2-INF-237 Vybrané partie z dátových štruktúr

2-INF-237 Selected Topics in Data Structures

- Instructor: Broňa Brejová
- E-mail: brejova@fmph.uniba.sk
- Office: M163
- Course webpage: http://compbio.fmph.uniba.sk/vyuka/vpds/

Full-text keyword search

Plnotextové vyhľadávanie kľúčových slov

Problem statement

Document: Sequence of words

Goal: Create an index for a static set of documents to answer the following queries efficiently.

Query: Given a word w, find all documents containing w.

Example:

Document 0: `Ema ma mamu.`

Document 1: `Mama ma Emu.`

Document 2: `Mama sa ma. Ema sa ma.`

Query `Mama` returns documents 1,2.

Full-text keyword search

Plnotextové vyhľadávanie kľúčových slov

Problem statement

Document: Sequence of words

Goal: Create an index for a static set of documents to answer the following queries efficiently.

Query: Given a word w, find all documents containing w.

Practical issues

Document: webpage/email/book/chapter/abstract/...

Preprocessing: lower/upper case, stemming (úprava na základný tvar), what is a word/word separator?, synonyms, ...

If there are many documents, how to rank them? (Information/text retrieval)

Preprocessing: divide into words, convert to lowercase, . . .

Document 0: ema, ma, mamu

Document 1: mama, ma, emu

Document 2: mama, sa, ma, ema, sa, ma

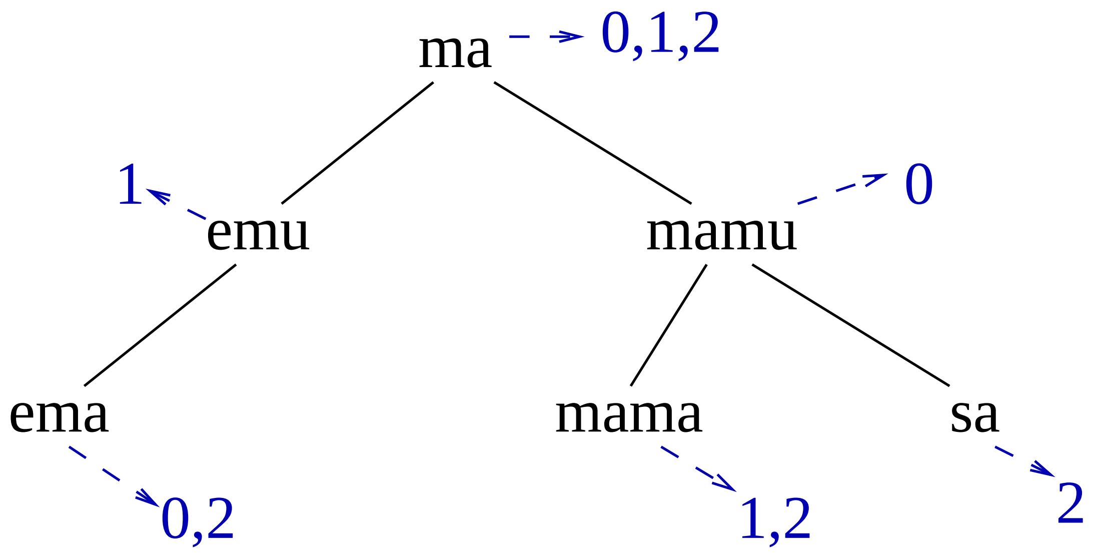
Inverted index: for each word a list of occurrences (document IDs)

ema:	0,2
emu:	1
ma:	0,1,2
mama:	1,2
mamu:	0
sa:	2

Implementing inverted index with balanced search trees

Balanced binary search tree, (e.g. red-black tree):

search, insert, delete using $O(\log n)$ comparisons



Trie (lexikografický strom)

