6. Index compression
What we have seen so far
Boolean retrieval

Input
Set of documents

Output
Subset of documents

query

lawyer AND Penang AND NOT silver
Standard inverted index

- ETH: 6 1 2 3 5 6 8
- Zürich: 5 3 4 7 8 9
- computer: 5 1 2 4 5 7
- data: 5 1 3 5 8 9
- CPU: 4 2 3 4 7
- information: 6 1 2 4 5 8 9
- retrieval: 4 3 5 7 8
Search structures

<table>
<thead>
<tr>
<th>Hash tables</th>
<th>Trees (B, B+)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Additional features

Wildcards

comput*

Spell correction

computer

"Pfäffikon SZ"

Phrase search
Help ETH Zurich to flexibly react to new challenges and to set new accents in the future.
Positional index (phrase search feature)

"ETH Zurich"

Help
ETH
Zurich
to
flexibly
react
Trigram index (wildcard, spell correction)
TermIDs

t1 1

t2 3

t3 1

t4 1

t5 2

t6 1

t7 3

t1 2

t2 4

t3 2

t4 3

t5 3

t6 2

t7 5

t1 3

t2 7

t3 4

t4 5

t5 4

t6 4

t7 7

...
Blocked Sort-Based Indexing
Single-Pass In-Memory Indexing
Auxiliary Index

- ETH
- Computer
- Information
- Course

Main index

Auxiliary index
Logarithmic Merging

n postings  →  2n postings  →  4n postings

Z₀  →  Z₁

I₀  →  I₂

Images of computer memory modules and hard drives are shown in the background.
Term Statistics
Number of terms
Number of terms
Notations used in the book

N: number of documents

T: number of tokens (positional postings)

M: number of terms (or types if stemming/lemmatization)
Number of terms
Number of terms

# Terms

# Tokens

?
Number of terms

# Terms

# Tokens

?
Number of terms

# Terms

# Tokens
Number of terms

# Terms

# Tokens

max
Number of terms

We ❤ when it's linear
Log-log scale

We love when it's linear
Log-log scale

\[ \log M = b \log T + a \]

We ♥ when it's linear
"Exponential" growth

\[ M = e^{aT^b} \]
Heaps' law

\[ M = kT^b \]
In practice

\[ M = kT^b \]

\[ b \approx \frac{1}{2} \]
In practice

\[ M = k\sqrt{T} \]
In practice

\[ M = k \sqrt{T} \]

30 \leq k \leq 100
Distribution of terms
Distribution of terms

the: 56,271,872

were: 3,323,884

nearer: 51,456

moderate: 19,245

champion: 9400

stocks: 6,537

parallelogram: 503

pachyderm: 79

capacitance: 45

 germanium: 12

sesquipedal: 7
Distribution of terms

# Tokens

- the
- of
- and
- to
- in
- I was
Distribution of terms

# Tokens

Rank
log-log scale
Zipf's law

\[ \log \text{Frequency} = a \log \text{Rank} + b \]
Zipf's law

\[
\log \text{Frequency} = a \log \text{Rank} + b
\]
Zipf's law

\[ \log \text{Frequency} = b - \log \text{Rank} \]
Zipf's law

\[ \text{Frequency} = \frac{k}{\text{Rank}} \]
Compression techniques already covered
Compression techniques already covered

1 2 3 4 5   Remove numbers
Compression techniques already covered

1 2 3 4 5  Remove numbers

Apple  apple  Case folding
Compression techniques already covered

1 2 3 4 5 \hspace{1cm} \text{Remove numbers}

\text{Apple} \quad \rightarrow \quad \text{apple} \hspace{1cm} \text{Case folding}

\text{and of the} \hspace{1cm} \text{Remove stopwords}
Compression techniques already covered

1-2-3-4-5  Remove numbers

Apple ➔ apple  Case folding

and of the  Remove stopwords

computing ➔ compute  Stemming
Compression techniques already covered

- Remove numbers
- Case folding
- Remove stopwords
- Stemming

This reduces the size of the dictionary!
Impact (number of terms/types)

1-2-3-4-5
Remove numbers -2%

Apple → apple
Case folding -17%

and of the
Remove stopwords -0%

computing → compute
Stemming -17%

Source: Information Retrieval book
Impact (number of nonpositional postings)

- Remove numbers: -8%
- Case folding: -3%
- Remove stopwords: -30%
- Stemming: -4%

Source: Information Retrieval book
Impact (number of tokens = positional postings)

- Remove numbers: -9%
- Case folding: -0%
- Remove stopwords: -47%
- Stemming: -0%

Source: Information Retrieval book
Dictionary compression
## Standard inverted index

