

Text Indexing

Lecture 01: Tries

Florian Kurpicz

The slides are licensed under a Creative Commons Attribution-ShareAlike 4.0 International License © ⓘ ⓘ: www.creativecommons.org/licenses/by-sa/4.0 | commit 224e27c compiled at 2022-10-24-13:00



<https://pingo.scc.kit.edu/952701>

Preliminaries (1/2)

Definition: Text

- let Σ be an **alphabet**
- $T \in \Sigma^*$ is a text
- $|T| = n$ is the length of the string
- $T = T[1]T[2]\dots T[n]$

Definition: Alphabet Types

- **constant size alphabet**: finite set not depending on n
- **integer alphabet**: alphabet is $\{1, \dots, \sigma\}$ and fits into constant number of words
- **finite alphabets**: alphabet of finite size

Preliminaries (2/2)

Definition: Substring, Prefix, and Suffix

Given a text $T = T[1]T[2] \dots T[n]$ of length n :

- $T[i..j] = T[i] \dots T[j]$ is called a **substring**,

a	b	b	a	a	b	b	a	\$
---	---	---	---	---	---	---	---	----

- $T[1..i]$ is called a **prefix**, and

a	b	b	a	a	b	b	a	\$
---	---	---	---	---	---	---	---	----

- $T[i..n]$ is called a **suffix** of T .

a	b	b	a	a	b	b	a	\$
---	---	---	---	---	---	---	---	----

Sentinel for Simplicity

Given a text T of length n over an alphabet Σ .

- we assume that $T[n] = \$$ with
- $\$ \notin \Sigma$ and $\$ < \alpha$ for all $\alpha \in \Sigma$
- otherwise, suffix can be prefix of another suffix

1	2	3	4	5	6	7	8
a	b	b	a	a	b	b	a

- $T[1..n] = abbaabba$ and $T[5..n] = abba$

Definition: Prefix-Free

A string is **prefix-free** if no suffix is a prefix of another suffix

String Dictionary

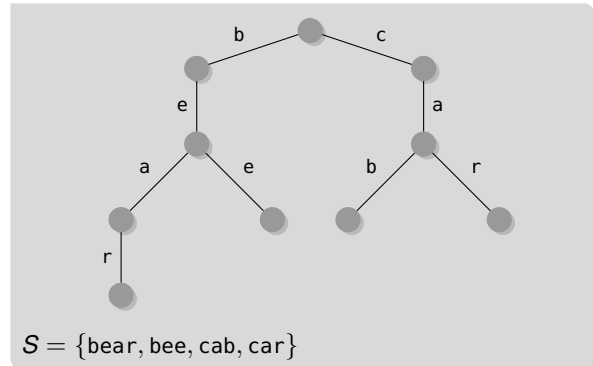
Given a set $S \subseteq \Sigma^*$ of **prefix-free** strings, we want to answer:

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

Definition: Trie

Given a set $S = \{S_1, \dots, S_k\}$ 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



Queries: Insert, Contains, and Delete a Pattern

Same for all

- start at root and follow existing children

Contains

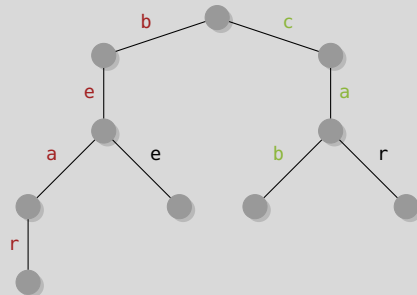
- is leaf found and whole pattern is matched

Delete

- if leaf is found backtrack and delete unique path
 - otherwise not found

Insert

- insert rest of pattern **!** prefix-free

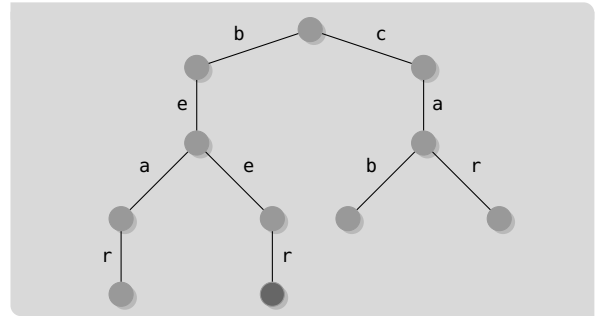


$S = \{\text{bear, bee, cab, car}\}$

- is **cab** in S
- remove **bear** from S
- how can we find the predecessor of **can**?