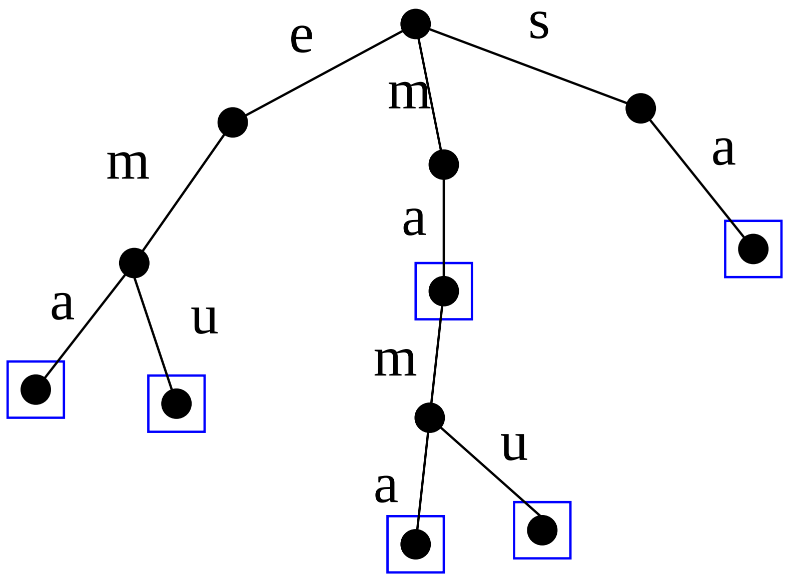
Represents a set of words

Edges labeled with characters

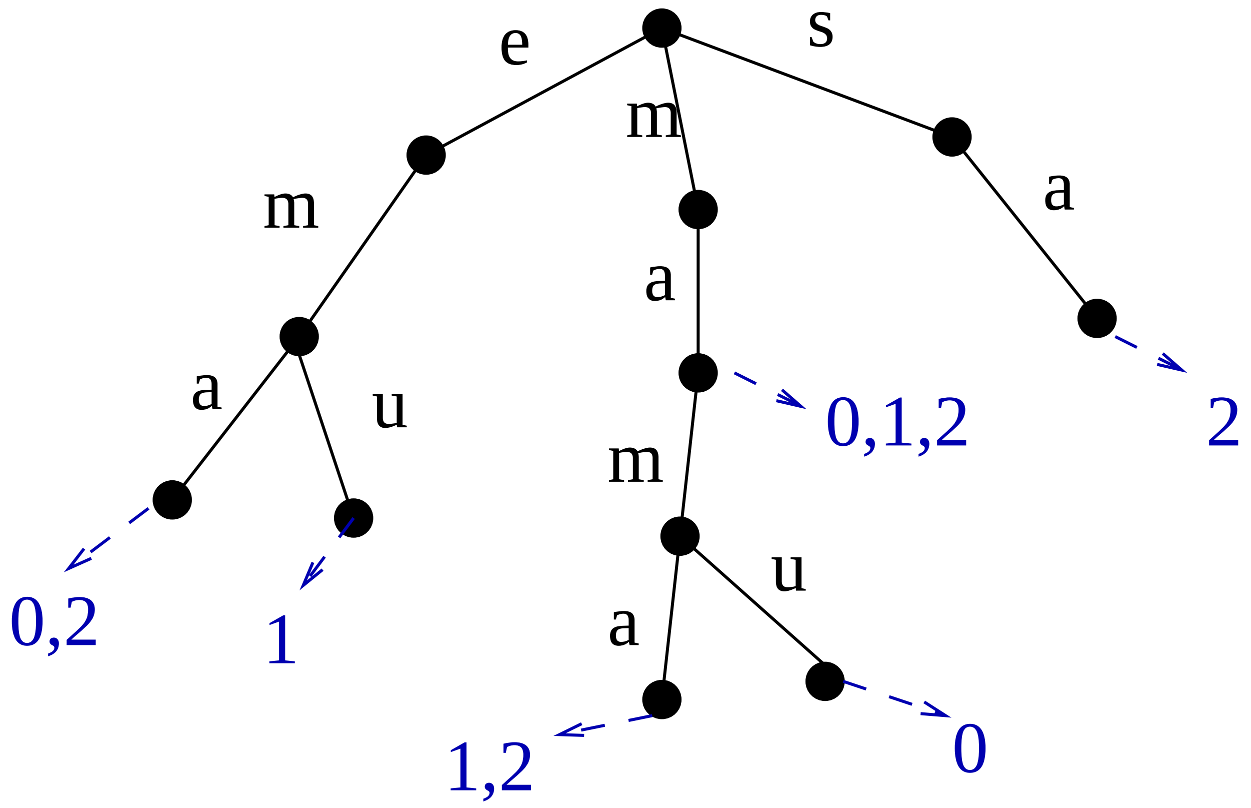
A node represents string read along the path from the root

