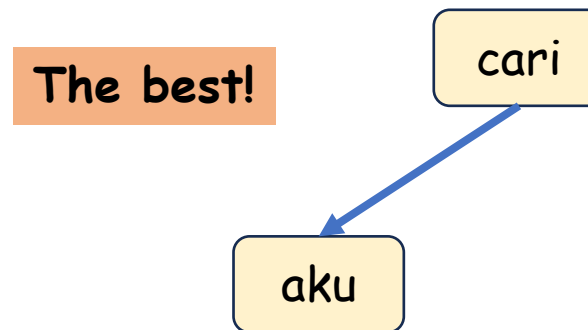
"aku" appears 20 times in the query logs, while "cari" is the most frequent keyword with 45 occurrences.

There can be following two possible BSTs:

For simplicity, we can just use the following cost function.



$$\text{Cost} = 20 \times 1 + 45 \times 2 = 110$$



The best!

$$\text{Cost} = 20 \times 2 + 45 \times 1 = 85$$

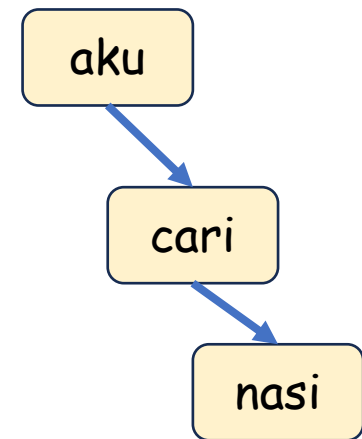
OPTIONAL: Optimal Binary Search Trees

Yang mau eksplorasi masalah ini,
akan diberikan nilai 300 point

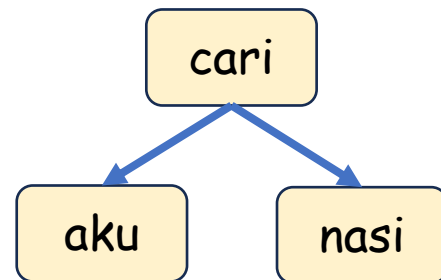
How to construct a binary search tree that minimizes the **expected search cost (ESC)**?

Input: `keys[] = {aku, cari, nasi}` `freqs[] = {10, 5, 20}`

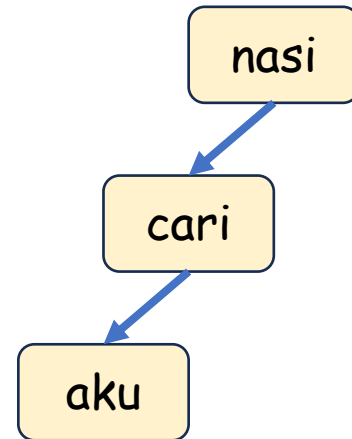
There can be following five possible BSTs:



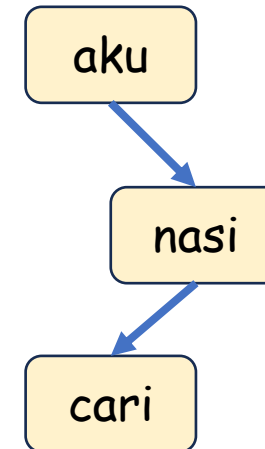
$$\begin{aligned}\text{Cost} &= 10 * 1 + \\ &\quad 5 * 2 + \\ &\quad 20 * 3 \\ &= 80\end{aligned}$$



$$\begin{aligned}\text{Cost} &= 5 * 1 + \\ &\quad 10 * 2 + \\ &\quad 20 * 2 \\ &= 65\end{aligned}$$

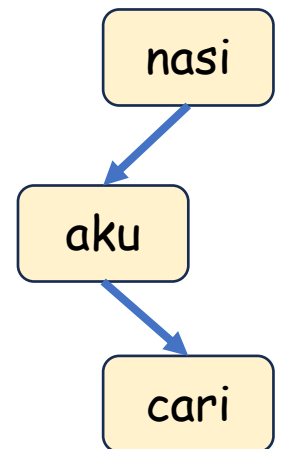


$$\begin{aligned}\text{Cost} &= 20 * 1 + \\ &\quad 5 * 2 + \\ &\quad 10 * 3 \\ &= 60\end{aligned}$$



$$\begin{aligned}\text{Cost} &= 10 * 1 + \\ &\quad 20 * 2 + \\ &\quad 5 * 3 \\ &= 65\end{aligned}$$

The best!



$$\begin{aligned}\text{Cost} &= 20 * 1 + \\ &\quad 10 * 2 + \\ &\quad 5 * 3 \\ &= 55\end{aligned}$$

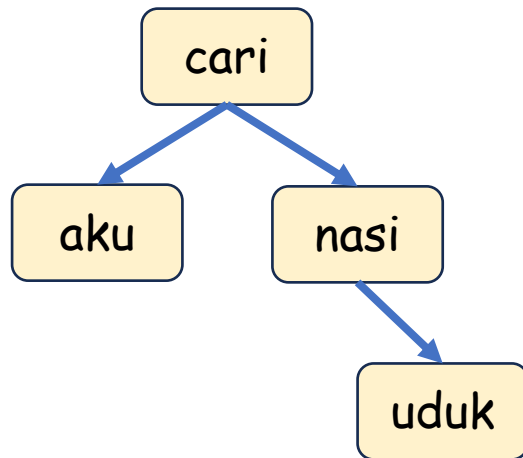
OPTIONAL: Optimal Binary Search Trees

Yang mau eksplorasi masalah ini,
akan diberikan nilai 300 point

This is an optimization problem. Define a cost function!

