Advanced Data Structures

Lecture 09: String B-Trees and Temporal Data Structures 2

Florian Kurpicz

PINGO

https://pingo.scc.kit.edu/172581

External Memory Model [AV88]

Definition: External Memory Model

- internal memory of M words
- instances of size $N \gg M$
- unlimited external memory
- transfer blocks of size B between memories

- measure number of blocks I/Os
- scanning N elements: $\Theta(N/B)$
- sorting N elements: $\Theta(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B})$

Set of Strings

- alphabet Σ of size σ
- k strings $\{s_1, \ldots, s_k\}$ over the alphabet Σ
- total size of strings is $N = \sum_{i=1}^{k} |s_i|$
- queries ask for pattern P of length m

String Dictionary

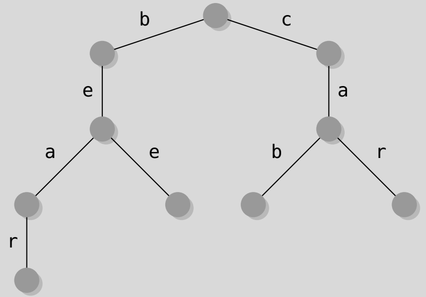
Given a set $S \subseteq \Sigma^\star$ of prefix-free strings, we want to answer:

- is $x \in \Sigma^\star$ in S
- add $x \notin S$ to S
- remove $x \in S$ from S
- predecessor and successor of $x \in \Sigma^\star$ in S

Definition: Trie

Given a set $S = \{S_1, \ldots, S_k\}$ of prefix-free strings, a trie is a labeled rooted tree $G = (V, E)$ with:

1. k leaves
2. $\forall S_i \in S$ there is a path from the root to a leaf, such that the concatenation of the labels is S_i
3. $\forall v \in V$ the labels of the edges $(v, \cdot)$ are unique



$S = \{\texttt{bear}, \texttt{bee}, \texttt{cab}, \texttt{car}\}$

Theoretical Comparison

Representation	Query Time (Contains)	Space in Words
arrays of variable size	$O(m \cdot \sigma)$	$O(N)$
arrays of fixed size	$O(m)$	$O(N \cdot \sigma)$
hash tables	$O(m)$ w.h.p.	$O(N)$
balanced search trees	$O(m \cdot \lg \sigma)$	$O(N)$
weight-balanced search trees	$O(m + \lg k)$	$O(N)$
two-levels with weight-balanced search trees	$O(m + \lg \sigma)$	$O(N)$

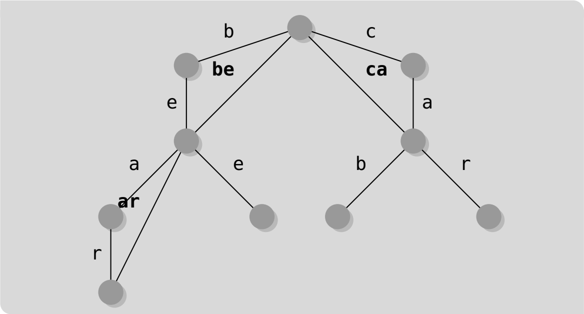
- more details in lecture Text Indexing

Compact Trie

- tries have unnecessary nodes
- branchless paths can be removed
- edge labels can consist of multiple characters

Definition: Compact Trie

- A compact trie is a trie where all branchless paths are replaced by a single edge.
- The label of the new edge is the concatenation of the replaced edges' labels.



(Recap) B-Trees

- search tree with out-degree in $[b, 2b)$
- works well in external memory
- uses separators to find subtree
- can be dynamic
- who knows B-trees **PINGO**

- example on the board

From Atomic Values to Strings

- strings take more time to compare
- load as few strings from disk as possible

String B-Tree [FG99]

- strings are stored in EM
- strings are identified by starting positions

- B-tree layout for sorted suffixes ℹ identified by position
- at least $b = \Theta(B)$ children
- tree height $O(\log_B N)$

- given node v
- $L(v)$ is lexicographically smallest string at v
- $R(v)$ is lexicographically largest string at v

