The Inverted Index

Definition: Inverted Index

Given a set of documents and terms that are contained in the documents, an inverted index stores the terms and associated with each term $t$

- the number of documents $f_t$ that contain $t$ and
- an ordered list $L(t)$ of documents containing $t$

1. The old night keeper keeps the keep in the town
2. In the big old house in the big 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 light
The Inverted Index

Definition: Inverted Index

Given a set of documents and terms that are contained in the documents, an inverted index stores the terms and associated with each term $t$:

- the number of documents $f_t$ that contain $t$ and
- an ordered list $L(t)$ of documents containing $t$

1. The old night keeper keeps the keep in the town
2. In the big old house in the big 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 light

<table>
<thead>
<tr>
<th>term</th>
<th>$f_t$</th>
<th>$L(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>1</td>
<td>[6]</td>
</tr>
<tr>
<td>big</td>
<td>2</td>
<td>[2,3]</td>
</tr>
<tr>
<td>dark</td>
<td>1</td>
<td>[6]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>had</td>
<td>1</td>
<td>[3]</td>
</tr>
<tr>
<td>house</td>
<td>2</td>
<td>[2,3]</td>
</tr>
<tr>
<td>in</td>
<td>5</td>
<td>[1,2,3,5,6]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
The Inverted Index

Definition: Inverted Index

Given a set of documents and terms that are contained in the documents, an inverted index stores the terms and associated with each term $t$

- the number of documents $f_t$ that contain $t$ and
- an ordered list $L(t)$ of documents containing $t$

1. The old night keeper keeps the keep in the town
2. In the big old house in the big 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 light

<table>
<thead>
<tr>
<th>term</th>
<th>$f_t$</th>
<th>$L(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>1</td>
<td>[6]</td>
</tr>
<tr>
<td>big</td>
<td>2</td>
<td>[2,3]</td>
</tr>
<tr>
<td>dark</td>
<td>1</td>
<td>[6]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>had</td>
<td>1</td>
<td>[3]</td>
</tr>
<tr>
<td>house</td>
<td>2</td>
<td>[2,3]</td>
</tr>
<tr>
<td>in</td>
<td>5</td>
<td>[1,2,3,5,6]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
The Inverted Index: Queries

### Conjunctive Queries
- Given two lists \( M \) and \( N \), return all documents contained in both lists: \( M \cap N \)

1. The old night keeper keeps the keep \( \text{in} \) the town
2. In the big old house \( \text{in} \) the big old gown
3. The house \( \text{in} \) the town had the big old keep
4. Where the old night keeper never did sleep
5. The night keeper keeps the keep \( \text{in} \) the night
6. And keeps \( \text{in} \) the dark and sleeps \( \text{in} \) the light
The Inverted Index: Queries

Conjunctive Queries

- Given two lists $M$ and $N$, return all documents contained in both lists: $M \cap N$

Disjunctive Queries

- Given two lists $M$ and $N$, return all documents contained in either list: $M \cup N$

<table>
<thead>
<tr>
<th>term $t$</th>
<th>$f_t$</th>
<th>$L(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>1</td>
<td>[6]</td>
</tr>
<tr>
<td>big</td>
<td>2</td>
<td>[2, 3]</td>
</tr>
<tr>
<td>dark</td>
<td>1</td>
<td>[6]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>had</td>
<td>1</td>
<td>[3]</td>
</tr>
<tr>
<td>house</td>
<td>2</td>
<td>[2, 3]</td>
</tr>
<tr>
<td>in</td>
<td>5</td>
<td>[1, 2, 3, 5, 6]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
The Inverted Index: Queries

Conjunctive Queries
- Given two lists $M$ and $N$, return all documents contained in both lists: $M \cap N$

Disjunctive Queries
- Given two lists $M$ and $N$, return all documents contained in either list: $M \cup N$

Phrase Queries
- Given two terms $t_1$ and $t_2$, return all documents containing $t_1 t_2$ among all previous discussed indices, where all previous discussed indices can do so

1. The old night keeper keeps the keep in the town
2. In the big old house in the big 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 light

<table>
<thead>
<tr>
<th>term</th>
<th>$t_i$</th>
<th>$L(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>1</td>
<td>[6]</td>
</tr>
<tr>
<td>big</td>
<td>2</td>
<td>[2, 3]</td>
</tr>
<tr>
<td>dark</td>
<td>1</td>
<td>[6]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>had</td>
<td>1</td>
<td>[3]</td>
</tr>
<tr>
<td>house</td>
<td>2</td>
<td>[2, 3]</td>
</tr>
<tr>
<td>in</td>
<td>5</td>
<td>[1, 2, 3, 5, 6]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Inverted Index: Representing the Terms (1/2)