Input: `keys[] = {aku, cari, nasi, uduk}` `freqs[] = {10, 5, 20, 15}`

For example:



$$\begin{aligned} \text{Cost} &= 5 * 1 + \\ &\quad 10 * 2 + \\ &\quad 20 * 2 + \\ &\quad 15 * 3 \\ &= 110 \end{aligned}$$

=

$$\begin{aligned} \text{Cost} &= (10 + 5 + 20 + 15) + \\ &\quad (10 + 20 + 15) + \\ &\quad (15) \\ &= 110 \end{aligned}$$

$$\begin{aligned} \text{Cost}(i, j) &= \sum_{k=i}^j \text{freqs}[i] + \\ &\quad \text{Cost}(i, \text{root}(i, j) - 1) + \\ &\quad \text{Cost}(\text{root}(i, j) + 1, j) \end{aligned}$$

$\text{root}(i, j)$ mengembalikan indeks elemen yang merupakan **root** pada sekumpulan elemen dari indeks i hingga j

OPTIONAL: Optimal Binary Search Trees

Yang mau eksplorasi masalah ini,
akan diberikan nilai 300 point

This is an optimization problem. Define a cost function!

$$Cost(i, j) = \sum_{k=i}^j freqs[k] + \min_{r \in [i, j]} [Cost(i, r - 1) + Cost(r + 1, j)]$$

We one by one try all nodes as root r

OPTIONAL: Optimal Binary Search Trees

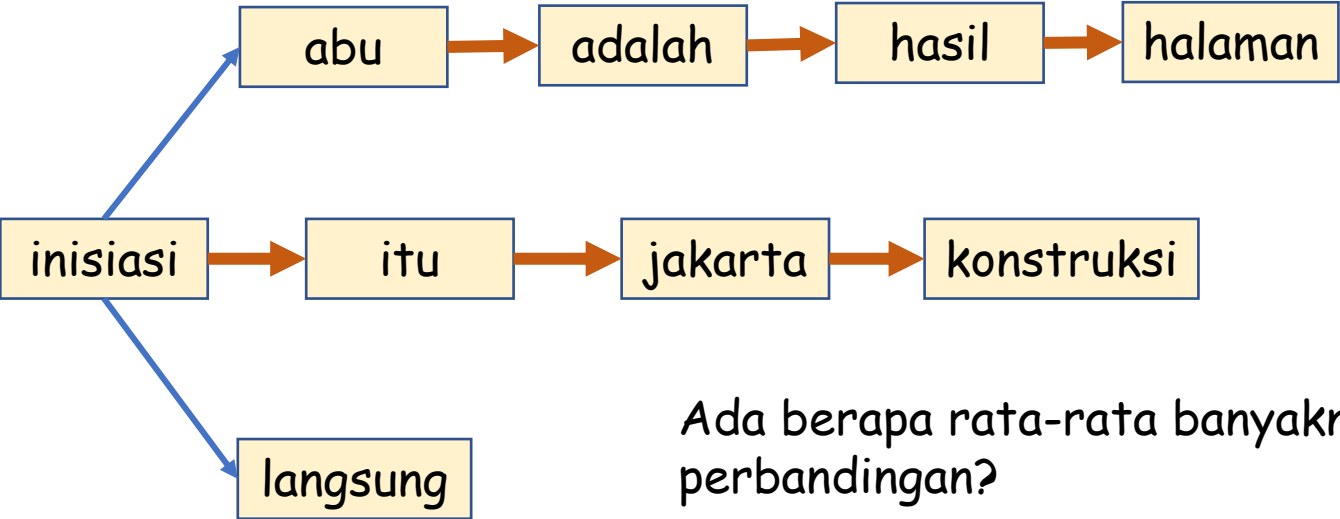
```
def opt_cost(freq, i, j):  
    if j < i:  
        return 0  
    if j == i:  
        return freq[i]  
  
    fsum = sum(freq[i:j+1])  
  
    min_cost = infinity  
    for r in range(i, j + 1):  
        cost = opt_cost(freq, i, r - 1) + opt_cost(freq, r + 1, j)  
        if cost < min_cost:  
            min_cost = cost  
  
    return min_cost + fsum  
  
freq = [10, 5, 20]  
print(opt_cost(freq, 0, 2))
```

1. Ini adalah versi rekursif yang tidak efisien; Anda perlu menggunakan Dynamic Programming
2. Solusi ini hanya mengembalikan nilai cost yang paling kecil; tidak mengembalikan struktur binary search tree-nya

Yang mau eksplorasi masalah ini, akan diberikan nilai 300 point

Proses Search di Dictionary DENGAN Blocking

3abu6adalah5hasil7halaman8inisiasi3itu7jakarta10konstruksi8langsung



Ada berapa rata-rata banyaknya perbandingan?

Yang pasti > 2,78

Term Pointer	Doc Freq.	Postings Pointer
	343	
	574,991	
	5045	
	835	
	231	
	498	
	67	
	92	
	763	

Proses search sudah tidak seperti search pada Binary Tree, tetapi ada proses Linear Search

Front Coding

Kita bisa memanfaatkan kesamaan prefix dari kata-kata yang sudah diurutkan secara leksikografis.

8automata8automate9automatic10automation...

Ada ide bagaimana cara compress String di atas?

Front Coding

Kita bisa memanfaatkan kesamaan prefix dari kata-kata yang sudah diurutkan secara leksikografis.

8automata8automate9automatic10automation...

8automat*a1#e2#ic3#ion...

Tanda * menandakan
automat adalah prefix.

Panjang karakter
tambahan setelah prefix
automat.