- given node v with children $v_0, \ldots, v_k$ with $k \in [b, 2b)$
- inner: store separators $L(v_0), R(v_0), \ldots, L(v_k), R(v_k)$
- leaf: store strings and link leaves

Search in String B-Tree

- task: find all occurrences of pattern P
- two traversals of String B-Tree
- identify leftmost/rightmost occurrence
- output all strings in $O(occ/B)$

- at every node with children $v_0, \dots, v_k$
- binary search for P in $L(v_0), \dots, R(v_k)$
  - if $R(v_i) < P < L(v_{i+1})$: not found
  - if $L(v_i) \leq P \leq R(v_i)$: continue in v_i

Lemma: String B-Tree

Using a String B-tree, a pattern P can be found in a set of strings with total length N in $O(|P|/B \log N)$ I/Os

Proof (Sketch)

- String B-Tree has height $\log_B N$
- load separators of node: $O(1)$ I/O
- load strings for binary search: $O(|P|/B)$ I/Os
- total: $O(\log_B N \cdot \log B \cdot |P|/B) = O(|P|/B \log N)$ I/Os

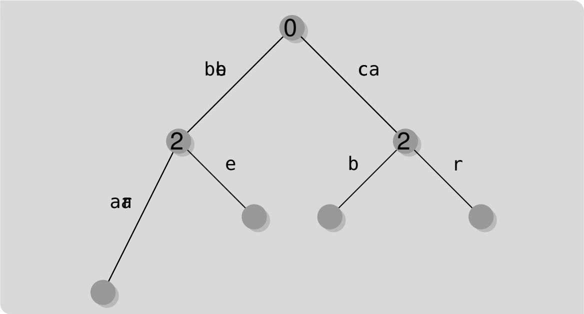
Improving String B-Tree with Patricia Tries (1/2)

Karlsruhe Institute of Technology

Patricia Trie

- for strings $S = \{S_0, \ldots, S_{k-1}\}$
- a compact trie where only branching characters are stored
- additionally the string depth is stored
- size $O(k)$ for k strings

- search requires two steps
- first blind search using only trie
- blind search can result in false matches
- second a comparison with resulting string
- use any leaf after matching pattern



- How do Patricia tries help? **PINGO**

Improving String B-Tree with Patricia Tries (2/2)

- in each inner node build Patricia trie for separators
- if blind search finds leaf w
- compute $L = lcp(P, w)$
- let u be first node on root-to-w path with $d \geq L$

$d = L$

- find matching children v_i and v_{i+1} of w with
- branching characters $c_i < P[L+1] < c_{i+1}$
- example on the board

$d > L$

- consider next branching character c on path
- if $P[L+1] < c$ continue in leftmost leaf
- if $P[L+1] > c$ continue in rightmost leaf

Searching in Improved String B-Tree

- at every node with children $v_0, \dots, v_k$
- load Patricia trie for $L(v_0), \dots, R(v_k)$
- search Patricia trie for w ℹ result of blind search
- load one string and compare with P
- identify child and continue

- How can this be improved even further?
  PINGO

Lemma: String B-Tree with PTs

Using a string B-tree with Patricia tries, a pattern P can be found in a set of strings with total length N with $O(|P|/B \log_B N)$ I/Os

Proof (Sketch)

- loading PT: $O(1)$ I/Os
- blind search: no I/Os
- loading one string: $O(|P|/B)$ I/Os
- identify child: no I/Os
- total $O(|P|/B \log_B N)$ I/Os

Improving Search with LCP-Values

- search for pattern in nodes
- path in String B-tree $p_0, p_1, p_2, \ldots$
- in Patricia tries PT_{p_i} compute $L = lcp(P, w)$
- all strings in p_i have prefix $P[0..L)$
- do not compare previously matched characters
- load only $|P| - L$ characters at next node
- pass L down the String B-tree

Lemma: String B-Tree with PTs and LCP

Using a String B-tree with Patricia tries and passing down the LCP-value, a pattern P can be found in a set of strings with total length N in $O(|P|/B + \log_B N)$ I/Os

Proof (Sketch)