<table>
<thead>
<tr>
<th>Term</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETH</td>
<td>6</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zürich</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>computer</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>data</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>information</td>
<td>6</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>retrieval</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Let us start compressing the dictionary.
Status quo
Status quo: Dictionary stored as a B+ tree
Status quo: Dictionary stored as a B+ tree

Points to postings lists
Status quo: Dictionary stored as a B+ tree

Pointers to postings lists
Let us start compressing the dictionary.
We can then make it fit in RAM.
Approach 1: Array

computer.....  6
CPU...........  5
data..........  5
ETH............  5
information..  4
retrieval.....  6
Zürich.......  4
Approach 1: Array

- computer: 6 bytes
- CPU: 5 bytes
- data: 5 bytes
- ETH: 5 bytes
- information: 4 bytes
- retrieval: 6 bytes
- Zürich: 4 bytes

20 bytes, 4 bytes, 4 bytes
Approach 1: Issue

- computer..... 6
- CPU.......... 5
- data........ 5
- ETH.......... 5
- information.. 4
- retrieval.... 6
- supercalifragilisticexpialidocious
Approach 2: String

computerCPUdataETHinformationretrievalZürich
Approach 2: String

computerCPUdataETHinformationretrievalZürich

6 bytes 4 bytes
Approach 2: String

6 bytes
5 bytes
5 bytes
5 bytes
4 bytes
6 bytes
4 bytes
4 bytes
3 bytes

computer CPU data ETH information retrieval Zürich
Approach 2: String

6 bytes
5 bytes
5 bytes
5 bytes
4 bytes
6 bytes
4 bytes

4 bytes
4 bytes
3 bytes

computer
CPU
data
ETH
information retrieval
Zürich

(+8 bytes)
Approach 3: Blocked storage

8computer3CPU4data3ETH11information9retrieval...

Only every k terms

\[ \frac{3}{k} \text{ bytes} \]

(+9 bytes)

4 bytes 4 bytes

4 5 5 5 5 4 6 4
No free lunch
No free lunch
No free lunch
Compromise between space and time
Binary search steps (no blocking)
Binary search steps (no blocking)

One extra "memory seek"
Binary search steps (no blocking)

Two extra "memory seeks"
Binary search steps (no blocking)

Average: $\text{avg}(0, 1, 2, 2, 1, 2, 2) = 1.4$
Binary search steps (with blocking)
Binary search steps (with blocking)

Two extra "memory seeks"
Binary search steps (with blocking)

Three extra "memory seeks"
Binary search steps (with blocking)

Average: \( \text{avg}(0,1,2,3,1,2,3) = 1.7 \)
Approach 4: Front coding

- 6
- 5
- 5
- 5
- 4
- 4

4 bytes 4 bytes 4 bytes

\[ \frac{3}{k} \] bytes

(\(+9\) bytes)

Only every \(k\) terms

8automata 8automate 9automatic 10automation
Approach 4: Front coding

Only every \( k \) terms

\[
\frac{3}{k} \text{ bytes}
\]

(less bytes)
How did we do?

Collection: 960 MB

Source: Information Retrieval book
How did we do?

Fixed Width

Collection: 960 MB

Source: Information Retrieval book

11.2 MB
How did we do?

Fixed Width

Unique string and pointers

Collection: 960 MB

11.2 MB

7.6 MB

Source: Information Retrieval book
How did we do?

- Fixed Width
- Unique string and pointers
- Blocking (k=4)

Collection: 960 MB

Source: Information Retrieval book
How did we do?

Fixed Width

Unique string and pointers

Blocking (k=4)

Blocking and front coding

Collection: 960 MB

Source: Information Retrieval book
Postings file compression
Standard inverted index

- ETH: 6 → 1 → 2 → 3 → 5 → 6 → 8
- Zürich: 5 → 3 → 4 → 7 → 8 → 9
- computer: 5 → 1 → 2 → 4 → 5 → 7
- data: 5 → 1 → 3 → 5 → 8 → 9
- CPU: 4 → 2 → 3 → 4 → 7
- information: 6 → 1 → 2 → 4 → 5 → 8 → 9
- retrieval: 4 → 3 → 5 → 7 → 8
Standard inverted index

We compressed this...
Standard inverted index

Now, we want to compress this.
Standard inverted index

In other words, we want to compress lists of integers.
Standard storage

1 2 3 5 6 8
4 bytes 4 bytes 4 bytes 4 bytes 4 bytes 4 bytes
Standard storage

4 bytes 4 bytes 4 bytes 4 bytes 4 bytes 4 bytes 4 bytes

(4 bytes = 32 bits)
Standard storage

4 bytes 4 bytes 4 bytes 4 bytes 4 bytes 4 bytes
(4 bytes = 32 bits)

Numbers between 0 and 4,294,967,296
Encoding gaps

4 bytes 4 bytes 4 bytes 4 bytes 4 bytes 4 bytes
(4 bytes = 32 bits)

Can we encode with less space?
Encoding gaps
Encoding gaps

0000010000100000 → 0000010000100010 → 0000010000100111 → 0000010000101000 → 0000010000101011
Encoding gaps

These are small gaps!