Root represents empty string

In each node store flag if node in the set, plus other data

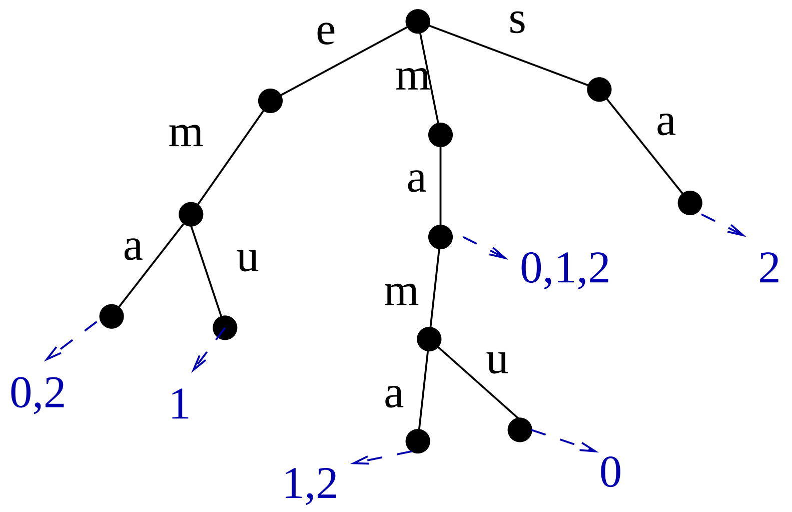


Inverted index implemented as a trie



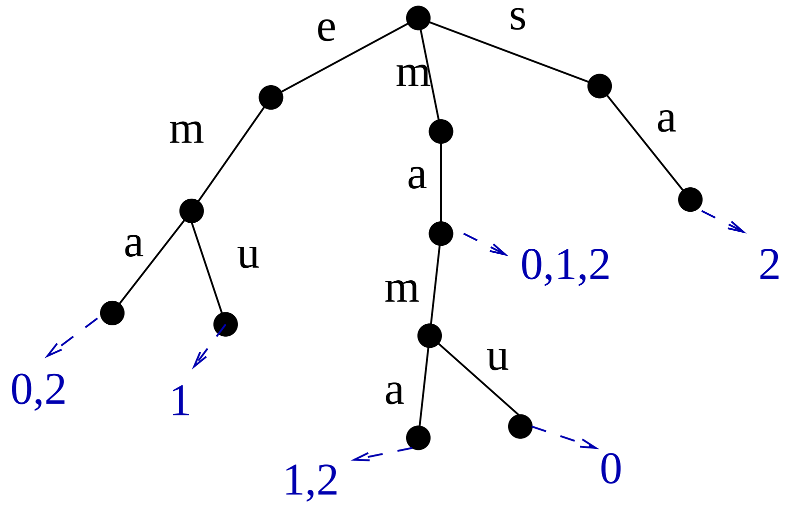
Searching for word w in a trie

```
node = root;
for(i=0; i<m; i++) {
    node = node->child[w[i]];
    if(! node) return empty_list;
}
return node->list;
```



Inserting word w from document d to a trie

```
node = root;
for(i=0; i<m; i++) {
    if(! node->child[w[i]]) {
        node->child[w[i]] = new node;
    }
    node = node->child[w[i]];
}
node->list.add(d)
```



