Example for wild card handling in TRIES

Sabine Bergler

Example:

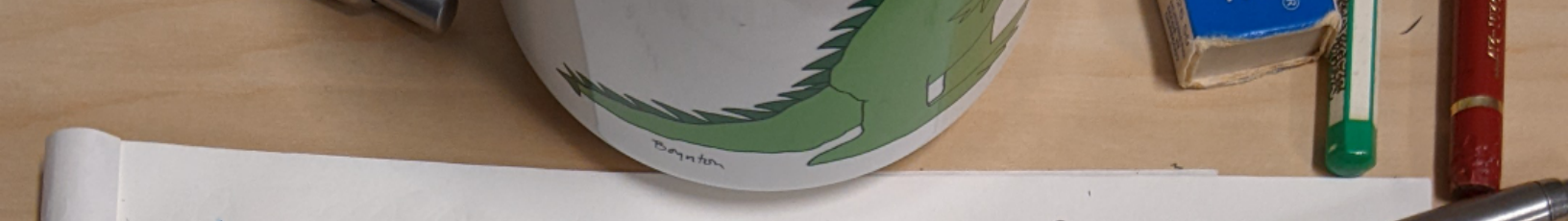
Terms and postings:

:

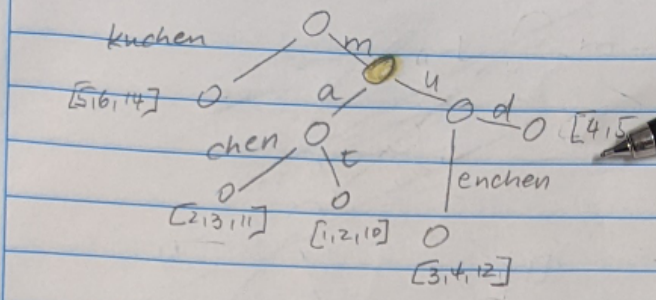
mat = [1,2,10]
machen = [2,3,11]
muenchen = [3,4,12,14]
mud = [4,5,13]
kuchen = [5,6,14]

Queries:

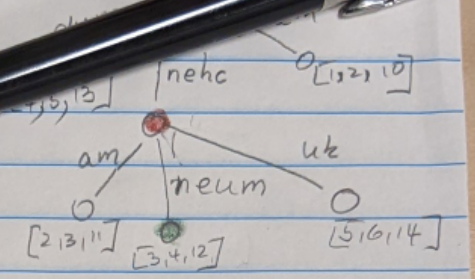
- m*chen
- m*nchen



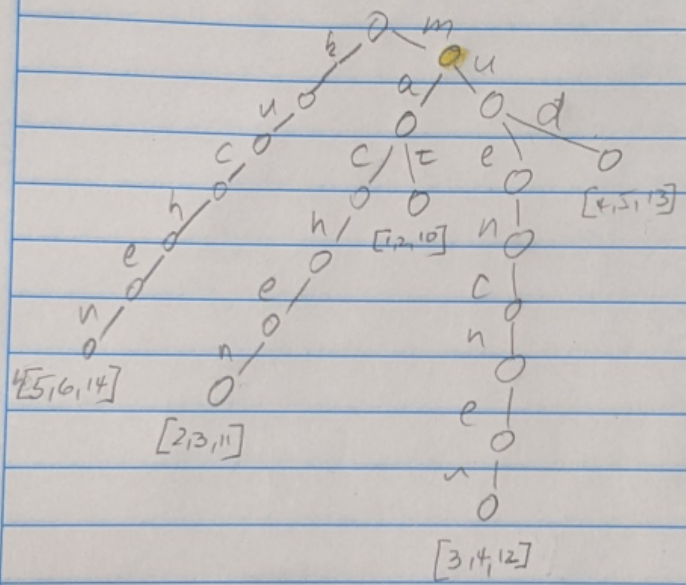
B tree



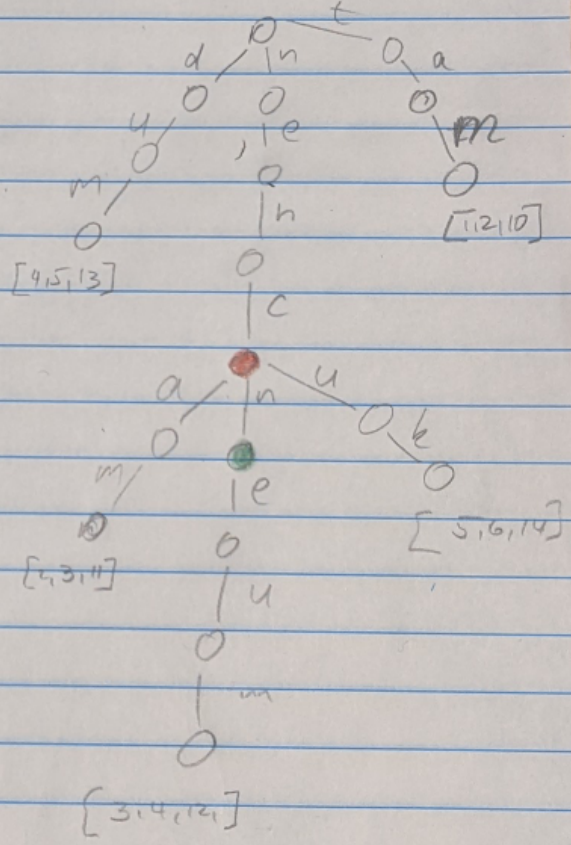
Backwards B-tree



TRIE



BACKWARDS TRIE



m^* = [1,2,3,4,5,10,11,12,13,14]

$*chen$ = [2,3,4,5,6,11,12,14]

$*uchen$ = [3,4,12]

$m^* \cap *chen$ = [2,3,4,5,11,12,14]

$m^* \cap *nchen$ = [3,4,12,]

m^* = { mat, machen, münchen, mud }

$*chen$ = { machen, münchen, kuchen }

$*nchen$ = { münchen }

$m^* \cap chen$ = { machen, münchen } => [2,3,4,11,12]

$m^* \cap nchen$ = { münchen } => [3,4,12]

*m**.postings ∩ **chen*.postings overgenerates

- When you intersect the postings lists for

*m** and **chen*, you get

[1,2,3,4,5,10,11,12,13,14] ∩ [2,3,4,5,6,11,12,14] = [2,3,4,5,11,12,14]

Where document 5 and 14 have *kuchen* and *mud*, but nothing that starts with *m* AND ends in *chen.*

Intersect strings

- Instead, intersect the term candidates, i.e. all the strings in the TRIE that start with *m*: {*mat,machen, muenchen, mud*} with all the strings in the backwards trie that end in *chen*: {*machen, muenchen, kuchen*}

- {*mat,machen, muenchen, mud*} ∩ {*machen, muenchen, kuchen*} = {*machen, muenchen*}

- The postings for {*machen, muenchen*} are the union of the postings for *machen* and for *muenchen*: [2,3,4,11,12]!