0000010000100000 0000010000100010 0000010000100111 0000010000101000 0000010000101011
Encoding gaps

1056 +2 1058 +5 1063 +1 1064 +3 1067
Encoding gaps

0000010000100000

1056 +2 1058 +5 1063 +1 1064 +3 1067
Encoding gaps

1056 +2 1058 +5 1063 +1 1064 +3 1067

0000010000100000 0010
0101
0001
0011
Encoding gaps

But this only works for frequent terms!
Encoding gaps

Can we have variable gap size?
Variable byte encoding
Fix-length encoding

We know exactly where the boundaries are.
Fix-length encoding

We know exactly where the boundaries are.

010011010100010101001010100101011110010001010100101010101010...
Fix-length encoding

We know exactly where the boundaries are.
Fix-length encoding

We know exactly where the boundaries are.

32 bits

0100110101000101010100101011110010001010100101010101010101010...
Fix-length encoding

We know exactly where the boundaries are.

32 bits 32 bits
01001101010001010101001011101000101010010101010101010101010...
We do not know \textit{a priori} where the boundaries are.

<table>
<thead>
<tr>
<th></th>
<th>1 bytes</th>
<th>2 bytes</th>
<th>4 bytes</th>
<th>3 bytes</th>
<th>5 bytes</th>
<th>4 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

010011010100010101010100101011110010001010100101010101010101010...
Variable length codings

We do not know \textit{a priori} where the boundaries are.
Prefix codes

From the bits, we can deduce when to stop.
Prefix codes: phone numbers

Example

117
0446321111
0016507232300

Internally

888
21111
00446341111
00016507232300
Prefix codes: phone numbers

Example
117
0446321111
0016507232300

Internally
888
21111
00446341111
00016507232300

00165072323001170446321111
Prefix codes: phone numbers

Example

117
0446321111
0016507232300

Internally

888
21111
00446341111
00016507232300

00165072323001170446321111
### Prefix codes: UTF-8

<table>
<thead>
<tr>
<th>Character</th>
<th>Codepoint</th>
<th>Codepoint in binary</th>
<th>UTF-8 (variable length)</th>
</tr>
</thead>
</table>
## Prefix codes: UTF-8

<table>
<thead>
<tr>
<th>Character</th>
<th>Codepoint</th>
<th>Codepoint in binary</th>
<th>UTF-8 (variable length)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>U+0050</td>
<td>1010 000</td>
<td></td>
</tr>
</tbody>
</table>
Prefix codes: UTF-8

<table>
<thead>
<tr>
<th>Character</th>
<th>Codepoint</th>
<th>Codepoint in binary</th>
<th>UTF-8 (variable length)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>U+0050</td>
<td>1010 000</td>
<td>Less than 7 bits</td>
</tr>
</tbody>
</table>
## Prefix codes: UTF-8

<table>
<thead>
<tr>
<th>Character</th>
<th>Codepoint</th>
<th>Codepoint in binary</th>
<th>UTF-8 (variable length)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>U+0050</td>
<td>1010 000</td>
<td>01010000</td>
</tr>
</tbody>
</table>

Less than 7 bits
Prefix codes: UTF-8

<table>
<thead>
<tr>
<th>Character</th>
<th>Codepoint</th>
<th>Codepoint in binary</th>
<th>UTF-8 (variable length)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>U+0050</td>
<td>1010 000</td>
<td>01010000</td>
</tr>
<tr>
<td>π</td>
<td>U+03C0</td>
<td>11 1100 0000</td>
<td></td>
</tr>
</tbody>
</table>
# Prefix codes: UTF-8

<table>
<thead>
<tr>
<th>Character</th>
<th>Codepoint</th>
<th>Codepoint in binary</th>
<th>UTF-8 (variable length)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>U+0050</td>
<td>1010 000</td>
<td>01010000</td>
</tr>
<tr>
<td>π</td>
<td>U+03C0</td>
<td>11 1100 0000</td>
<td></td>
</tr>
</tbody>
</table>