Sekilas Info Untuk Reuters-RCV1

Teknik Compression	Ukuran Dictionary
Uncompressed	11,2 MB
Dictionary-as-a-String without blocking	7,6 MB
Dictionary-as-a-String with blocking dengan $k = 4$	7,1 MB
Dictionary-as-a-String with blocking dengan $k = 4$ + Front Coding	5,9 MB
...	...

Postings Compression

Reuters-RCV1

- 800,000 documents
- 100M postings

$$\log_2(800000) \approx 20 \text{ bits}$$

Terms	Doc Freq.	Postings Pointer
abu	343	→
adalah	574,991	→
hasil	5045	→
...
oksidasi	198	→
untuk	487,249	→
zat	349	→

Berapa minimal bits yang dapat digunakan untuk menyimpan sebuah posting?

12	34	543	896	...
----	----	-----	-----	-----

1	3	4	5	...
---	---	---	---	-----

10	13	15	18	...
----	----	----	----	-----

45	62	78	987	...
----	----	----	-----	-----

1	3	5	7	...
---	---	---	---	-----

20	34	35	67	...
----	----	----	----	-----

Jadi, berapa ukuran **uncompressed** postings lists di koleksi Reuters-RCV1 (dalam bytes)?

Bisakah kita compress dan menggunakan < 20 bits / docID?

Compression Principles

- **Compressibility** dibatasi dengan *information content* dari sebuah **dataset**.
- *Information content* dari sebuah text T , yang terdiri dari rangkaian kata-kata $T = [w_1, w_2, w_3, \dots, w_n]$, dapat dihitung dengan menggunakan gagasan **Entropy** H :

$$H(T) = - \sum_{w \in T} \frac{f_w}{N(T)} \log_2 \left(\frac{f_w}{N(T)} \right)$$

f_w adalah frekuensi kemunculan kata w pada text T

$N(T)$ adalah panjang dari T

Compression Principles

Contoh: "the more dilligent the more success the better"

freq(the) = 3; freq(more) = 2;

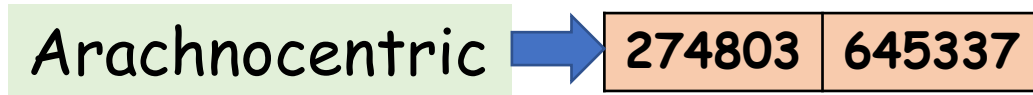
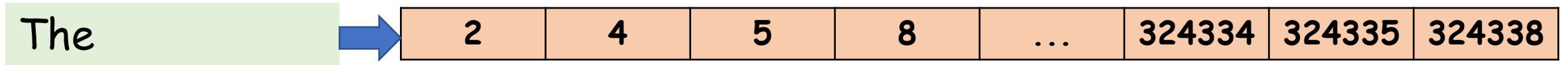
freq(dilligent) = freq(success) = freq(better) = 1

$$H(T) = -\frac{3}{8}\log_2\left(\frac{3}{8}\right) - \frac{2}{8}\log_2\left(\frac{2}{8}\right) - \frac{1}{8}\log_2\left(\frac{1}{8}\right) - \frac{1}{8}\log_2\left(\frac{1}{8}\right) - \frac{1}{8}\log_2\left(\frac{1}{8}\right) = 2,055 \text{ bits}$$

0.43 bits 0.5 bits 1.125 bits

Intuisi: Yang sering muncul, harus menggunakan sedikit bit; Yang jarang muncul, boleh menggunakan banyak bit

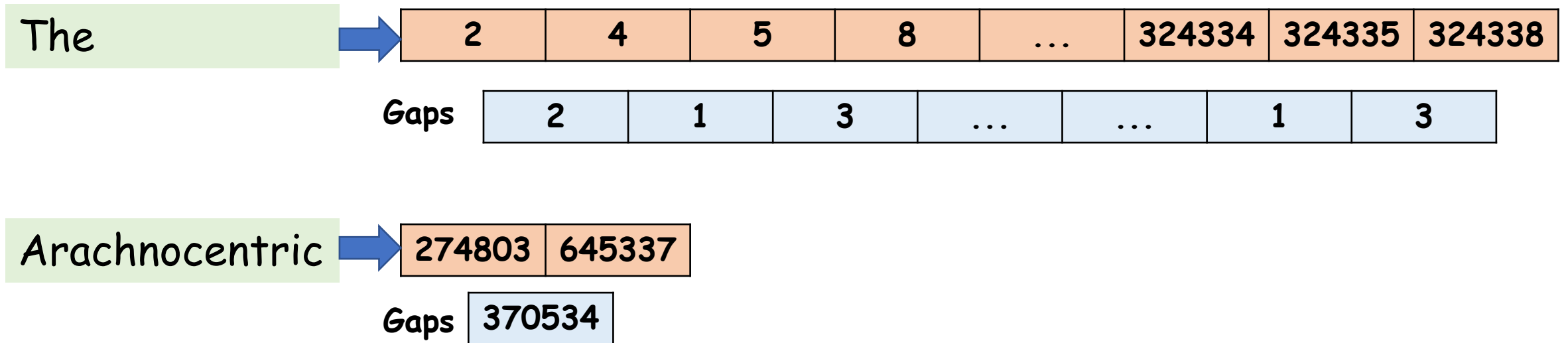
Term "the" vs term "arachnocentric"



Bagaimana agar bits untuk "The" < bits untuk "Arachnocentric"?

Kita mengamati bahwa posting untuk kata yang sering muncul seperti "the" sangat berdekatan.

Gaps between postings are shorter!