Why Prefix-Free

- insert beer
- bee cannot be found
- remember which node refers to a string
- or (much preferred) make strings prefix free



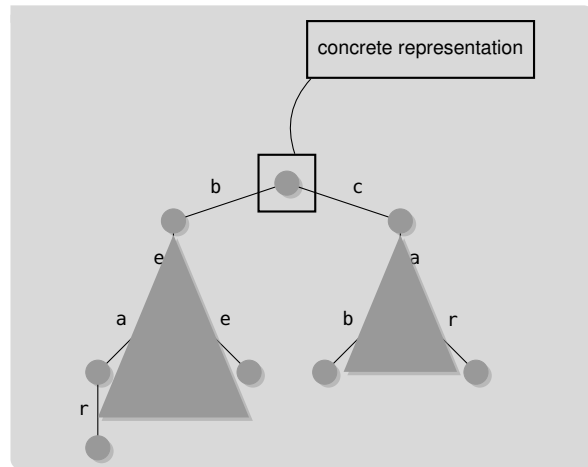
Next Steps

Setting


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

We Want to Know

- query times
 - space requirements
-
- both depend on the representation of children
 - look at different representations



Arrays of Variable Size

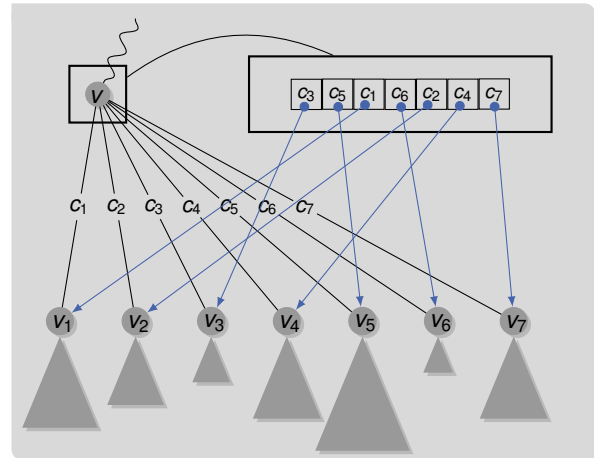
- store children (character and pointer) in the order they are added
- to find child scan array
- to delete child swap with last and remove last
 - children are not ordered
-  **PINGO** query time?

Query Time (Contains)


- $O(m \cdot \sigma)$

Space

- $O(N)$ words



Arrays of Fixed Size

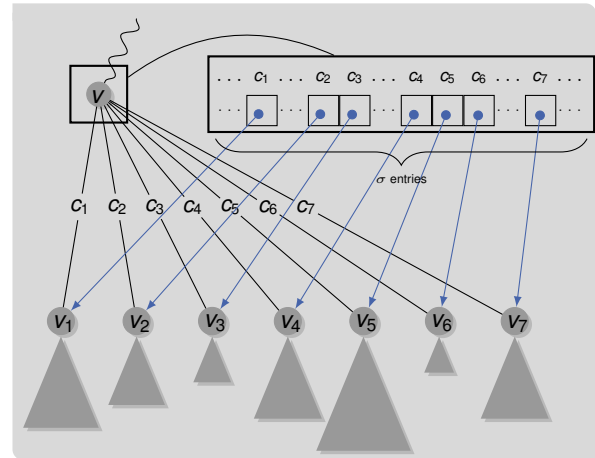
- children (pointer) are stored in arrays of size σ
- use null to mark non-existing children
- finding and deleting children is trivial
-  PINGO query time?