- terms can be represented using tries
- in each leaf, store pointer to list for term

- simple representation
- easy to add and remove terms
Inverted Index: Representing the Words (2/2)

- use multiplicative hash function
- \( h(t[1] \ldots t[\ell]) = ((\sum_{i=1}^{\ell} a_i \cdot t[i]) \mod p) \mod m \)
- for prime \( p < m \) and
- fixed random integers \( a_i \in [1, p] \)

- good worst cast guarantee
- \( \text{Prob}[h(x) = h(y)] = O(1/m) \) for \( x \neq y \)
Inverted Index: Document Lists

- document ids are sorted
- if ids are in \([1, U]\), storing them requires \(\lceil \lg U \rceil\) bits per id

Now

- different ideas on how to better store ids
- not all ideas work with all algorithms
- different space usage and complexity

Binary Codes

- an integer \(x\) can be represented as binary \((x)_2\)
- for fast access, all binary representations must have the same width
Difference Encoding

- given a document list \( N = [d_1, \ldots, d_{|N|}] \)
- the document ids are sorted: \( d_1 < \cdots < d_{|N|} \)
- store first id
- represent other ids by difference: \( \delta_i = d_i - d_{i-1} \)

**Definition: \( \Delta \)-Encoding**

A \( \Delta \)-encoded document list \( N = [d_1, \ldots, d_{|N|}] \) is
\( N = [d_1, d_2 - d_1, \ldots, d_{|N|} - d_{|N-1|}] \)
Difference Encoding

- given a document list \( N = [d_1, \ldots, d_{|N|}] \)
- the document ids are sorted: \( d_1 < \cdots < d_{|N|} \)
- store first id
- represent other ids by difference: \( \delta_i = d_i - d_{i-1} \)

Definition: \( \Delta \)-Encoding

A \( \Delta \)-encoded document list \( N = [d_1, \ldots, d_{|N|}] \) is \( N = [d_1, d_2 - d_1, \ldots, d_{|N|} - d_{|N|-1}] \)

- Just ids:
  - \( N = [4, 11, 12, 30, 42, 54] \)
  - \( \Delta \)-encoded
    - \( N = [4, 7, 1, 18, 12, 12] \)
Difference Encoding

- given a document list \( N = [d_1, \ldots, d_{|N|}] \)
- the document ids are sorted: \( d_1 < \cdots < d_{|N|} \)
- store first id
- represent other ids by difference: \( \delta_i = d_i - d_{i-1} \)

**Definition: \( \Delta \)-Encoding**

A \( \Delta \)-encoded document list \( N = [d_1, \ldots, d_{|N|}] \) is \( N = [d_1, d_2 - d_1, \ldots, d_{|N|} - d_{|N-1|}] \)

- can this be compressed further?
- accessing id requires scanning

Just ids:
- \( N = [4, 11, 12, 30, 42, 54] \)

\( \Delta \)-encoded
- \( N = [4, 7, 1, 18, 12, 12] \)
Definition: Unary Codes

Given an integer \( x > 0 \), its unary code \((x)_1\) is \(1^{x-1}0\)

- \(|(x)_1| = x\) bits
- encoded integers can be accessed using rank and select queries
- if 0 has to be encoded, all codes require an additional bit
Definition: Unary Codes

Given an integer \( x > 0 \), its unary code \((x)_1\) is \(1^{x-1}0\)

- \(|(x)_1| = x \text{ bits}\)
- encoded integers can be accessed using rank and select queries
- if 0 has to be encoded, all codes require an additional bit

Just ids:
- \( N = [4, 11, 12, 30, 42, 54] \)

\(\Delta\)-encoded
- \( N = [4, 7, 1, 18, 12, 12] \)

Unary Codes:
- \( N = [1110111111001^{17}01^{11}011111111110] \)
Ternary Encoding