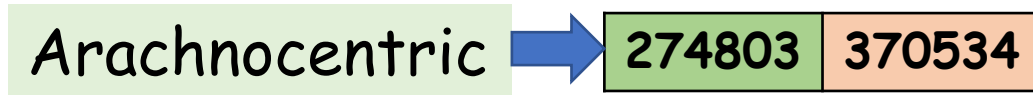
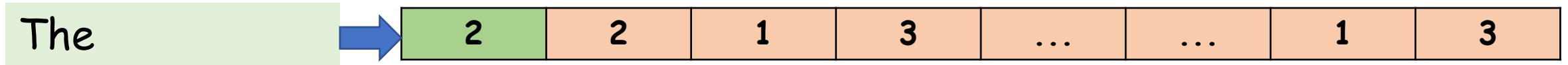


Bagaimana agar bits untuk "The" < bits untuk "Arachnocentric"?

Kita mengamati bahwa posting untuk kata yang sering muncul seperti "the" sangat berdekatan.

Sepertinya list of postings bisa diganti list of gaps 😊

* Kecuali posting yang pertama



Ok! Satu masalah solved 😊

Sekarang, muncul masalah baru. Bagaimana agar gap kecil 1,2,3 bisa di-encode dengan sedikit bit dibandingkan dengan gap besar seperti 3705434?

Variable Byte (VB) Compression

- Integers (gaps) are represented using **variable number of bytes**, as opposed to the fixed number of bytes.

Penanda untuk byte terakhir

- 1 byte --> 7 "payload" bit + 1 continuation bit


Integer Range	Number of Bytes
0 - 127	1
128 - 16383	2
16384 - 2097151	3
...	...

Contoh Angka	VB Codes
5	1 0000101
824	00000110 1 0111000
215406	00001101 00001100 1 0110001

VBENCODE(numbers)

VBENCODENUMBER(n)

```
1   $bytes \leftarrow \langle \rangle$ 
2  while  $true$ 
3  do PREPEND( $bytes, n \bmod 128$ )
4    if  $n < 128$ 
5      then BREAK
6     $n \leftarrow n \operatorname{div} 128$ 
7   $bytes[\operatorname{LENGTH}(bytes)] += 128$ 
8  return  $bytes$ 
```

 **atau $n \gg 7$**

VBENCODE($numbers$)

```
1   $bytestream \leftarrow \langle \rangle$ 
2  for each  $n \in numbers$ 
3  do  $bytes \leftarrow \text{VBENCODENUMBER}(n)$ 
4     $bytestream \leftarrow \text{EXTEND}(bytestream, bytes)$ 
5  return  $bytestream$ 
```

VBENCODE(numbers)

VBEncode(214577):

VBENCODENUMBER(n)

```
1   $bytes \leftarrow \langle \rangle$ 
2  while  $true$ 
3  do PREPEND( $bytes, n \bmod 128$ )
4    if  $n < 128$ 
5      then BREAK
6     $n \leftarrow n \operatorname{div} 128$ 
7   $bytes[\operatorname{LENGTH}(bytes)] += 128$ 
8  return  $bytes$ 
```

$n = 214577$

$bytes = []$

VBENCODE($numbers$)

```
1   $bytestream \leftarrow \langle \rangle$ 
2  for each  $n \in numbers$ 
3  do  $bytes \leftarrow \text{VBENCODENUMBER}(n)$ 
4     $bytestream \leftarrow \text{EXTEND}(bytestream, bytes)$ 
5  return  $bytestream$ 
```

VBENCODE(numbers)

VBENCODENUMBER(n)

```
1   $bytes \leftarrow \langle \rangle$ 
2  while  $true$ 
3  do PREPEND( $bytes, n \bmod 128$ )
4      if  $n < 128$ 
5          then BREAK
6       $n \leftarrow n \text{ div } 128$ 
7   $bytes[\text{LENGTH}(bytes)] += 128$ 
8  return  $bytes$ 
```

VBENCODE($numbers$)

```
1   $bytestream \leftarrow \langle \rangle$ 
2  for each  $n \in numbers$ 
3  do  $bytes \leftarrow \text{VBENCODENUMBER}(n)$ 
4       $bytestream \leftarrow \text{EXTEND}(bytestream, bytes)$ 
5  return  $bytestream$ 
```

VBEncode(214577):

$n = 214577$

$bytes = [00110001]$

$214577 \% 128 = 49 = 00110001$ (biner)

VBENCODE(numbers)

VBENCODENUMBER(n)

```
1   $bytes \leftarrow \langle \rangle$ 
2  while  $true$ 
3  do PREPEND( $bytes, n \bmod 128$ )
4    if  $n < 128$ 
5      then BREAK
6     $n \leftarrow n \operatorname{div} 128$ 
7   $bytes[\text{LENGTH}(bytes)] += 128$ 
8  return  $bytes$ 
```

VBENCODE($numbers$)

```
1   $bytestream \leftarrow \langle \rangle$ 
2  for each  $n \in numbers$ 
3  do  $bytes \leftarrow \text{VBENCODENUMBER}(n)$ 
4     $bytestream \leftarrow \text{EXTEND}(bytestream, bytes)$ 
5  return  $bytestream$ 
```

VBEncode(214577):

$n = 214577 // 128 = 1676$

$bytes = [00110001]$

$214577 \% 128 = 49 = 00110001$ (biner)

VBENCODE(numbers)

VBENCODENUMBER(n)

```
1  bytes  $\leftarrow \langle \rangle$ 
2  while true
3  do PREPEND(bytes,  $n \bmod 128$ )
4    if  $n < 128$ 
5      then BREAK
6     $n \leftarrow n \text{ div } 128$ 
7  bytes[LENGTH(bytes)] += 128
8  return bytes
```