Query Time (Contains)


- $O(m)$ ⓘ optimal

Space

- $O(N \cdot \sigma)$ words ⓘ very bad



Hash Tables

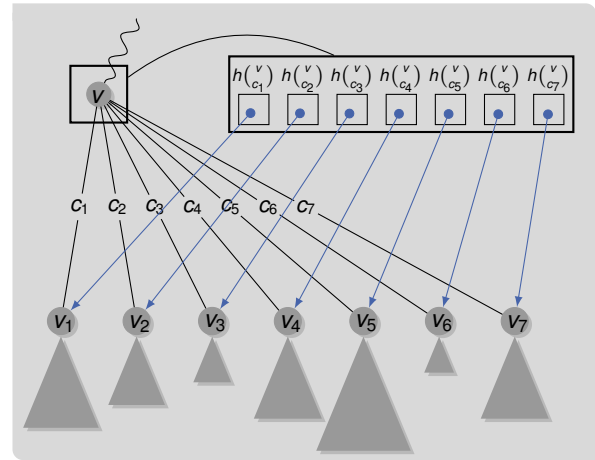
- either use a hash table per node
 - has overhead
- or use global hash table for whole trie
-  **PINGO** query time?

Query Time (Contains)


- $O(m)$ w.h.p.

Space

- $O(N)$ words



Balanced Search Trees

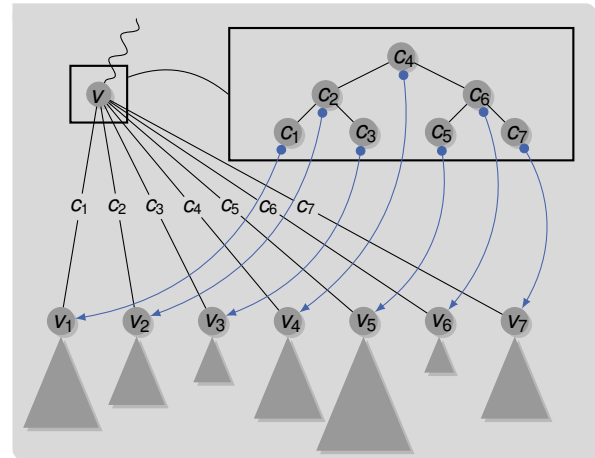
- children are stored in balanced search trees
- e.g., AVL tree, red-black tree, ...
- in static setting sorted array and binary search
-  PINGO query time?

Query Time (Contains)

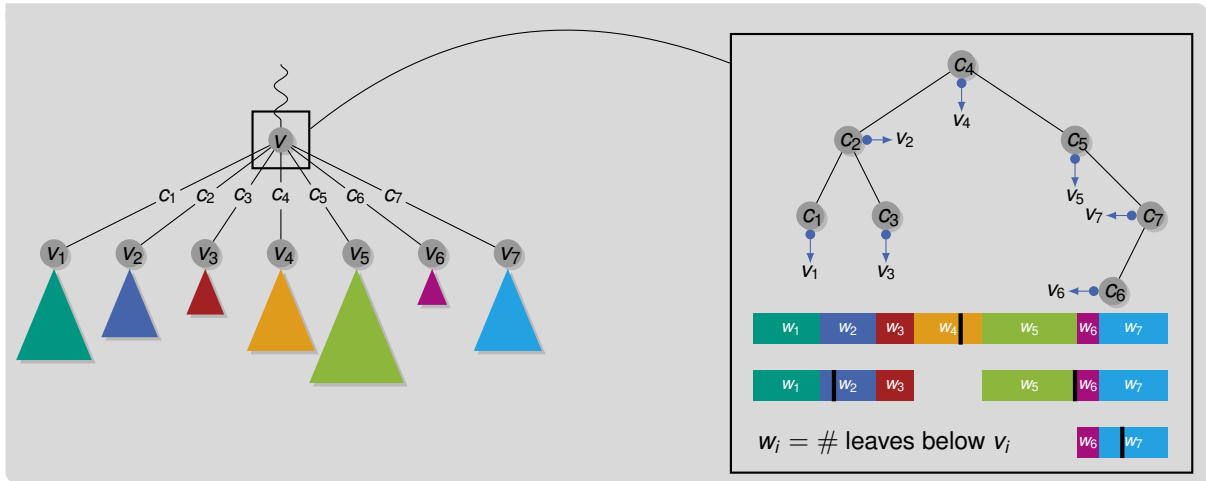
- $O(m \cdot \lg \sigma)$

Space


- $O(N)$ words



Weight-Balanced Search Trees (1/2)