- **P** is a character with codepoint `U+0050` in UTF-8. Its codepoint in binary is `1010 000`, which is less than 7 bits.
- **π** is a character with codepoint `U+03C0` in UTF-8. Its codepoint in binary is `11 1100 0000`, which is less than 11 bits.
## Prefix codes: UTF-8

<table>
<thead>
<tr>
<th>Character</th>
<th>Codepoint</th>
<th>Codepoint in binary</th>
<th>Less than 7 bits</th>
<th>Codepoint in binary</th>
<th>Less than 11 bits</th>
<th>UTF-8 (variable length)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>U+0050</td>
<td>1010 000</td>
<td></td>
<td>01010000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>π</td>
<td>U+03C0</td>
<td>111100 0000</td>
<td></td>
<td>11001111 10000000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Prefix codes: UTF-8

<table>
<thead>
<tr>
<th>Character</th>
<th>Codepoint</th>
<th>Codepoint in binary</th>
<th>UTF-8 (variable length)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>U+0050</td>
<td>1010 000</td>
<td>01010000</td>
</tr>
<tr>
<td>π</td>
<td>U+03C0</td>
<td>11 1100 0000</td>
<td>11001111 10000000</td>
</tr>
<tr>
<td>€</td>
<td>U+20AC</td>
<td>10 0000 1010 1100</td>
<td></td>
</tr>
</tbody>
</table>
## Prefix codes: UTF-8

<table>
<thead>
<tr>
<th>Character</th>
<th>Codepoint</th>
<th>Codepoint in binary</th>
<th>UTF-8 (variable length)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>U+0050</td>
<td>1010 000</td>
<td>01010000</td>
</tr>
<tr>
<td>π</td>
<td>U+03C0</td>
<td>11 1100 0000</td>
<td>11001111 10000000</td>
</tr>
<tr>
<td>€</td>
<td>U+20AC</td>
<td>10 0000 1010 1100</td>
<td></td>
</tr>
</tbody>
</table>
## Prefix codes: UTF-8

<table>
<thead>
<tr>
<th>Character</th>
<th>Codepoint</th>
<th>Codepoint in binary</th>
<th>UTF-8 (variable length)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>U+0050</td>
<td>1010 000</td>
<td>01010000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Less than 7 bits</td>
<td></td>
</tr>
<tr>
<td>π</td>
<td>U+03C0</td>
<td>11 1100 0000</td>
<td>11001111 10000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Less than 11 bits</td>
<td></td>
</tr>
<tr>
<td>€</td>
<td>U+20AC</td>
<td>10 0000 1010 1100</td>
<td>11100010 10000010 10101100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Less than 16 bits</td>
<td></td>
</tr>
</tbody>
</table>
Variable byte encoding (here with 8 bit packets)

000000000
Variable byte encoding (here with 8 bit packets)

000000000

continuation bit  encoding on n-1 bits (here 7)
Case 1: less than 7 bits required

4 (100)
Case 1: less than 7 bits required

4 (0000100)
Case 1: less than 7 bits required

4 (0000100)

0000100
Case 1: less than 7 bits required

4 \text{(0000100)}

10000100

1 = ends here
Case 2: Between 8 and 14 bits required

270 (1 0000 1110)
Case 2: Between 8 and 14 bits required

270 \((0000010 00011110)\)
Case 2: Between 8 and 14 bits required

270 (00000010 00011110)
Case 2: Between 8 and 14 bits required

270 \((00000010 \ 00011110)\)
And so on and so forth...

0000010 0001110 0101101 0011010 1110101
And so on and so forth...

0000010 0001110 0101101 0011010 1110101

0000010 0001110 0101101 0011010 1110101
And so on and so forth...

0000010 0001110 0101101 0011010 1110101

0 = doesn't end here  
1 = ends here
### Variable byte encoding: example with 4 bit packets

<table>
<thead>
<tr>
<th>decimal</th>
<th>binary</th>
<th>variable byte encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0000</td>
</tr>
</tbody>
</table>
Variable byte encoding: example with 4 bit packets

<table>
<thead>
<tr>
<th>decimal</th>
<th>binary</th>
<th>variable byte encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1001</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>1010</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>1011</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>1100</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>1101</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>1110</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>1111</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Variable byte encoding: example with 4 bit packets

<table>
<thead>
<tr>
<th>decimal</th>
<th>binary</th>
<th>variable byte encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>000</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>001</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
<td>010</td>
</tr>
<tr>
<td>3</td>
<td>011</td>
<td>011</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>111</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>0001 1000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>0001 1001</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>0001 1010</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>0001 1011</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>0001 1100</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>0001 1101</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>0001 1110</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>0001 1111</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Variable byte encoding: example with 4 bit packets