- passing down LCP-value: no I/Os
- telescoping sum $\sum_{i \leq h} \frac{L_i - L_{i-1}}{B}$
- $h = \log_B N$ height of String B-tree
- L_i is LCP-value on Level i
- $L_0 = 0$ and $L_h \leq |P|$
- total: $O(|P|/B + \log_B N)$ I/Os

Recap: Persistent Data Structures

- lecture based on: `http://courses.csail.mit.edu/6.851/spring12/lectures/L01`

Persistence

- change in the past creates new branch
- similar to version control
- everything old/new remains the same

Retroactivity

- change in the past affects future
- make change in earlier version changes all later versions

Definition: Partial Persistence

Only the latest version can be updated

Definition: Full Persistence

Any version can be updated

Definition: Confluent Persistence

Like full persistence, but two versions can be combined to a new version

Definition: Functional

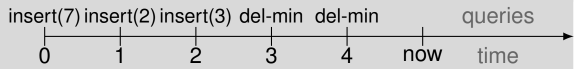
Nodes cannot be modified, only new nodes can be created

Retroactive Data Structures

Operations

- $\texttt{INSERT}(t, \mathit{operation})$: insert operation at time t
- $\texttt{DELETE}(t)$: delete operation at time t
- $\texttt{QUERY}(t, \mathit{query})$: ask *query* at time t

- for a priority queue updates are
  - insert
  - delete-min
- time is integer ⓘ for simplicity otherwise use order-maintenance data structure



Definition: Partial Retroactivity

`QUERY` is only allowed for $t = \infty$ ⓘ now

Definition: Full Retroactivity

`QUERY` is allowed at any time t

Definition: Nonoblivious Retroactivity

`INSERT`, `DELETE`, and `QUERY` at any time t but also identify changed `QUERY` results

Easy Cases: Partial Retroactivity

Karlsruhe Institute of Technology

- commutative updates
- invertible updates
  - operation op^{-1} such that $op^-1(op(\cdot)) = \emptyset$
  - `DELETE` becomes `INSERT` inverse operation
- makes partial retroactivity easy
- $\texttt{INSERT}(t, operation) = \texttt{INSERT}(\infty, operation)$
- $\texttt{DELETE}(t, op) = \texttt{INSERT}(\infty, op^{-1})$

Partial Retroactivity

- hashing
- dynamic dictionaries
- array with updates only ℹ $A[i] += value$

Search Problems

Definition: Search Problem

A search problem is a problem on a set S of objects with operations *insert*, *delete*, and $query(x, S)$

Definition: Decomposable Search Problem

A decomposable search problem is a search problem, with

- $query(x, A \cup B) = f(query(x, A), query(x, B))$
- with f requiring $O(1)$ time

- which decomposable search problem have we seen **PINGO**

- predecessor and successor search
- range minimum queries

- nearest neighbor
- point location
- ...

- these types of problems are also "easy"

Decomposable Search Problems: Full Retroactivity

Lemma: Full Retroactivity for DSP

Every decomposable search problems can be made fully retroactive with a $O(\log m)$ overhead in space and time, where m is the number of operations

Proof (Sketch)

- use balances search tree segment tree
- each leaf corresponds to an update
- node n corresponds to interval of time $[s_n, e_n]$
- if an object exists in the time interval $[s, e]$, then it appears in node n if $[s_n, e_n] \subseteq [s, e]$ if none of n's ancestors' are $\subseteq [s, e]$
- each object occurs in $O(\log n)$ nodes

Proof (Sketch, cnt.)

- to query find leaf corresponding to t
- look at ancestors to find all objects
- $O(\log m)$ results which can be combined in $O(\log m)$ time

- data structure is stored for each operation!
- $O(\log m)$ space overhead!

General Full Retroactivity

Karlsruhe Institute of Technology

Lemma: Lower Bound

Rewinding m operations has a lower bound of $\Omega(m)$ overhead

- general case

Proof (Sketch)

- two values X and Y
- initially $X = \emptyset$ and $Y = \emptyset$
- supported operations
  - $X = x$
  - $Y += value$
  - $Y = X \cdot Y$
  - *query* Y

Proof (Sketch, cnt.)