Definition: Ternary Codes

Given an integer $x > 0$, represent it in ternary using
- 00 to represent 0
- 01 to represent 1
- 10 to represent 2

and append 11 to each code to obtain its ternary code $(x)_3$

$$|(x)_3| = 2\lfloor \log_3(x - 1) \rfloor + 2$$
Ternary Encoding

Definition: Ternary Codes

Given an integer $x > 0$, represent it in ternary using
- 00 to represent 0
- 01 to represent 1
- 10 to represent 2

and append 11 to each code to obtain its ternary code $(x)_3$

$|(x)_3| = 2\lfloor \log_3(x-1) \rfloor + 2$

Just ids:
- $N = [4, 11, 12, 30, 42, 54]$  
  $\Delta$-encoded
- $N = [4, 7, 1, 18, 12, 12]$

Unary Codes:
- $N = [111011111001170111011111111110]$  

Ternary Codes:
- $N =$  
  $[100011100111011011011011100101101011]$
Lemma: Zeckendorf’s Thorem

Let $f_i$ be the $i$-th Fibonacci number, then each integer $x > 0$ can be represented as

$$n = \sum_{i=2}^{k} c_i f_i$$

with $c_i \in \{0, 1\}$ and $c_i + c_{i+1} < 2$

Definition: Fibonacci Code

Given an integer $x > 0$ use the sequence of $c_i$’s followed by a 1 as its Fibonacci code $(x)_\phi$.
Fibonacci Encoding

**Lemma: Zeckendorf’s Theorem**

Let $f_i$ be the $i$-th Fibonacci number, then each integer $x > 0$ can be represented as

$$n = \sum_{i=2}^{k} c_i f_i$$

with $c_i \in \{0, 1\}$ and $c_i + c_{i+1} < 2$

**Definition: Fibonacci Code**

Given an integer $x > 0$ use the sequence of $c_i$’s followed by a 1 as its Fibonacci code $(x)_\phi$

- 11 does not occur in any sequence
- To compute find largest Fibonacci number $f_i < x$ and repeat process for $x - f_i$
- Fibonacci codes are smaller than ternary codes for smaller integers
**Lemma: Zeckendorf’s Theorem**

Let $f_i$ be the $i$-th Fibonacci number, then each integer $x > 0$ can be represented as

$$n = \sum_{i=2}^{k} c_i f_i$$

with $c_i \in \{0, 1\}$ and $c_i + c_{i+1} < 2$

**Definition: Fibonacci Code**

Given an integer $x > 0$ use the sequence of $c_i$’s followed by a 1 as its Fibonacci code $(x)_\phi$

- 11 does not occur in any sequence
- to compute find largest Fibonacci number $f_i < x$ and repeat process for $x - f_i$
- Fibonacci codes are smaller than ternary codes for smaller integers

- $f_2 = 1$, $f_3 = 2$, $f_4 = 3$, $f_5 = 5$, $f_6 = 8$, $f_7 = 13$
- 4: $f_2 + f_4 = 1011$
- 7: $f_3 + f_5 = 01011$
- 1: $f_2 = 11$
- 18: $f_5 + f_7 = 0001011$
- 12: $f_2 + f_4 + f_6 = 101011$
**Elias-γ-Encoding [Eli75]**

**Definition: Elias-γ-Code**

Given an integer $x > 0$, its *Elias-gamma-code* $(x)_\gamma$ is

$$(x)_\gamma = 0 \lfloor \lg x \rfloor (x)_2$$

- $|(x)_\gamma| = 2 \lfloor \lg x \rfloor + 1$ bit
- first part gives length of binary representation
- first bit of $(x)_2$ is one bit
**Definition: Elias-γ-Code**

Given an integer $x > 0$, its Elias-\textit{gamma}-code $(x)_\gamma$ is

$$(x)_\gamma = 0^{\lceil \log x \rceil} (x)_2$$

- $| (x)_\gamma | = 2 \lceil \log x \rceil + 1 \text{ bit}$
- first part gives length of binary representation
- first bit of $(x)_2$ is one bit
**Elias-δ-Encoding [Eli75]**

**Definition: Elias-δ-Code**

Given an integer \( x > 0 \), its Elias-δ-code \((x)_\delta\) is

\[
(x)_\delta = (\lfloor \lg x \rfloor + 1)^\gamma(x)_2[2..|(x)_2|]
\]

- encode length of binary representation using Elias-γ code
- first bit of binary representation not required anymore
- \( |(x)_\delta| = 2[\lg(\lfloor \lg x \rfloor + 1)] + 1 + \lfloor \lg x \rfloor \) bits
Elias-$\delta$-Encoding [Eli75]