VBENCODE($numbers$)

```
1  bytestream  $\leftarrow \langle \rangle$ 
2  for each  $n \in numbers$ 
3  do bytes  $\leftarrow$  VBENCODENUMBER( $n$ )
4    bytestream  $\leftarrow$  EXTEND(bytestream, bytes)
5  return bytestream
```

VBEncode(214577):

$n = 1676$

bytes = [00001100 00110001]

$214577 \% 128 = 49 = 00110001$ (biner)

$1676 \% 128 = 12 = 00001100$

VBENCODE(numbers)

VBENCODENUMBER(n)

```
1  bytes  $\leftarrow \langle \rangle$ 
2  while true
3  do PREPEND(bytes,  $n \bmod 128$ )
4    if  $n < 128$ 
5      then BREAK
6     $n \leftarrow n \div 128$ 
7  bytes[LENGTH(bytes)] += 128
8  return bytes
```

VBENCODE($numbers$)

```
1  bytestream  $\leftarrow \langle \rangle$ 
2  for each  $n \in numbers$ 
3  do bytes  $\leftarrow$  VBENCODENUMBER( $n$ )
4    bytestream  $\leftarrow$  EXTEND(bytestream, bytes)
5  return bytestream
```

VBEncode(214577):

$n = 1676 // 128 = 13$

bytes = [00001100 00110001]

$214577 \% 128 = 49 = 00110001$ (biner)

$1676 \% 128 = 12 = 00001100$

VBENCODE(numbers)

VBENCODENUMBER(*n*)

```
1  bytes ← ⟨⟩
2  while true
3  do PREPEND(bytes, n mod 128)
4      if n < 128
5          then BREAK
6      n ← n div 128
7  bytes[LENGTH(bytes)] += 128
8  return bytes
```

VBENCODE(*numbers*)

```
1  bytestream ← ⟨⟩
2  for each n ∈ numbers
3  do bytes ← VBENCODENUMBER(n)
4      bytestream ← EXTEND(bytestream, bytes)
5  return bytestream
```

VBEncode(214577):

n = 13

bytes = [00001101 00001100 00110001]

214577 % 128 = 49 = 00110001 (biner)

1676 % 128 = 12 = 00001100

13 % 128 = 13 = 00001101

VBENCODE(numbers)

VBENCODENUMBER(*n*)

```
1  bytes ← ⟨⟩
2  while true
3  do PREPEND(bytes, n mod 128)
4    if n < 128
5      then BREAK
6    n ← n div 128
7  bytes[LENGTH(bytes)] += 128
8  return bytes
```

VBENCODE(*numbers*)

```
1  bytestream ← ⟨⟩
2  for each n ∈ numbers
3  do bytes ← VBENCODENUMBER(n)
4    bytestream ← EXTEND(bytestream, bytes)
5  return bytestream
```

VBEncode(214577):

n = 13

bytes = [00001101 00001100 00110001]

214577 % 128 = 49 = 00110001 (biner)

1676 % 128 = 12 = 00001100

13 % 128 = 13 = 00001101

Stop While Loop

VBENCODE(numbers)

VBENCODENUMBER(*n*)

```
1  bytes ← ⟨⟩
2  while true
3  do PREPEND(bytes, n mod 128)
4    if n < 128
5      then BREAK
6    n ← n div 128
7  bytes[LENGTH(bytes)] += 128
8  return bytes
```

VBENCODE(*numbers*)

```
1  bytestream ← ⟨⟩
2  for each n ∈ numbers
3  do bytes ← VBENCODENUMBER(n)
4    bytestream ← EXTEND(bytestream, bytes)
5  return bytestream
```

VBEncode(214577):

n = 13

bytes = [00001101 00001100 10110001]

214577 % 128 = 49 = 00110001 (biner)

1676 % 128 = 12 = 00001100

13 % 128 = 13 = 00001101

Bit pertama di byte terakhir diganti 1

=

Nilainya + 128

VBDECODE(numbers)

VBDECODE(*bytestream*)

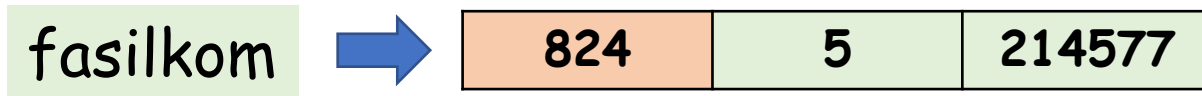
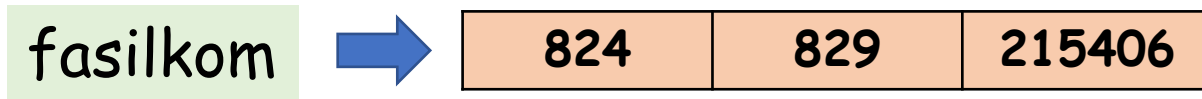
```
1  numbers  $\leftarrow \langle \rangle$ 
2  n  $\leftarrow 0$ 
3  for i  $\leftarrow 1$  to LENGTH(bytestream)
4  do if bytestream[i] < 128
5      then n  $\leftarrow 128 \times n + \text{bytestream}[i]$ 
6      else n  $\leftarrow 128 \times n + (\text{bytestream}[i] - 128)$ 
7          APPEND(numbers, n)
8          n  $\leftarrow 0$ 
9  return numbers
```

[00001101 00001100 10110001]