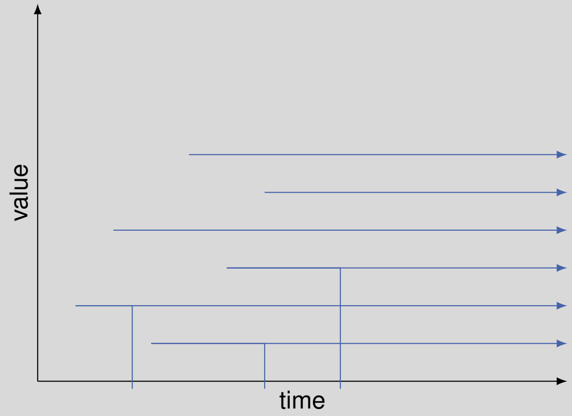
- perform operations
  - $Y += a_n$
  - $Y = X \cdot Y$
  - $Y += a_{n=1}$
  - $Y = X \cdot Y$
  - $\dots$
  - $Y += a_0$
- what are we computing here? **PINGO**
- $Y = a_n \cdot X^n + a_{n-1} X^{n-1} + \dots + a_0$
- evaluate polynomial at $X = x$ using `t=0,X=x`
- this requires $\Omega(n)$ time [FHM01]

Priority Queues: Partial Retroactivity (1/6)

- priority queue with
  - insert
  - delete-min
- delete-min makes PQ non-commutative

Lemma: Partial Retroactive PQ

A priority queue can be partial retroactive with only $O(\log m)$ overhead per partially retroactive operation

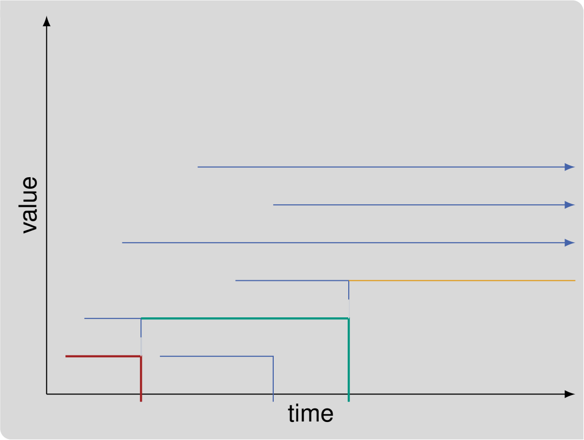


Priority Queues: Partial Retroactivity (2/6)

- what is the problem with
  - `INSERT(t,`delete-min())
  - `INSERT(t,`insert`(i))`

- `INSERT(t,`delete-min()) creates chain-reaction
- `INSERT(t,`insert`(i))` creates chain-reaction

- can we solve `DELETE(t,`delete-min`())` using `INSERT(t,`insert`(i))`? **PINGO**
- insert deleted minimum right after deletion



Priority Queues: Partial Retroactivity (3/6)

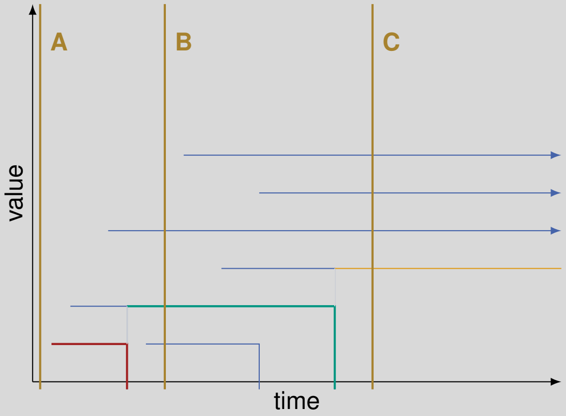
- let Q_t be elements in PQ at time t

- what values are in Q_∞? partial retroactivity
- what value inserts $\texttt{INSERT}(t, \text{insert}(v))$ in Q_∞
- values is $\max\{v, v' : v' \text{ deleted at time} \geq t\}$
- maintaining deleted elements is hard can change a lot

Definition: Bridge

A time t' is a bridge if $Q_{t'} \subseteq Q_\infty$

- all elements present at t' are present at t_∞



- what times are bridges? **PINGO**

Priority Queues: Partial Retroactivity (4/6)