Lecture 8

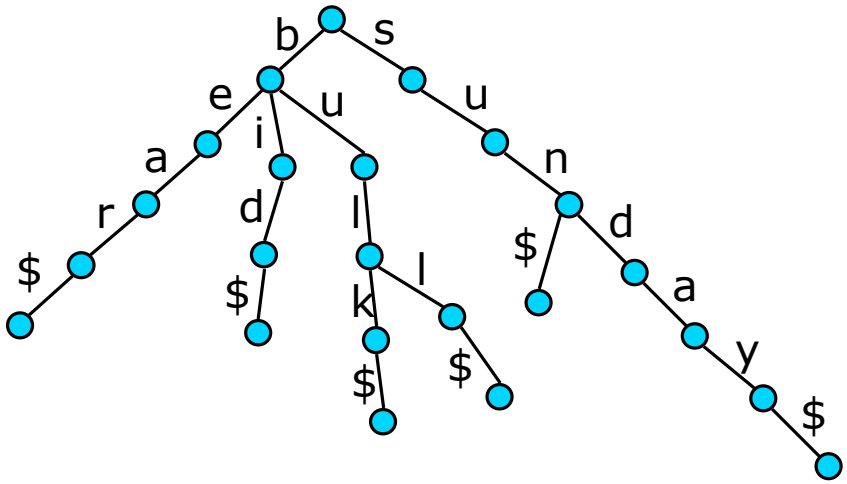
Text Indexing

Rasmus Pagh

in part based on slides by
S. Srinivasa Rao and Paolo Ferragina

Tries

- *Trie* (name from the word "re*trie*val"): a data structure for storing a set of strings
  - Let's assume that all strings end with "$" (not in Σ)



Set of strings: {**bear, bid, bulk, bull, sun, sunday**}

Tries

- Properties of a ***trie***:
  - A multi-way tree.
  - Each ***node*** has from 1 to Σ+1 children.
  - Each ***edge*** of the tree is labeled with a character.
  - Each ***leaf*** node corresponds to the stored string, which is a concatenation of characters on a path from the root to this node.

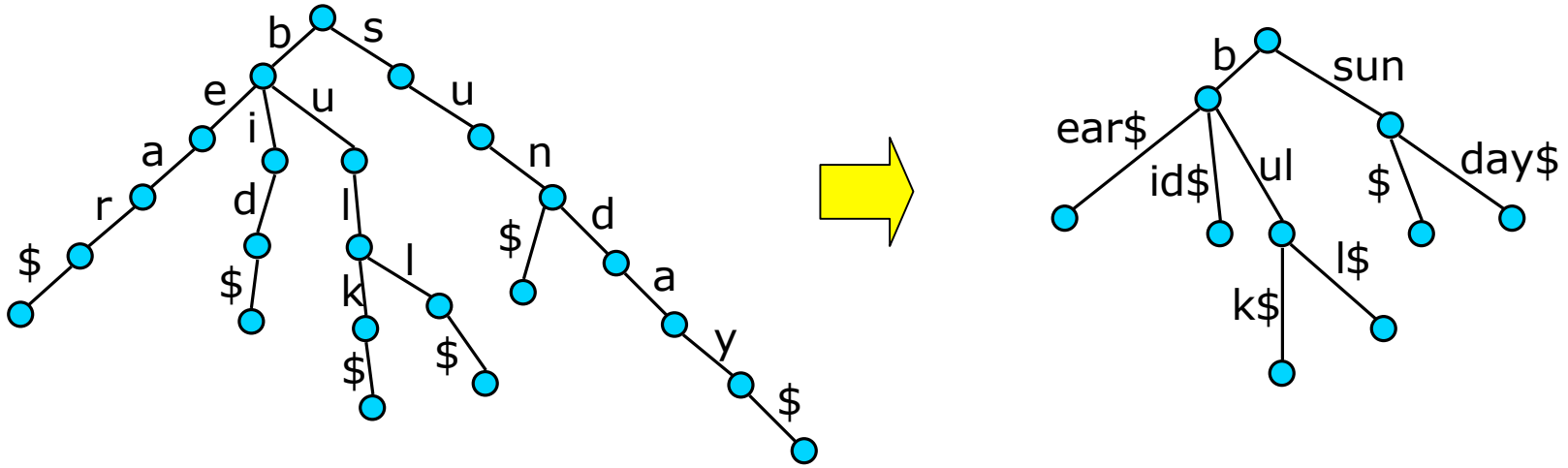
Analysis of the Trie

Given k strings of total length N:

- Size:
  - O(N) in the worst-case
- Search, insertion, and deletion (string of length *m*):
  - O(m) (assuming Σ is constant)
- Observation:
  - Having chains of one-child nodes is wasteful

Compact Tries

- *Compact Trie*:
  - Replace a *chain* of one-child nodes with an edge labeled with a string
  - Each non-leaf node (except root) has at least two children



Compact Tries

- Implementation:
  - Strings are external to the structure in one array, edges are labeled with indices in the array (*from*, *to*)
  - Improves the space to O(k)