Weight-Balanced Search Trees (2/2)

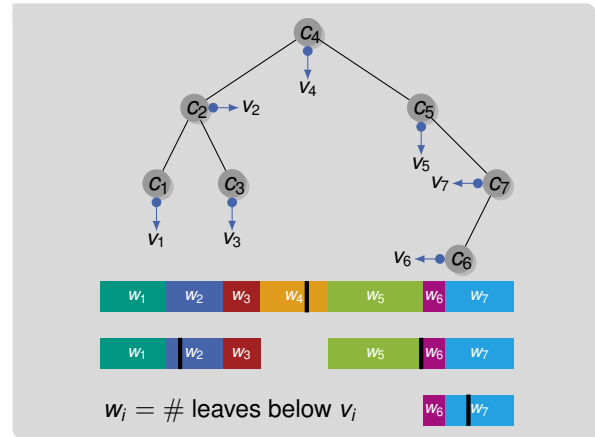
- use weight-balanced search trees at each node
-  PINGO query time?

Query Time (Contains)


- $O(m + \lg k)$
- match character of pattern
- or halve number of strings

Space

- $O(N)$ words

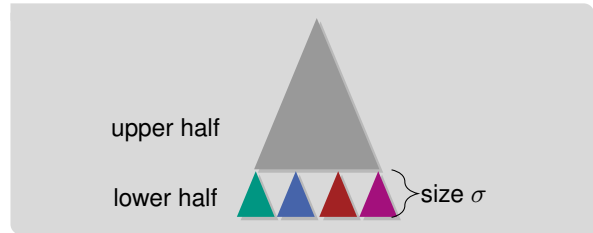


Two-Levels with Weight-Balanced Search Trees

- split tree into upper and lower half
- lower half deepest nodes such that subtrees have size $O(\sigma)$
- weight-balanced search trees for lower half
- fixed-size arrays in upper half **i** branching nodes only
-  **PINGO** query time?

Query Time (Contains)

- upper half: $O(m)$
- lower half: $O(m + \lg \sigma)$
- total: $O(m + \lg \sigma)$



Space

- upper half: $O(N)$ words
i $O(N/\sigma)$ branching nodes
- lower half: $O(N)$ words
- total: $O(N)$ words

Theoretical Comparison

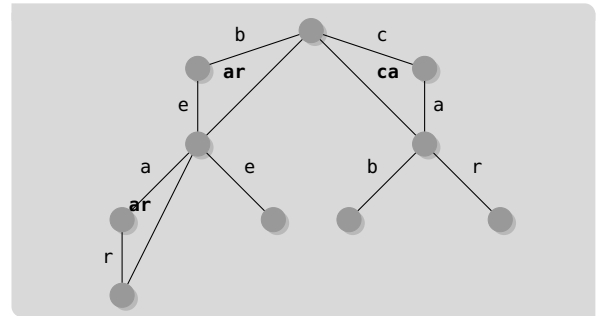
Representation	Query Time (Contains)	Space in Words
arrays of variable size	$O(m \cdot \sigma)$	$O(N)$
arrays of fixed size	$O(m)$	$O(N \cdot \sigma)$
hash tables	$O(m)$ w.h.p.	$O(N)$
balanced search trees	$O(m \cdot \lg \sigma)$	$O(N)$
weight-balanced search trees	$O(m + \lg k)$	$O(N)$
two-levels with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

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.



Conclusion and Outlook

This Lecture

- dictionaries
- tries with different space-time trade-off

Next Lecture

- suffix trees and suffix arrays
- no lecture on Halloween(!)
- next lecture 07.11.2022