Karlsruhe Institute of Technology

Text Indexing

Lecture 10: Inverted Index

Florian Kurpicz

PINGO

https://pingo.scc.kit.edu/569917

The Inverted Index

Karlsruhe Institute of Technology

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

Karlsruhe Institute of Technology

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
```

term t	f_t	$L(t)$
and	1	$[6]$
big	2	$[2, 3]$
dark	1	$[6]$
...	...	...
had	1	$[3]$
house	2	$[2, 3]$
in	5	$[1, 2, 3, 5, 6]$
...	...	...

The Inverted Index

Karlsruhe Institute of Technology

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

term t	f_t	$L(t)$
and	1	[6]
big	2	[2, 3]
dark	1	[6]
...	...	...
had	1	[3]
house	2	[2, 3]
in	5	[1, 2, 3, 5, 6]
...	...	...

The Inverted Index: Queries

Karlsruhe Institute of Technology

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 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: 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: $M \cup N$

```
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
```

term t	f_t	$L(t)$
and	1	[6]
big	2	[2, 3]
dark	1	[6]
...	...	...
had	1	[3]
house	2	[2, 3]
in	5	[1, 2, 3, 5, 6]
...	...	...

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 $t_1 t_2$ ℹ all previous discussed indices can do so

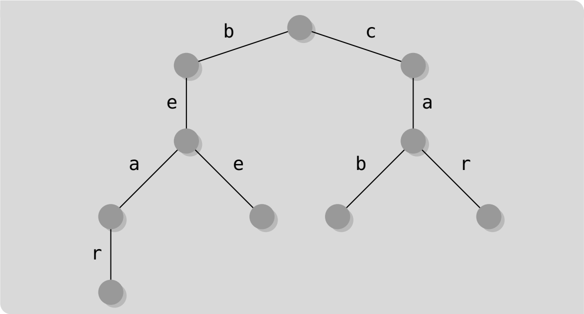
```
1 The old night keeper keeps the keep in the town
2 In the big old house in the big old gown
3 The house in the town had the big old keep
4 Where the old night keeper never did sleep
5 The night keeper keeps the keep in the night
6 And keeps in the dark and sleeps in the light
```

term t	f_t	$L(t)$
and	1	[6]
big	2	[2, 3]
dark	1	[6]
...	...	...
had	1	[3]
house	2	[2, 3]
in	5	[1, 2, 3, 5, 6]
...	...	...

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
- $h(t[1]\dots t[\ell]) = ((\sum_{i=1}^{\ell} a_i \cdot t[i]) \bmod p) \bmod m$
- 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

Binary Codes

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

Now

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

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: Δ-Encoding

A **Δ-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: $\delta_i = d_i - d_{i-1}$

Just ids:

- $N = [4, 11, 12, 30, 42, 54]$

Δ-encoded

- $N = [4, 7, 1, 18, 12, 12]$

Definition: Δ-Encoding

A **Δ-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

Karlsruhe Institute of Technology

- 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: $\delta_i = d_i - d_{i-1}$

Just ids:

- $N = [4, 11, 12, 30, 42, 54]$

Δ-encoded

- $N = [4, 7, 1, 18, 12, 12]$

Definition: Δ-Encoding

A **Δ-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

Unary Encoding

Karlsruhe Institute of Technology

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

Unary Encoding

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

Just ids:

- $N = [4, 11, 12, 30, 42, 54]$

Δ-encoded

- $N = [4, 7, 1, 18, 12, 12]$

Unary Codes:

- $N = [1110111111001^{17}01^{11}0111111111110]$

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 $(x)_3$

- $|(x)_3| = 2\lfloor \lg_3(x-1) \rfloor + 2$

Ternary Encoding

Karlsruhe Institute of Technology

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 $(x)_3$

- $|(x)_3| = 2\lfloor \lg_3(x-1) \rfloor + 2$

Just ids:

- $N = [4, 11, 12, 30, 42, 54]$

Δ-encoded

- $N = [4, 7, 1, 18, 12, 12]$

Unary Codes:

- $N = [1110111111001^{17}01^{11}011111111110]$

Ternary Codes:

- $N = [100011\ 100011\ 01101011\ 01001011\ 01001011]$

Fibonacci Encoding

Karlsruhe Institute of Technology

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

Karlsruhe Institute of Technology

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 $(x)_\phi$

- 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

Fibonacci Encoding

Karlsruhe Institute of Technology

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 $(x)_\phi$

- 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

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

- 4: 00 100
- 7: 00 111
- 1: 1
- 18: 0000 10010
- 12: 000 1000

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