**Definition: Elias-$\delta$-Code**

Given an integer $x > 0$, its Elias-$\delta$-code $(x)_\delta$ is

$$(x)_\delta = (\lfloor \lg x \rfloor + 1)_\gamma (x)_2 [2..|(x)_2|]$$

- encode length of binary representation using Elias-$\gamma$ code
- first bit of binary representation not required anymore
- $|(x)_\delta| = 2\lfloor \lg(\lfloor \lg x \rfloor + 1) \rfloor + 1 + \lfloor \lg x \rfloor$ bits

<table>
<thead>
<tr>
<th>Elias-$\gamma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4: 00100</td>
</tr>
<tr>
<td>7: 00111</td>
</tr>
<tr>
<td>1: 1</td>
</tr>
<tr>
<td>18: 000010010</td>
</tr>
<tr>
<td>12: 0001000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Elias-$\delta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4: 01100</td>
</tr>
<tr>
<td>7: 01111</td>
</tr>
<tr>
<td>1: 1</td>
</tr>
<tr>
<td>18: 001010010</td>
</tr>
<tr>
<td>12: 00100100</td>
</tr>
</tbody>
</table>
Definition: Golomb Code

Given an integer $x > 0$ and a constant $b > 0$, the Golomb code consists of

- $q = \lfloor \frac{x}{b} \rfloor$
- $r = x - qb = x \% b$
- $c = \lceil \lg b \rceil$

with

$$(x)_{\text{Gol}(b)} = (q)_1(r)_2$$

where $(r)_2$ depends on its size

- $r < 2^\lceil \lg b \rceil - 1$: $r$ requires $\lceil \lg b \rceil$ bits and starts with a 0
- $r \geq 2^\lceil \lg b \rceil - 1$: $r$ requires $\lceil \lg b \rceil$ bits and starts with a 1 and it encodes $r - 2^\lceil \lg b \rceil - 1$
Definition: Golomb Code

Given an integer $x > 0$ and a constant $b > 0$, the Golomb code consists of

- $q = \left\lfloor \frac{x}{b} \right\rfloor$
- $r = x - qb = x \% b$
- $c = \lceil \lg b \rceil$

with

$$(x)_{\text{Gol}(b)} = (q)_1 (r)_2$$

where $(r)_2$ depends on its size

- $r < 2^{\lceil \lg b \rceil - 1}$: $r$ requires $\lceil \lg b \rceil$ bits and starts with a 0
- $r \geq 2^{\lceil \lg b \rceil - 1}$: $r$ requires $\lceil \lg b \rceil$ bits and starts with a 1 and it encodes $r - 2^{\lceil \lg b \rceil - 1}$

$b$ has to be fixed for all codes

still variable length
Golomb Encoding [Gol66]

Definition: Golomb Code

Given an integer $x > 0$ and a constant $b > 0$, the Golomb code consists of

- $q = \lfloor \frac{x}{b} \rfloor$
- $r = x - qb = x \% b$
- $c = \lceil \log_2 b \rceil$

with

$$ (x)_{\text{Gol}(b)} = (q)_1(r)_2 $$

where $(r)_2$ depends on its size

- $r < 2^{\lceil \log_2 b \rceil - 1}$: $r$ requires $\lceil \log_2 b \rceil$ bits and starts with a 0
- $r \geq 2^{\lceil \log_2 b \rceil - 1}$: $r$ requires $\lceil \log_2 b \rceil$ bits and starts with a 1 and it encodes $r - 2^{\lceil \log_2 b \rceil - 1}$

- $b$ has to be fixed for all codes
- still variable length

- for $b = 5$, there are 4 remainders: $00, 01, 100, 101$, and $110$
- $2^{\lceil \log_5 5 \rceil - 1} = 2$
- $0, 1 < 2$: $00$ and $01$ require 2 bits
- $2, 3, 4 \geq 2$: require 3 bits and encode $0, 1, 2$
  starting with 1
Comparison of Codes
Back to Queries: Conjunctive Queries

Task

- given terms $t_1, \ldots, t_k$
- intersect $L(t_1) \cap L(t_2) \cap \cdots \cap L(t_k)$

- pairwise intersection usually works best
- intersection of two lists is of interest
- start with two shortest and continue like that
Back to Queries: Conjunctive Queries

**Task**
- given terms $t_1, \ldots, t_k$
- intersect $L(t_1) \cap L(t_2) \cap \cdots \cap L(t_k)$
- pairwise intersection usually works best
- intersection of two lists is of interest
- start with two shortest and continue like that

**Setting**
- two lists $M$ and $N$ with
- $|M| = m$ and $|N| = n$ and
- $m \leq n$
- different algorithms to intersect lists
- assuming lists are $\Delta$ encoded
**Naive Scanning**

---

**Zipper**

- scan both lists as in binary merging

---

Lemma: Running Time Zipper

Intersecting two sorted lists of sizes $m$ and $n$ using zipper requires $O(m + n)$ time.