Lemma: Deletions after Bridges

If time t' is closest bridge preceding time t, then

$$\max\{v' : v' \text{ deleted at time} \geq t\}$$

$$=$$

$$\max\{v' \notin Q_\infty : v' \text{ inserted at time} \geq t'\}$$

Proof (Sketch)

- $\max\{v' \notin Q_\infty : v' \text{ inserted at time} \geq t'\} \in \{v' : v' \text{ deleted at time} \geq t\}$
  - if maximum value is deleted between t' and t
  - then this time is a bridge
  - contradicting that t' is bridge preceding t

Proof (Sketch, cnt.)

- $\max\{v' : v' \text{ deleted at time} \geq t\} \in \{v' \notin Q_\infty : v' \text{ inserted at time} \geq t'\}$
  - if v' is deleted at some time $\geq t$
  - then it is not in Q_∞

- what values are in Q_∞? ℹ partial retroactivity
- what value inserts $\texttt{INSERT}(t, \text{insert}(v))$ in Q_∞
- $\max\{v, v' \notin Q_\infty : v' \text{ inserted at time} \geq t'\}$

Priority Queues: Partial Retroactivity (5/6)

- keep track of inserted values
- use balanced binary search trees for $O(\log m)$ overhead

- BBST for Q_∞ ℹ changed for each update
- BBST where leaves are inserts ordered by time augmented with
  - for each node x store $\max\{v' \notin Q_\infty : v' \text{ inserted in subtree of } x\}$
- BBST where leaves are all updates ordered by time augmented with
  - leaves store 0 for inserts with $v \in Q_\infty$, 1 for inserts with $v \notin Q_\infty$ and -1 for delete-mins
  - inner nodes store subtree sums

- how can we find bridges? **PINGO**
- use third BBST and find prefix of updates summing to 0
- requires $O(\log n)$ time as we traverse tree at most twice
- this results in bridge t'

- use second BBST to identify maximum value not in Q_∞ on path to t'
- since BBST is augmented with these values, this requires $O(\log n)$ time

- update all BBSTs in $O(\log n)$ time

Priority Queues: Partial Retroactivity (6/6)

Lemma: Partial Retroactive PQ

A priority queue can be partial retroactive with only $O(\log m)$ overhead per partially retroactive operation

- requires three BBSTs
- updates need to update all BBSTs

Conclusion and Outlook

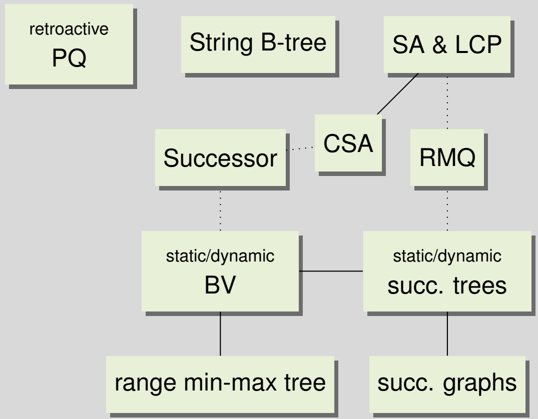
This Lecture

- string B-tree
- retroactive data structures

Next Lecture

- learned data structures

Advanced Data Structures



Bibliography I

[AV88] Alok Aggarwal and Jeffrey Scott Vitter. "The Input/Output Complexity of Sorting and Related Problems". In: *Commun. ACM* 31.9 (1988), pages 1116–1127. DOI: 10.1145/48529.48535.

[FG99] Paolo Ferragina and Roberto Grossi. "The String B-tree: A New Data Structure for String Search in External Memory and Its Applications". In: *J. ACM* 46.2 (1999), pages 236–280. DOI: 10.1145/301970.301973.

[FHM01] Gudmund Skovbjerg Frandsen, Johan P. Hansen, and Peter Bro Miltersen. "Lower Bounds for Dynamic Algebraic Problems". In: *Inf. Comput.* 171.2 (2001), pages 333–349. DOI: 10.1006/inco.2001.3046.