- $|(x)_\delta| = 2\lfloor lg(\lfloor \lg x \rfloor + 1) \rfloor + 1 + \lfloor \lg x \rfloor$ bits

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
- $|(x)_\delta| = 2\lfloor lg(\lfloor \lg x \rfloor + 1) \rfloor + 1 + \lfloor \lg x \rfloor$ bits

Elias-γ

- 4: 00 100
- 7: 00 111
- 1: 1
- 18: 0000 10010
- 12: 000 1000

Elias-δ

- 4: 0 11 00
- 7: 0 11 11
- 1: 1
- 18: 00 101 0010
- 12: 00 100 100

Golomb Encoding [Gol66]

Definition: Golomb Code

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

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

with

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

where $(r)_2$ depends on its size

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

Golomb Encoding [Gol66]

Definition: Golomb Code

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

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

with

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

where $(r)_2$ depends on its size

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

- 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

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

with

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

where $(r)_2$ depends on its size

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

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

- for $b = 5$, there are 4 remainders: $00, 01, 100, 101$, and 110
- $2^{\lfloor \lg 5 \rfloor - 1} = 2$
- $0, 1 < 2$: $00 and 01$ 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 $t_1, \dots, t_k$
- 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 Δ encoded

Naive Scanning

Zipper

- scan both lists as in binary merging

Naive Scanning

Karlsruhe Institute of Technology

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

Karlsruhe Institute of Technology

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

Naive Scanning

Karlsruhe Institute of Technology

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 $\max\{id\colon id \in N\} <$ some element in M, then all elements in N are compared
- resulting in $O(n+m)$ time

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

Naive Scanning

Karlsruhe Institute of Technology

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

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

- example on the board

Binary Search (1/2)

Simple Binary Search

- search each document in M in N using binary search

Binary Search (1/2)

Karlsruhe Institute of Technology

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.

Binary Search (1/2)

Karlsruhe Institute of Technology

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

- example on the board

Binary Search (1/2)

Karlsruhe Institute of Technology

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

- example on the board

- binary search not work with Δ-encoding

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
  - $M[1 + p_m, |M|] \cap N[1 + p_n, |N|]$

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.

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.

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

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.

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

- example on board

Exponential Search

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 $N[j], N[j+1], N[j+2], N[j+4], \dots$ until
- $N[j+2^k] \geq M[i+1]$ ℹ 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

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.

Exponential Search

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
- $N[j+2^k] \geq M[i+1]$ ℹ 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

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

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
- $N[j+2^k] \geq M[i+1]$ ℹ 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

- works well if lists do not fit into main memory
- still not working with Δ-encoding

Engineered Representations

Karlsruhe Institute of Technology

Two-Level Representation

- store every B-th element of the list in top-level