Proof (Sketch)

compare entries until one list is empty

if $\max\{id : id \in \mathbb{N}\} < \text{some element in } M$, then all elements in $N$ are compared resulting in $O(n + m)$ time.

works well with $\Delta$-encoding in real implementations zipping is good until $n > 2m$ [BS05]

example on the board / chalkboard-teacher
Naive Scanning

**Zipper**
- scan both lists as in binary merging

**Lemma: Running Time Zipper**
Intersecting two sorted lists of sizes $m$ and $n$ using zipper requires $O(m + n)$ time.

Naive Scanning

**Zipper**
- scan both lists as in binary merging

**Lemma: Running Time Zipper**
Intersecting two sorted lists of sizes $m$ and $n$ using zipper requires $O(m + n)$ time.

**Proof (Sketch)**
- compare entries until one list is empty
- if $\max\{id : id \in N\} < \text{some element in } M$, then all elements in $N$ are compared
- resulting in $O(n + m)$ time
**Naive Scanning**

**Zipper**
- scan both lists as in binary merging

**Lemma: Running Time Zipper**
Intersecting two sorted lists of sizes $m$ and $n$ using zipper requires $O(m + n)$ time.

**Proof (Sketch)**
- compare entries until one list is empty
- if $\max\{id: id \in N\} < \text{some element in } M$, then all elements in $N$ are compared
- resulting in $O(n + m)$ time

- works well with $\Delta$-encoding
- in real implementations zipping is good until $n > 20m$ [BS05]
**Naive Scanning**

**Zipper**
- scan both lists as in binary merging

**Lemma: Running Time Zipper**
Intersecting two sorted lists of sizes $m$ and $n$ using zipper requires $O(m + n)$ time.

**Proof (Sketch)**
- compare entries until one list is empty
- if $\max\{id : id \in N\} < \text{some element in } M$, then all elements in $N$ are compared
- resulting in $O(n + m)$ time

- works well with $\Delta$-encoding
- in real implementations zipping is good until $n > 20m$ [BS05]

- example on the board
Simple Binary Search

- search each document in $M$ in $N$ using binary search
Simple Binary Search

- search each document in $M$ in $N$ using binary search

Lemma: Running Time Simple Binary Search

Intersecting two sorted lists of sizes $m$ and $n$ using a simple binary search requires $O(m \lg n)$ time.
Simple Binary Search
- search each document in $M$ in $N$ using binary search

Lemma: Running Time Simple Binary Search
Intersecting two sorted lists of sizes $m$ and $n$ using a simple binary search requires $O(m \lg n)$ time.

Proof (Sketch)
- binary search on $N$ because $n \geq m$
- for each id in $N$ binary search in $O(\lg n)$ time
- resulting in $O(m \lg n)$ total time
Simple Binary Search
- search each document in $M$ in $N$ using binary search

Lemma: Running Time Simple Binary Search
Intersecting two sorted lists of sizes $m$ and $n$ using a simple binary search requires $O(m \lg n)$ time.

Proof (Sketch)
- binary search on $N$ because $n \geq m$
- for each id in $N$ binary search in $O(\lg n)$ time
- resulting in $O(m \lg n)$ total time
Binary Search (1/2)

Simple Binary Search
- search each document in $M$ in $N$ using binary search

Lemma: Running Time Simple Binary Search
Intersecting two sorted lists of sizes $m$ and $n$ using a simple binary search requires $O(m \lg n)$ time.