<table>
<thead>
<tr>
<th>decimal</th>
<th>binary</th>
<th>variable byte encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1001</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>1010</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>1011</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>1100</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>1101</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>1110</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>1111</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>0001 1000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>0001 1001</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>0001 1010</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>0001 1011</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>0001 1100</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>0001 1101</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>0001 1110</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>0001 1111</td>
</tr>
<tr>
<td>16</td>
<td>10000</td>
<td>0010 1000</td>
</tr>
<tr>
<td>17</td>
<td>10001</td>
<td>0010 1001</td>
</tr>
<tr>
<td>18</td>
<td>10010</td>
<td>0010 1010</td>
</tr>
<tr>
<td>19</td>
<td>10011</td>
<td>0010 1011</td>
</tr>
<tr>
<td>20</td>
<td>10100</td>
<td>0010 1100</td>
</tr>
<tr>
<td>21</td>
<td>10101</td>
<td>0010 1101</td>
</tr>
<tr>
<td>22</td>
<td>10110</td>
<td>0010 1110</td>
</tr>
<tr>
<td>23</td>
<td>10111</td>
<td>0010 1111</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Variable byte encoding: example with 4 bit packets

<table>
<thead>
<tr>
<th>decimal</th>
<th>binary</th>
<th>variable byte encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1001</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>1010</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>1011</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>1100</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>1101</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>1110</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>1111</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>0001 1000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>0001 1001</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>0001 1010</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>0001 1011</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>0001 1100</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>0001 1101</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>0001 1110</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>0001 1111</td>
</tr>
<tr>
<td>16</td>
<td>10000</td>
<td>0010 1000</td>
</tr>
<tr>
<td>17</td>
<td>10001</td>
<td>0010 1001</td>
</tr>
<tr>
<td>18</td>
<td>10010</td>
<td>0010 1010</td>
</tr>
<tr>
<td>19</td>
<td>10011</td>
<td>0010 1011</td>
</tr>
<tr>
<td>20</td>
<td>10100</td>
<td>0010 1100</td>
</tr>
<tr>
<td>21</td>
<td>10101</td>
<td>0010 1101</td>
</tr>
<tr>
<td>22</td>
<td>10110</td>
<td>0010 1110</td>
</tr>
<tr>
<td>23</td>
<td>10111</td>
<td>0010 1111</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Variable byte encoding: example with 4 bit packets

<table>
<thead>
<tr>
<th>decimal</th>
<th>binary</th>
<th>variable byte encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>0010</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>0011</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>0110</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>0111</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>1001</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>1010</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>1011</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>1100</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>1101</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>1110</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>1111</td>
</tr>
<tr>
<td>16</td>
<td>10000</td>
<td>0010 1000</td>
</tr>
<tr>
<td>17</td>
<td>10001</td>
<td>0010 1001</td>
</tr>
<tr>
<td>18</td>
<td>10010</td>
<td>0010 1010</td>
</tr>
<tr>
<td>19</td>
<td>10011</td>
<td>0010 1011</td>
</tr>
<tr>
<td>20</td>
<td>10100</td>
<td>0010 1100</td>
</tr>
<tr>
<td>21</td>
<td>10101</td>
<td>0010 1101</td>
</tr>
<tr>
<td>22</td>
<td>10110</td>
<td>0010 1110</td>
</tr>
<tr>
<td>23</td>
<td>10111</td>
<td>0010 1111</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>64</td>
<td>1000000</td>
<td>0001 0000 1000</td>
</tr>
</tbody>
</table>
Variable byte encoding: example with 4 bit packets

<table>
<thead>
<tr>
<th>decimal</th>
<th>binary</th>
<th>variable byte encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1001</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>1010</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>1011</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>1100</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>1101</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>1110</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>1111</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>0001 1000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>0001 1001</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>0001 1010</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>0001 1011</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>0001 1100</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>0001 1101</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>0001 1110</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>0001 1111</td>
</tr>
<tr>
<td>16</td>
<td>10000</td>
<td>0010 1000</td>
</tr>
<tr>
<td>17</td>
<td>10001</td>
<td>0010 1001</td>
</tr>
<tr>
<td>18</td>
<td>10010</td>
<td>0010 1010</td>
</tr>
<tr>
<td>19</td>
<td>10011</td>
<td>0010 1011</td>
</tr>
<tr>
<td>20</td>
<td>10100</td>
<td>0010 1100</td>
</tr>
<tr>
<td>21</td>
<td>10101</td>
<td>0010 1101</td>
</tr>
<tr>
<td>22</td>
<td>10110</td>
<td>0010 1110</td>
</tr>
<tr>
<td>23</td>
<td>10111</td>
<td>0010 1111</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>64</td>
<td>1000000</td>
<td>0001 0000 1000</td>
</tr>
</tbody>
</table>