- in addition to Δ-encoded ids
- store original id for each sampled value in id-list

Engineered Representations

Karlsruhe Institute of Technology

Two-Level Representation

- store every B-th element of the list in top-level
- in addition to Δ-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 Δ-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

Skipper [MZ96]

- scan top-level and
- go down in Δ-encoded list as soon as possible

- avoids complex binary search control strucutre

- example on board

Intersection with Randomized Inverted Indices [ST07]

- assume ids are in $[0, U)$ with $U = 2^{2^u}$
- ids have to be random ℹ more details in paper
- choose tuning parameter B ℹ 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 $\{d_j \bmod 2^{k_N} : d_j / 2^{k_N} = i\}$
- due to randomization, average bucket size is between $B/2$ and B
- elements in buckets can be Δ-encoded

Intersection with Randomized Inverted Indices [ST07]

- assume ids are in $[0, U)$ with $U = 2^{2^u}$
- ids have to be random ℹ more details in paper
- choose tuning parameter B ℹ 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 $\{d_j \bmod 2^{k_N} : d_j / 2^{k_N} = i\}$
- due to randomization, average bucket size is between $B/2$ and B
- elements in buckets can be Δ-encoded

- example on board

Intersection with Randomized Inverted Indices [ST07]

- assume ids are in $[0, U)$ with $U = 2^{2^u}$
- ids have to be random ℹ more details in paper
- choose tuning parameter B ℹ 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 $\{d_j \bmod 2^{k_N} : d_j / 2^{k_N} = i\}$
- due to randomization, average bucket size is between $B/2$ and B
- elements in buckets can be Δ-encoded

- example on board

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 ℹ continuing is important
- scan bucket until element $\geq M[i]$ is found
- if equal, output $M[i]$

Intersection with Randomized Inverted Indices [ST07]

- assume ids are in $[0, U)$ with $U = 2^{2^u}$
- ids have to be random ℹ more details in paper
- choose tuning parameter B ℹ 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 $\{d_j \bmod 2^{k_N} : d_j / 2^{k_N} = i\}$
- due to randomization, average bucket size is between $B/2$ and B
- elements in buckets can be Δ-encoded

- example on board

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 ℹ continuing is important
- scan bucket until element $\geq M[i]$ is found
- if equal, output $M[i]$

Lemma: Running Time

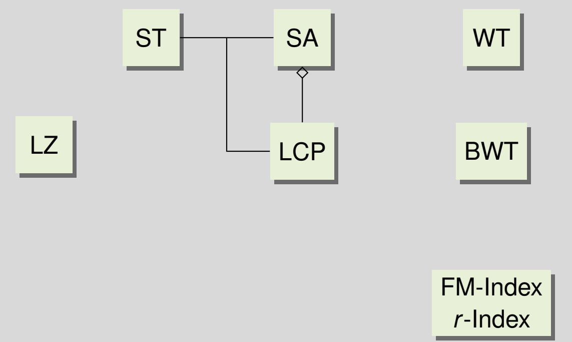
Intersecting two sorted lists of sizes m and n using a randomized inverted indices requires $O(m + \min\{n, Bm\})$ time.

Conclusion and Outlook

This Lecture

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

Linear Time Construction



Conclusion and Outlook

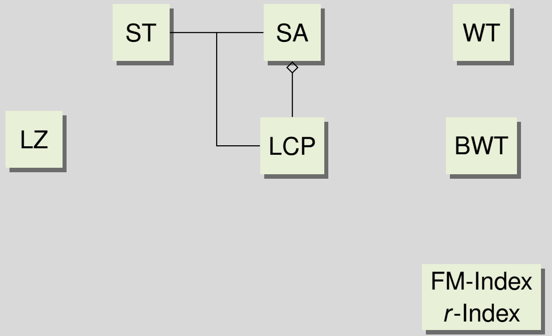
This Lecture

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

Next Lecture

- top-k retrieval

Linear Time Construction



Oral Exam

Karlsruhe Institute of Technology

Remaining Lectures

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

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

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

- any questions

Bibliography I

[BS05] Ricardo A. Baeza-Yates and Alejandro Salinger. “Experimental Analysis of a Fast Intersection Algorithm for Sorted Sequences”. In: *SPIRE*. Volume 3772. Lecture Notes in Computer Science. Springer, 2005doi10.1007/11575832_2, pages 13–24.

[Eli75] Peter Elias. “Universal Codeword Sets and Representations of the Integers”. In: *IEEE Trans. Inf. Theory* 21.2 (1975), pages 194–203. DOI: 10.1109/TIT.1975.1055349.

[Gol66] Solomon W. Golomb. “Run-length Encodings (Corresp.)”. In: *IEEE Trans. Inf. Theory* 12.3 (1966), pages 399–401. DOI: 10.1109/TIT.1966.1053907.

[MZ96] Alistair Moffat and Justin Zobel. “Self-Indexing Inverted Files for Fast Text Retrieval”. In: *ACM Trans. Inf. Syst.* 14.4 (1996), pages 349–379.

[ST07] Peter Sanders and Frederik Transier. “Intersection in Integer Inverted Indices”. In: *ALENEX*. SIAM, 2007. DOI: 10.1137/1.9781611972870.7.