Proof (Sketch)
- binary search on $N$ because $n \geq m$
- for each id in $N$ binary search in $O(\lg n)$ time
- resulting in $O(m \lg n)$ total time
Double Binary Search

- let $p_m = \lfloor m/2 \rfloor$
- search for $M[p_m]$ in $N$ using binary search
- let result be position $p_n$
- if $M[p_m] = N[p_n]$ add $M[p_m]$ to result
- continue recursively by intersecting
  - $M[1, p_m] \cap N[1, p_n]$ and
  - $M[1 + p_m, |M|] \cap N[1 + p_n, |N|]$
### Binary Search (2/2)

**Double Binary Search**
- Let $p_m = \lfloor \frac{m}{2} \rfloor$
- Search for $M[p_m]$ in $N$ using binary search
- Let result be position $p_n$
- If $M[p_m] = N[p_n]$ add $M[p_m]$ to result
- Continue recursively by intersecting
  - $M[1, p_m] \cap N[1, p_n]$ and
  - $M[1 + p_m, |M|] \cap N[1 + p_n, |N|]$

**Lemma: Running Time Double Binary Search**
Intersecting two sorted lists of sizes $m$ and $n$ using a double binary search requires $O(m \lg \frac{n}{m})$ time.
Double Binary Search

- let $p_m = \lfloor \frac{m}{2} \rfloor$
- search for $M[p_m]$ in $N$ using binary search
- let result be position $p_n$
- if $M[p_m] = N[p_n]$ add $M[p_m]$ to result
- continue recursively by intersecting
  - $M[1, p_m] \cap N[1, p_n]$ and
  - $M[1 + p_m, |M|] \cap N[1 + p_n, |N|]$

Lemma: Running Time Double Binary Search

Intersecting two sorted lists of sizes $m$ and $n$ using a double binary search requires $O(m \lg \frac{n}{m})$ time.

Proof (Sketch)

- look at running time of binary search at each recursion depth
  - depth 0: $lgn$
  - depth 1: $2 \lg \frac{n}{2}$
  - depth 2: $4 \lg \frac{n}{4}$
  - depth $m$: $m \lg \frac{n}{m}$

Since depth of recursion is at most $m$, this results in

$$\sum_{i=0}^{\lg m} \frac{m}{2^i} (\lg \frac{n}{m} + i) = m (\lg \frac{n}{m} \sum_{i=0}^{\lg m} \frac{1}{2^i} + \sum_{i=0}^{\lg m} \frac{1}{2^i})$$

- total: $O(m \lg \frac{n}{m})$
Double Binary Search

- let \( p_m = \lfloor \frac{m}{2} \rfloor \)
- search for \( M[p_m] \) in \( N \) using binary search
- let result be position \( p_n \)
- if \( M[p_m] = N[p_n] \) add \( M[p_m] \) to result
- continue recursively by intersecting
  - \( M[1, p_m] \cap N[1, p_n] \) and
  - \( M[1 + p_m, |M|] \cap N[1 + p_n, |N|] \)

Lemma: Running Time Double Binary Search

Intersecting two sorted lists of sizes \( m \) and \( n \) using a double binary search requires \( O(m \lg \frac{n}{m}) \) time.

Proof (Sketch)

- look at running time of binary search at each recursion depth
  - depth 0: \( \lg n \)
  - depth 1: \( 2 \lg \frac{n}{2} \)
  - depth 2: \( 4 \lg \frac{n}{4} \)
  - depth \( m \): \( m \lg \frac{n}{m} \)

Since depth of recursion is at most \( m \), this results in

\[
\sum_{i=0}^{\lfloor \frac{m}{2} \rfloor} (\lg \frac{n}{m} + i) = m(\lg \frac{n}{m} \sum_{i=0}^{\lfloor \frac{m}{2} \rfloor} \frac{1}{2^i} + \sum_{i=0}^{\lfloor \frac{m}{2} \rfloor} \frac{1}{2^i})
\]

total: \( O(m \lg \frac{n}{m}) \)

- example on board
Exponential Search

- assume that $M[1..i]$ have been processed and
- $M[i]$ is closest to $N[j]$ for some $j$
- now find $M[i + 1]$ in $N$ by comparing it to $N[j], N[j + 1], N[j + 2], N[j + 4], \ldots$ until
- $N[j + 2^k] \geq M[i + 1]$ if $N[j + 2^k] = M[i + 1]$, we are done with this iteration
- binary search for $M[i + 1]$ in $N[j + 2^{k-1}..j + 2^k]$
Exponential Search

- Assume that $M[1..i]$ have been processed and $M[i]$ is closest to $N[j]$ for some $j$.
- Now find $M[i + 1]$ in $N$ by comparing it to $N[j], N[j + 1], N[j + 2], N[j + 4], \ldots$ until
- $N[j + 2^k] \geq M[i + 1]$ if $N[j + 2^k] = M[i + 1]$, we are done with this iteration.
- Binary search for $M[i + 1]$ in $N[j + 2^{k-1}..j + 2^k]$.