50% less space
Variable byte encoding is a parameterized encoding

<table>
<thead>
<tr>
<th>n</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>xx</td>
</tr>
<tr>
<td>4</td>
<td>xxxx</td>
</tr>
<tr>
<td>8</td>
<td>xxxxxxxx</td>
</tr>
<tr>
<td>16</td>
<td>xxxxxxxxxxxxxx</td>
</tr>
</tbody>
</table>
Example (here, 8 bits)

100101011101111100101000011100000110100100101100100101000101010111
Example (here, 8 bits)

1001010111011111110010000011100000110100100101100100101000010101011
Example (here, 8 bits)
Example (here, 8 bits)
Example (here, 8 bits)
Example (here, 8 bits)

10010101 | 11011111 | 11001000 | 00111000 | 00110100 | 10010110 | 01001010 | 00101000 | 10101011

101011 1011111 1001000 11100001101000010110

21 95 72 924,182
No free lunch
Compromise for variable byte encoding

Big packets

1111010101010100101

Little compression

Little overhead
Compromise for variable byte encoding

Big packets
1111010101010100101
Little compression
Little overhead

Small packets
1111101101101100001
Much compression
Lot of bits to manipulate
Can we compress even more?
Can we compress even more?

bitwise?
Gamma encoding
Peter Elias

1923 - 2001
Unary code
Unary code

12 ones
Unary code

12

1111111111110

and a zero to mark the stop
# First integers in unary code

<table>
<thead>
<tr>
<th>integer</th>
<th>length (unary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>110</td>
</tr>
<tr>
<td>3</td>
<td>1110</td>
</tr>
<tr>
<td>4</td>
<td>11110</td>
</tr>
<tr>
<td>5</td>
<td>111110</td>
</tr>
<tr>
<td>6</td>
<td>1111110</td>
</tr>
<tr>
<td>7</td>
<td>11111110</td>
</tr>
<tr>
<td>8</td>
<td>111111110</td>
</tr>
<tr>
<td>9</td>
<td>1111111110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Example (here, 8 bits)

11111111011111011111111111111011101111100111111111111011111111101111110
Example (here, 8 bits)

111111101111101111111111110111011111001111111111110111111110111110
Example (here, 8 bits)
Example (here, 8 bits)
Gamma encoding: example
Gamma encoding: example

10011
Gamma encoding: example

19

binary

10011
Gamma encoding: example

19

binary

10011

0011
Gamma encoding: example

19

1

0011

Length in unary

11110

0011
Gamma encoding: example

19

binary

10011

Length in unary

11110

111100011
Gamma encoding on the first integers

decimal
0
1
2
3
Gamma encoding on the first integers

<table>
<thead>
<tr>
<th>decimal</th>
<th>binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
</tr>
</tbody>
</table>
Gamma encoding on the first integers

<table>
<thead>
<tr>
<th>decimal</th>
<th>binary</th>
<th>binary without leading 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>1</td>
</tr>
</tbody>
</table>
Gamma encoding on the first integers

<table>
<thead>
<tr>
<th>decimal</th>
<th>binary</th>
<th>binary without leading 1</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Gamma encoding on the first integers

<table>
<thead>
<tr>
<th>decimal</th>
<th>binary</th>
<th>binary without leading 1</th>
<th>length</th>
<th>length (unary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>
## Gamma encoding on the first integers

<table>
<thead>
<tr>
<th>decimal</th>
<th>binary</th>
<th>binary without leading 1</th>
<th>length</th>
<th>length (unary)</th>
<th>gamma code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>101</td>
</tr>
</tbody>
</table>
Gamma encoding on the first integers

<table>
<thead>
<tr>
<th>decimal</th>
<th>binary</th>
<th>binary without leading 1</th>
<th>length</th>
<th>length (unary)</th>
<th>gamma code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>101</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>00</td>
<td>2</td>
<td>110</td>
<td>11000</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>01</td>
<td>2</td>
<td>110</td>
<td>11001</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>10</td>
<td>2</td>
<td>110</td>
<td>11010</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>11</td>
<td>2</td>
<td>110</td>
<td>11011</td>
</tr>
</tbody>
</table>
## Gamma encoding on the first integers