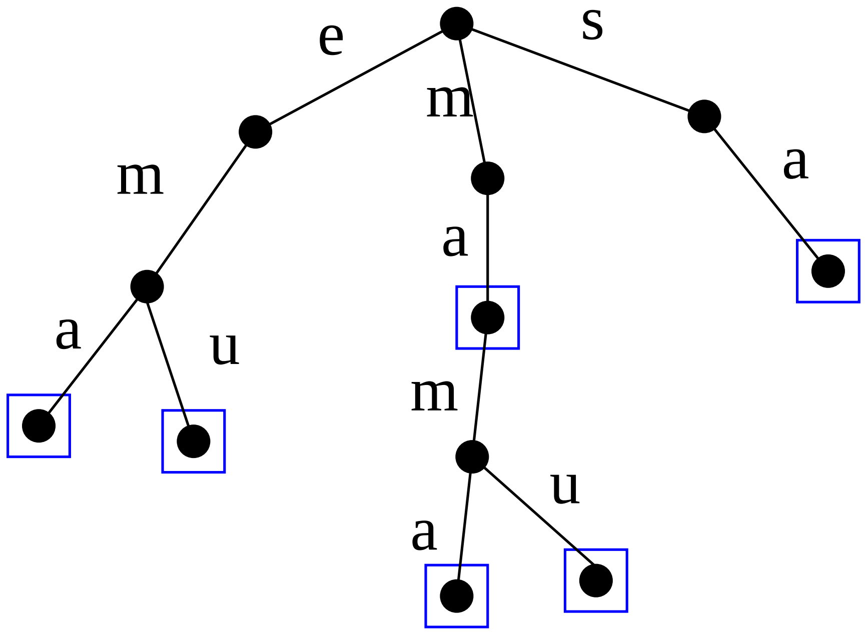
What about delete?

Trie

Assume word of length m, small alphabet

Insert, search, delete in $O(m)$ time if alphabet is small

How to store each node if alphabet large?



Trie

In each node: map from alphabet to pointers to children nodes

Implementation of this map for an alphabet of size σ:

	Search	Insert	Memory
Array of size σ	$O(m)$	$O(m\sigma)$	$O(D\sigma)$
Sorted array	$O(m \log \sigma)$	$O(m \log \sigma + \sigma)$	$O(D)$
Bin. search tree	$O(m \log \sigma)$	$O(m \log \sigma)$	$O(D)$

D – total length of all words

m – length of the word to be searched/inserted

σ – alphabet size

Implementations of inverted index

	Query	Preprocessing
Binary search tree (balanced)	$O(m \log n + p)$	$O(mN \log n)$
Hashing - expected/average case	$O(m + p)$	$O(mN)$
Trie	$O(m \log \sigma + p)$	$O(mN \log \sigma)$

m – max. length of a word

n – the number of distinct words

N – total number of words

σ – alphabet size

p – the number of documents found

Queries with multiple keywords

Searching with 2 keywords (connected by AND)

Intersection of two lists of occurrences

Assume input lists sorted (by some criterion)

Lengths of lists m and n ($m \leq n$)

Any ideas?

Queries with multiple keywords

Find intersection of two sorted arrays (lengths $m < n$)

- Linear-time merge $O(m + n)$
- m-times binary search $O(m \log n)$
- Doubling search $O(m \log \frac{n}{m})$

More than two arrays: add one by one, or use a different algorithm

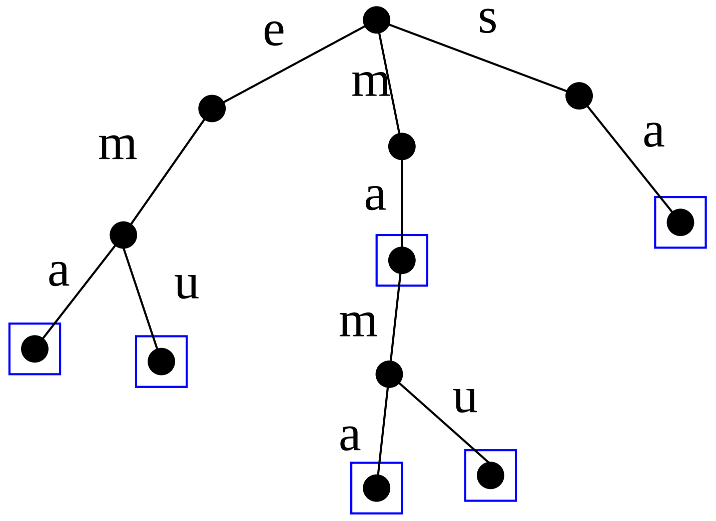
Also possibly preprocess sets for faster answers [Cohen, Porat 2010]

Applications of tries

Work with individual words:

- Keyword search
- Spell-checking
- Counting word frequencies

Also used in multiple pattern search (Aho-Corasick algorithm) and LZW compression


e
s
m
m
a
a
a
u
a
m
a
u