Lemma: Running Time Exponential Search

Intersecting two sorted lists of sizes $m$ and $n$ using an exponential search requires $O(m \lg \frac{n}{m})$ time.
Exponential Search

- assume that $M[1..i]$ have been processed and
- $M[i]$ is closest to $N[j]$ for some $j$
- now find $M[i+1]$ in $N$ by comparing it to $N[j], N[j+1], N[j+2], N[j+4], \ldots$ until
- $N[j+2^k] \geq M[i+1]$ if $N[j+2^k] = M[i+1]$,
  we are done with this iteration
- binary search for $M[i+1]$ in $N[j+2^{k-1}..j+2^k]$

Lemma: Running Time Exponential Search

Intersecting two sorted lists of sizes $m$ and $n$ using a exponential search requires $O(m \lg \frac{n}{m})$ time.

Proof

- searching for each element $M[i]$ requires $O(\lg d_i)$ time
- $d_i$ is distance between $M[i-1]$ and $M[i]$ in $N$
- $O(\sum_{i}^m \lg d_i)$, which is maximal if $d_i = \frac{n}{m}$
- total: $O(m \frac{n}{m})$
Exponential Search

- assume that $M[1..i]$ have been processed and $M[i]$ is closest to $N[j]$ for some $j$
- now find $M[i + 1]$ in $N$ by comparing it to $N[j], N[j + 1], N[j + 2], N[j + 4], \ldots$ until $N[j + 2^k] \geq M[i + 1]$ if $N[j + 2^k] = M[i + 1]$
- we are done with this iteration
- binary search for $M[i + 1]$ in $N[j + 2^{k-1}..j + 2^k]$

Lemma: Running Time Exponential Search

Intersecting two sorted lists of sizes $m$ and $n$ using a exponential search requires $O(m \lg \frac{n}{m})$ time.

Proof

- searching for each element $M[i]$ requires $O(\lg d_i)$ time
- $d_i$ is distance between $M[i - 1]$ and $M[i]$ in $N$
- $O(\sum_i^m \lg d_i)$, which is maximal if $d_i = \frac{n}{m}$
- total: $O(m \frac{n}{m})$

example on board
Exponential Search

- assume that $M[1..i]$ have been processed and $M[i]$ is closest to $N[j]$ for some $j$
- $N[j + 2^k] \geq M[i + 1]$ if $N[j + 2^k] = M[i + 1]$, we are done with this iteration
- binary search for $M[i + 1]$ in $N[j + 2^{k-1}.j + 2^k]$

Proof

- searching for each element $M[i]$ requires $O(\lg d_i)$ time
- $d_i$ is distance between $M[i − 1]$ and $M[i]$ in $N$
- $O(\sum_i^m \lg d_i)$, which is maximal if $d_i = \frac{n}{m}$
- total: $O(m\frac{n}{m})$

Lemma: Running Time Exponential Search

Intersecting two sorted lists of sizes $m$ and $n$ using a exponential search requires $O(m \lg \frac{n}{m})$ time.

example on board

- works well if lists do not fit into main memory
- still not working with $\Delta$-encoding
Engineered Representations

Two-Level Representation

- store every $B$-th element of the list in top-level
- in addition to $\Delta$-encoded ids
- store original id for each sampled value in id-list
Engineered Representations

Two-Level Representation
- store every $B$-th element of the list in top-level
- in addition to $\Delta$-encoded ids
- store original id for each sampled value in id-list

Binary Search
- binary search on top-level
- scan on list in relevant interval

example on board 📚
## Engineered Representations

### Two-Level Representation
- store every $B$-th element of the list in top-level
- in addition to $\Delta$-encoded ids
- store original id for each sampled value in id-list

### Binary Search
- binary search on top-level
- scan on list in relevant interval

### Skipper [MZ96]
- scan top-level and
- go down in $\Delta$-encoded list as soon as possible
- avoids complex binary search control structure

- example on board

- example on board
Intersection with Randomized Inverted Indices [ST07]