<table>
<thead>
<tr>
<th>decimal</th>
<th>binary</th>
<th>binary without leading 1</th>
<th>length</th>
<th>length (unary)</th>
<th>gamma code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>N/A</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>101</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>00</td>
<td>2</td>
<td>110</td>
<td>11000</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>01</td>
<td>2</td>
<td>110</td>
<td>11001</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>10</td>
<td>2</td>
<td>110</td>
<td>11010</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>11</td>
<td>2</td>
<td>110</td>
<td>11011</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>000</td>
<td>3</td>
<td>1110</td>
<td>1110000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>001</td>
<td>3</td>
<td>1110</td>
<td>1110001</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>010</td>
<td>3</td>
<td>1110</td>
<td>1110010</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>011</td>
<td>3</td>
<td>1110</td>
<td>1110011</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>100</td>
<td>3</td>
<td>1110</td>
<td>111100</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>101</td>
<td>3</td>
<td>1110</td>
<td>111101</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>110</td>
<td>3</td>
<td>1110</td>
<td>1111010</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>111</td>
<td>3</td>
<td>1110</td>
<td>1111011</td>
</tr>
</tbody>
</table>
### Gamma encoding on the first integers

<table>
<thead>
<tr>
<th>decimal</th>
<th>binary</th>
<th>binary without leading 1</th>
<th>length</th>
<th>length (unary)</th>
<th>gamma code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>101</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>00</td>
<td>2</td>
<td>110</td>
<td>11000</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>01</td>
<td>2</td>
<td>110</td>
<td>11001</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>10</td>
<td>2</td>
<td>110</td>
<td>11010</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>11</td>
<td>2</td>
<td>110</td>
<td>11011</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>000</td>
<td>3</td>
<td>1110</td>
<td>1110000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>001</td>
<td>3</td>
<td>1110</td>
<td>1110001</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>010</td>
<td>3</td>
<td>1110</td>
<td>1110010</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>011</td>
<td>3</td>
<td>1110</td>
<td>1110011</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>100</td>
<td>3</td>
<td>1110</td>
<td>1110100</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>101</td>
<td>3</td>
<td>1110</td>
<td>1110101</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>110</td>
<td>3</td>
<td>1110</td>
<td>1110110</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>111</td>
<td>3</td>
<td>1110</td>
<td>1110111</td>
</tr>
<tr>
<td>16</td>
<td>10000</td>
<td>0000</td>
<td>4</td>
<td>11110</td>
<td>111100000</td>
</tr>
<tr>
<td>17</td>
<td>10001</td>
<td>0001</td>
<td>4</td>
<td>11110</td>
<td>111100001</td>
</tr>
<tr>
<td>18</td>
<td>10010</td>
<td>0010</td>
<td>4</td>
<td>11110</td>
<td>111100010</td>
</tr>
<tr>
<td>19</td>
<td>10011</td>
<td>0011</td>
<td>4</td>
<td>11110</td>
<td>111100011</td>
</tr>
<tr>
<td>20</td>
<td>10100</td>
<td>0100</td>
<td>4</td>
<td>11110</td>
<td>111100100</td>
</tr>
<tr>
<td>21</td>
<td>10101</td>
<td>0101</td>
<td>4</td>
<td>11110</td>
<td>111100101</td>
</tr>
<tr>
<td>22</td>
<td>10110</td>
<td>0110</td>
<td>4</td>
<td>11110</td>
<td>111100110</td>
</tr>
<tr>
<td>23</td>
<td>10111</td>
<td>0111</td>
<td>4</td>
<td>11110</td>
<td>111100111</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>64</td>
<td>1000000</td>
<td>000000</td>
<td>6</td>
<td>111110</td>
<td>11111000000</td>
</tr>
</tbody>
</table>

...
Gamma encoding properties

Variable length encoding
Gamma encoding properties

Variable length encoding

Prefix encoding
Gamma encoding properties

- Variable length encoding
- Prefix encoding
- Universal encoding
Shannon Entropy

\[ H(X) = \mathbb{E}[I(X)] \]
Shannon Entropy

\[ H(X) = \mathbb{E}[I(X)] \]

"Amount of information" = number of bits
Shannon Entropy

\[ H(X) = \mathbb{E}[I(X)] \]

"Amount of information" = number of bits
Shannon Entropy

\[ H(X) = \mathbb{E}[-\log_2(p_X(X))] \]