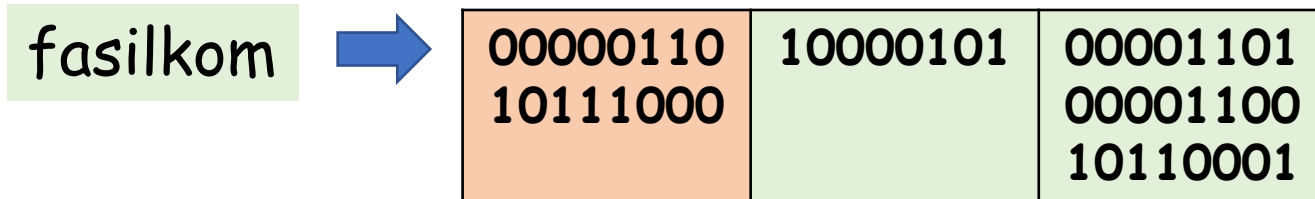


214577

Jadi, Bagaimana Akhirnya?

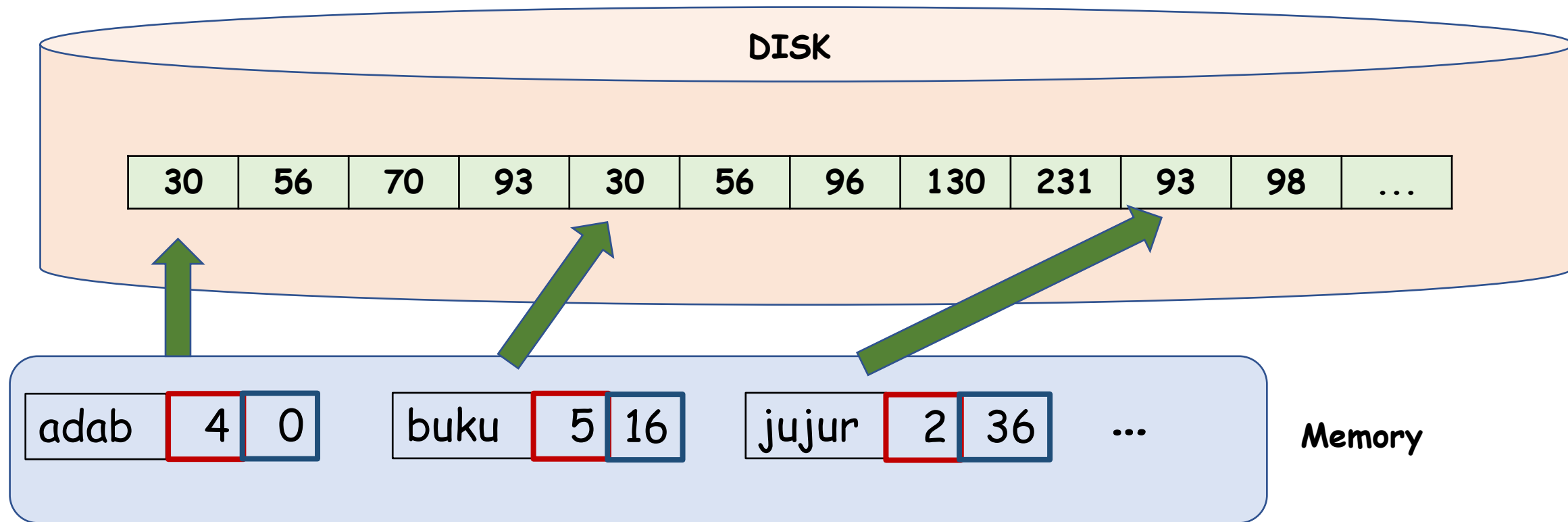


Ubah ke list of gaps

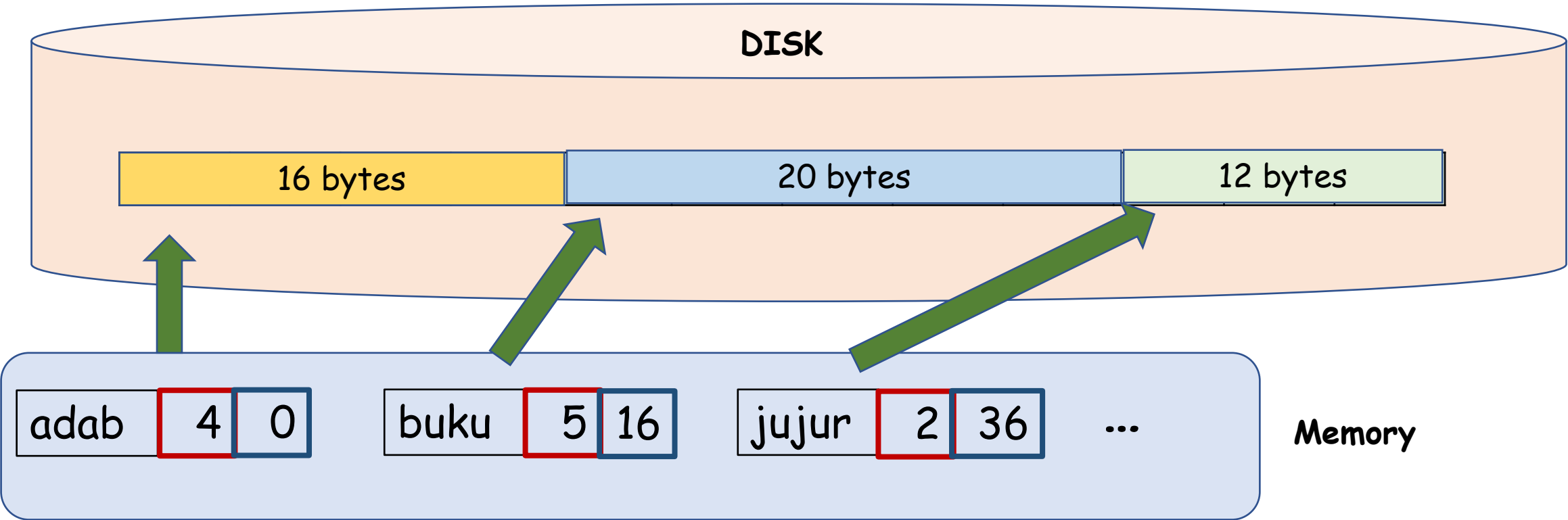


Encode dengan teknik compression,