- assume ids are in $[0, U)$ with $U = 2^u$
- ids have to be random
- choose tuning parameter $B$
- determine average bucket size
- given a list $N = [d_1, \ldots, d_n]$ and $k_N = \lceil \lg \frac{UB}{n} \rceil$
- per list, represent ids in
  - buckets $b_i^N$ containing
  - partial ids $\{d_j \mod 2^{k_N} : d_j/2^{k_N} = i\}$
- due to randomization, average bucket size is between $B/2$ and $B$
- elements in buckets can be $\Delta$-encoded
Intersection with Randomized Inverted Indices [ST07]

- assume ids are in \([0, U)\) with \(U = 2^{2^u}\)
- ids have to be random\(^1\) more details in paper
- choose tuning parameter \(B\) \(^2\) determine average bucket size
- given a list \(N = [d_1, \ldots, d_n]\) and \(k_N = \lceil \lg \frac{UB}{n} \rceil\)
- per list, represent ids in
  - buckets \(b_i^N\) containing
  - partial ids \(\{d_j \mod 2^{k_N} : d_j / 2^{k_N} = i\}\)
- due to randomization, average bucket size is between \(B/2\) and \(B\)
- elements in buckets can be \(\Delta\)-encoded

\(^1\) example on board

---

22/23 2022-01-10 Florian Kurpicz | Text Indexing | 10 Inverted Index

Institute for Theoretical Computer Science, Algorithmics II
assume ids are in $[0, U)$ with $U = 2^{2^u}$
ids have to be random $\uparrow$ more details in paper
choose tuning parameter $B$ $\uparrow$ determine average bucket size
given a list $N = [d_1, \ldots, d_n]$ and $k_N = \lceil \lg \frac{UB}{n} \rceil$
per list, represent ids in
- buckets $b^N_i$ containing
- partial ids $\{d_j \mod 2^{k_N} : d_j/2^{k_N} = i\}$
due to randomization, average bucket size is between $B/2$ and $B$
elements in buckets can be $\Delta$-encoded

Intersection
- for each element $M[i]$ find bucket of $N$
- can be same bucket as for $M[i-1]$, if so, continue at position of $M[i-1]$ in bucket $\uparrow$ continuing is important
- scan bucket until element $\geq M[i]$ is found
- if equal, output $M[i]$

example on board 🔴
Intersection with Randomized Inverted Indices [ST07]

- Assume ids are in \([0, U]\) with \(U = 2^{2u}\).
- Ids have to be random \(^1\) more details in paper.
- Choose tuning parameter \(B \uparrow\) determine average bucket size.
- Given a list \(N = [d_1, \ldots, d_n]\) and \(k_N = \lceil \lg \frac{UB}{n} \rceil\).
- Per list, represent ids in
  - Buckets \(b_i^N\) containing
  - Partial ids \(\{d_j \mod 2^{k_N} : d_j/2^{k_N} = i\}\).
- Due to randomization, average bucket size is between \(B/2\) and \(B\).
- Elements in buckets can be \(\Delta\)-encoded.

Intersection

- For each element \(M[i]\) find bucket of \(N\).
- Can be same bucket as for \(M[i - 1]\), if so, continue at position of \(M[i - 1]\) in bucket.
- Continuing is important.
- Scan bucket until element \(\geq M[i]\) is found.
- If equal, output \(M[i]\).

Lemma: Running Time

Intersecting two sorted lists of sizes \(m\) and \(n\) using a randomized inverted indices requires \(O(m + \min\{n, Bm\})\) time.
Conclusion and Outlook

This Lecture

- inverted index
- space efficient encodings of document lists
- efficient intersection algorithms

Linear Time Construction

```
  ST   SA
   ^   |
    |   LCP
    |   |
    |   WT
    |   |
    |   BWT
    |   |
    |   FM-Index
    |   r-Index
  LZ
```

Next Lecture

top-
k retrieval
Conclusion and Outlook

This Lecture
- inverted index
- space efficient encodings of document lists
- efficient intersection algorithms

Next Lecture
- top-$k$ retrieval

Linear Time Construction

ST  SA  WT
LZ  LCP  BWT
FM-Index
$r$-Index
Oral Exam

Remaining Lectures
- 17.01. top-k retrieval
- 24.01. longest common extensions
- 31.01. TBD & Q&A
- 07.02. project presentation

- 09.03. is default date for all(?) exams
- 08.02. possible, but needs good arguments

- any questions

- 20 minute long oral exam
- conducted by Prof. Sanders and me
- most likely virtual  

Technic check
Bibliography I