"Amount of information" = number of bits
Shannon Entropy

\[ H(X) = \mathbb{E}[I(X)] = - \sum_{x \in X(\Omega)} p_X(x) \log_2 p_X(x) \]
Shannon Entropy

\[ H(X) = \mathbb{E}[I(X)] = - \sum_{x \in X(\Omega)} p_X(x) \log_2 p_X(x) \]
Expected length of gamma encoding

$$\mathbb{E}[L_{\gamma}(X)] \leq 3H(X) = 3\mathbb{E}[I(X)]$$

one factor from optimal!
Expected length of gamma encoding

\[ \mathbb{E}[L_\gamma(X)] \leq 2H(X) + 1 = 2\mathbb{E}[I(X)] + 1 \]

one factor from optimal!
How much can we compress the inverted index?
Zipf's law

\[ \text{Frequency} = \frac{k}{\text{Rank}} \]
Zipf's law (renormalized)

Renormalized frequency $= \frac{c}{\text{Rank}}$
Zipf's law (renormalized)

Renormalized frequency $= \frac{c}{\text{Rank}}$

$$\sum_{i=1}^{i=M} \frac{c}{\text{Rank}} = 1$$
Zipf's law

\[
\text{Number of occurrences per document} = \frac{\text{Document length} \times c}{\text{Rank}}
\]
Zipf's law

Number of occurrences per document = \( \frac{Lc}{\text{Rank}} \)
Zipf's law

Number of postings = Number of documents × Number of occurrences per documents
Zipf's law

Number of postings = \( \frac{NLc}{Rank} \)
Zipf's law

Number of postings = $\frac{NLc}{Rank}$
Zipf's law

Number of postings = \( \frac{NLc}{Rank} \)
Zipf's law

Number of postings \(= \frac{NLc}{Rank}\)
Zipf's law

Number of postings = \( \frac{NLc}{Rank} \)
Zipf's law

\[
N \text{ postings}
\]

\[
\frac{N}{2} \text{ postings}
\]

Number of postings = \( \frac{NLc}{Rank} \)

Rank = Lc

Rank = 2Lc

Rank = 3Lc
Zipf's law

\[ N \text{ postings} \]
 Rank = \( Lc \)

\[ \frac{N}{2} \text{ postings} \]
 Rank = \( 2Lc \)

\[ \frac{N}{3} \text{ postings} \]
 Rank = \( 3Lc \)

Number of postings = \( \frac{NLc}{Rank} \)
Zipf's law

Approximations

- $N$ postings
  - Rank = $Lc$

- $N/2$ postings
  - Rank = $2Lc$

- $N/3$ postings
  - Rank = $3Lc$

Number of postings = $\frac{NLc}{Rank}$
Zipf's law
Zipf's law

\[ \text{gap} = j \]

N/j postings
Zipf's law

\[
\begin{align*}
N \text{ postings} & \quad \text{gap = 1} \\
\frac{N}{2} \text{ postings} & \quad \text{gap = 2} \\
\frac{N}{3} \text{ postings} & \quad \text{gap = 3}
\end{align*}
\]

\[
\text{Number of postings} = \text{Number of documents} \times \frac{L_c}{\text{Rank}}
\]
Zipf's law

\[
\text{Rank} = (j-1)Lc \\
\frac{N}{j} \text{ postings} \quad \text{gap} = j \\
\text{Rank} = jLc
\]
Zipf's law

\[
\frac{N}{j} \text{ postings} \\
\text{gap} = j \\
\Rightarrow \text{Rank} = (j-1)Lc \\
\Rightarrow \text{Rank} = jLc
\]

\[
\#\text{bits per term} \approx \frac{N}{j} \times (2 \log_2(j) + 1)
\]
Zipf's law

\[ \frac{N}{j} \text{ postings} \]

gap = j

\[ \text{Rank} = (j-1)Lc \]
\[ \text{Rank} = jLc \]

\[ \#\text{bits per term block} \approx \frac{NLc}{j} \times (2 \log_2(j) + 1) \]
Zipf's law

$$\# \text{bits} \approx \sum_{j=1}^{j = \frac{M}{Lc}} \frac{NLc}{j} \times (2 \log_2(j) + 1)$$
Zipf's law

\[ \#\text{bits} \approx \sum_{j=1}^{j=M/Lc} \frac{2NLc \log_2(j)}{j} \]
How did we do?

Collection: 960 MB
How did we do?

Uncompressed on 32 bits

Collection: 960 MB

400 MB
How did we do?

Uncompressed on 32 bits

Uncompressed on 20 bits

Collection: 960 MB
How did we do?

Uncompressed on 32 bits
400 MB

Uncompressed on 20 bits
250 MB

Variable byte encoding (gaps)
116 MB

Collection: 960 MB
How did we do?

Uncompressed on 32 bits: 400 MB

Uncompressed on 20 bits: 250 MB

Variable byte encoding (gaps): 116 MB

Elias $\gamma$ encoding (gaps): 101 MB

Collection: 960 MB
Credits

This week:

Chapter 5

Index compression