misal VB encoding

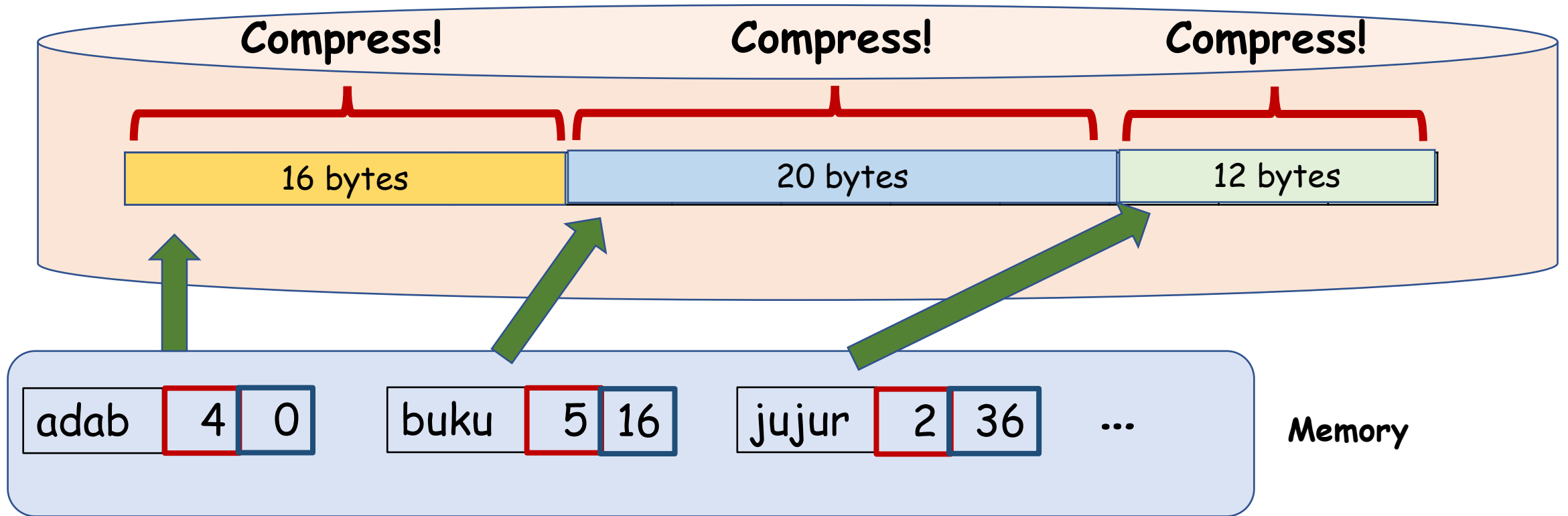
Barulah, setelah itu disimpan di storage. Saat membaca, encoded bits perlu di-decode dahulu sebelum dikembalikan ke bentuk list of docIDs asli.



Uncompressed, dengan docID disimpan di 4 byte-slot

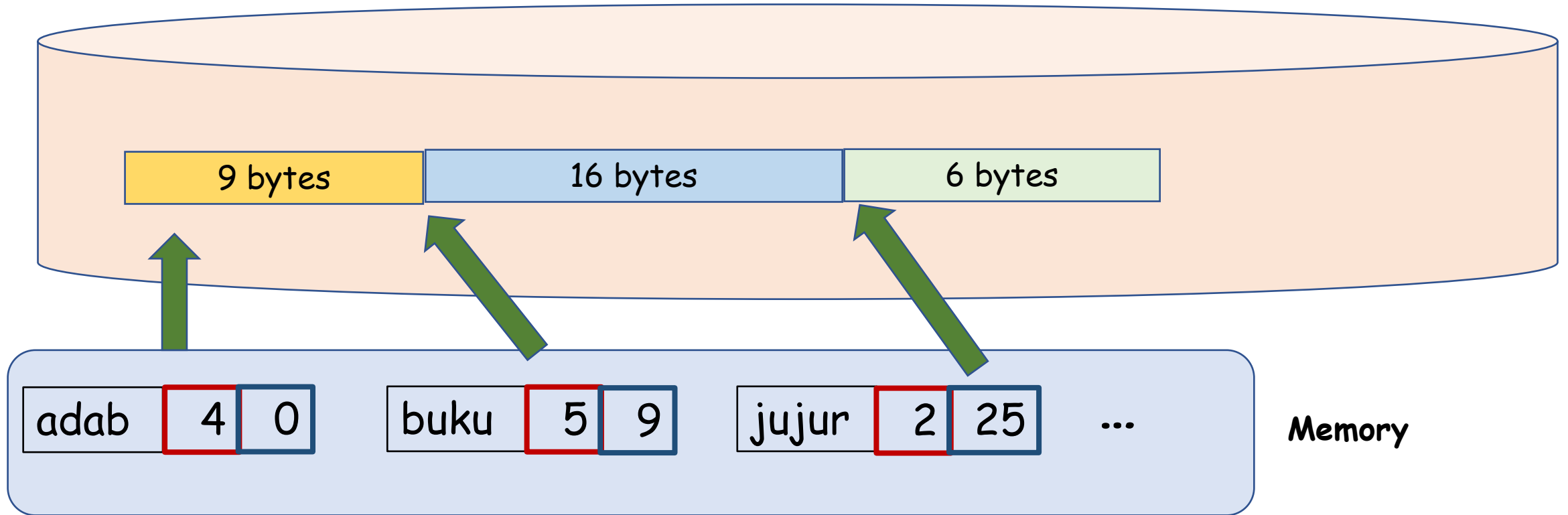


Sekarang, setiap list of postings akan dikompresi



Kita tidak kompresi gabungan semua lists of postings.
Kita kompresi masing-masing list of postings secara terpisah!

Compressed



Kita tidak kompresi gabungan semua lists of postings.
Kita kompresi masing-masing list of postings secara terpisah!

OptPForDelta Compression

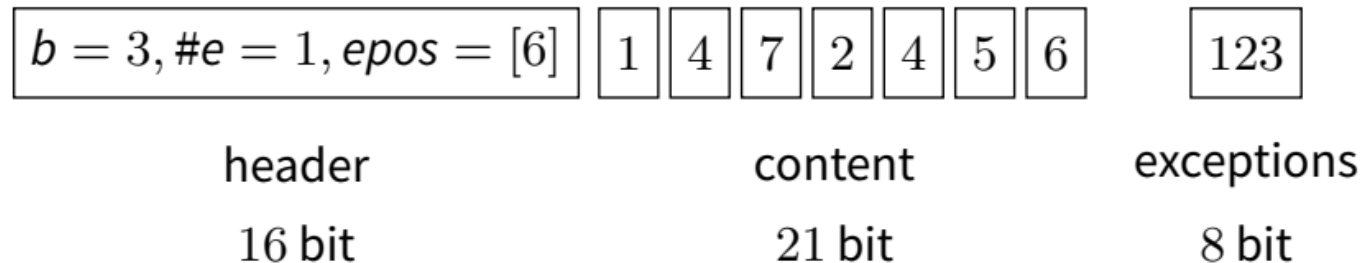
Kelompokkan **K** buah gaps dan encode dengan fixed-number of bits **b**.
Encode angka yang $> 2^b$ secara terpisah sebagai sebuah exception.

Bagaimana cara pilih **b**? pilih agar kira-kira untuk setiap group ada **sekitar kira-kira 10% exceptions**.

Contoh untuk $K = 8$

[1, 2, 4, 4, 5, 6, 7, 123]

Kita pilih $b = 3$



Algoritma Compression Yang Lain

- Elias-Gamma
- Simple-8b
- Simple-9
- SIMD-BP128
- ...

Nilai partisipasi:

Silakan pilih salah satu dan lakukan kajian singkat bagaimana cara kerjanya!

Dan apa bedanya dengan VB Encoding?

Elias-Gamma Coding

How to encode positive integers X ?

- Find the largest N , with $2^N \leq X$
- Encode N using unary coding --> N zeroes followed by one
- Convert $X - 2^N$ into N digits in binary; and append them

Elias-Gamma Coding

How to encode 19 ?

- $2^4 \leq 19$ or $2^4 + 3 = 19$
- $N = 4$ ----> 00001
- In 4 digit-binary, 3 is 0011
- So the Gamma code for 19 is 000010011

Elias-Gamma Coding

Nomor	Binary	Coding
$1 = 2^0 + 0$	1	1
$2 = 2^1 + 0$	10	0 1 0
$3 = 2^1 + 1$	11	0 1 1
$4 = 2^2 + 0$	100	00 1 00
$5 = 2^2 + 1$	101	00 1 01
$6 = 2^2 + 2$	110	00 1 10
$7 = 2^2 + 3$	111	00 1 11
$8 = 2^3 + 0$	1000	000 1 000
$9 = 2^3 + 1$	1001	000 1 001
$10 = 2^3 + 2$	1010	000 1 010

Elias-Gamma Coding

Cara decoding?

- Dari kiri, baca terus nol dan hitung jumlahnya. Jumlah tersebut adalah N .
- Kemudian, baca N bit kedepan, dan ubah bit tersebut ke decimal. Angka decimal tersebut adalah K .

- $X = 2^N + K$

Elias-Gamma vs VB coding

- "In terms of **compression rate**, numbers less than 16 are best encoded using Elias-gamma, numbers larger than 16 are best encoded using **Variable Byte**."
- Mengapa?

Byte-Level vs Bit-Level Coding

- Bit-level coding biasanya menghasilkan hasil kompresi yang lebih baik.
- Namun, bit-level coding memerlukan operasi bit manipulation yang banyak (karena terbentur machine word, yang biasanya 8 bit, 16 bit, ...). Jadi, biasanya proses coding jadi lebih lambat.

Space Usage & Speeds

Koleksi GOV2

	Coding Speed (millions of integers / second)	Decoding Speed (millions of integers / second)	Space (Bits / int)
Variable Byte Coding (Byte-Level)	730	680	8.7
OptPForDelta Coding (Bit-Level)	23	710	4.5

Koleksi ClueWeb09

	Coding Speed (millions of integers / second)	Decoding Speed (millions of integers / second)	Space (Bits / int)
Variable Byte Coding (Byte-Level)	570	540	9.6
OptPForDelta Coding (Bit-Level)	14	500	7.1