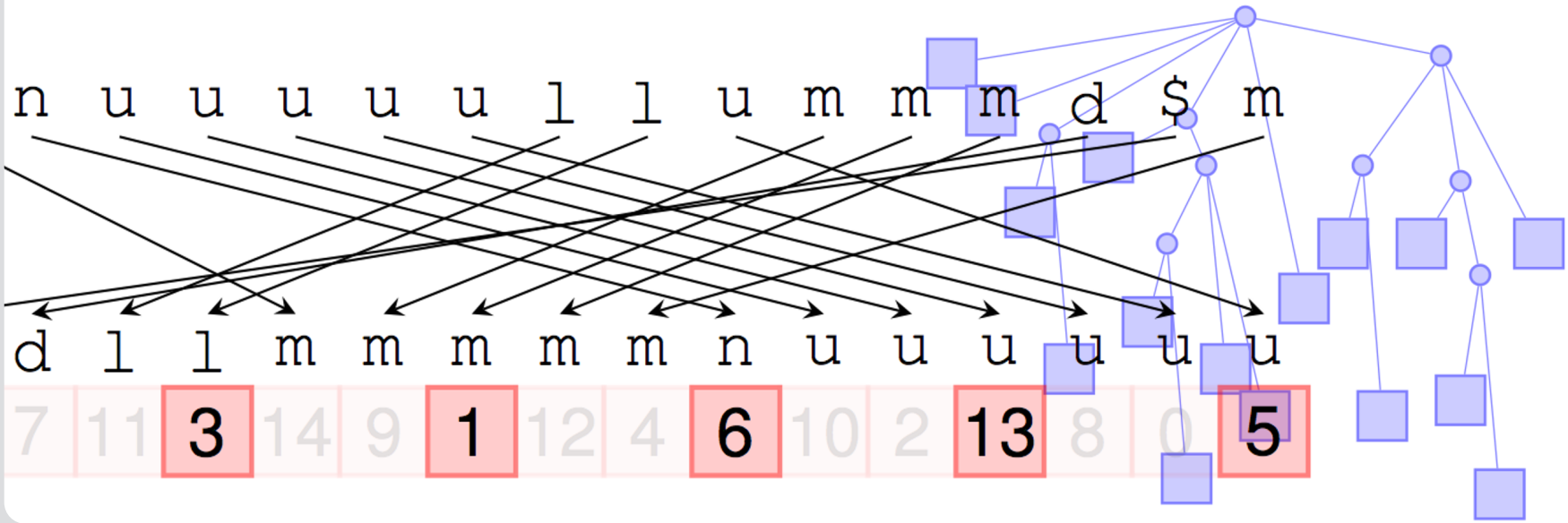
Text Indexing: Content

Simon Gog – *gog@kit.edu*

Institute of Theoretical Informatics - Algorithmics



Text Indexing

Motivation

Problem (String Matching)

Given a text T of length n, determine how many times a pattern P (of length m) occurs in T.

Sequential string match

E.g. use `fgrep` to search in a 200 MB XML file:

```
>time fgrep -c "<title>Indexing" dblp.xml.200MB
176

real    0m5.156s
user    0m5.113s
sys     0m0.042s
```

Text Indexing

Motivation

Sequential string match

- Best sequential string matching algorithms (e.g. Knuth-Morris-Pratt) solve problem in $\mathcal{O}(n+m)$ time.
- For multi-pattern searches there exists also efficient algorithms (e.g. Aho-Corasick). Time complexity $\mathcal{O}(n+M)$, where M is the sum of the length of all patterns.
- Drawback: Time complexity depends linearly on text size n.

Text Indexing

Motivation

Indexed String Matching

Pre-process 200 MB XML file:

```
>./index-daemon -f dblp.xml.200MB &
```

Query the constructed index:

```
>time echo "<title>Indexing" | ./index-client
176

real    0m0.006s
user    0m0.002s
sys     0m0.004s
```

0.006s vs. 5.156s of sequential solution.*

* Code available in the lecture branch of the SDSL repo.

Text Indexing

Motivation

Indexed String Matching

- Pre-compute data structure (*index*)
- Use index to answer the queries
- Time complexity still linearly dependent on m but not linearly dependent on n (or even independent).
- Another factor: alphabet size

Practical concerns

- Size of index (especially for in-memory indexes)
- Resource requirements for index construction

Text Indexing

Motivation

Beyond String Matching

Indexes can solve more complex queries efficiently:

- Find longest repeat (e.g. useful in LZ compression)
- Find suffix-prefix matches (e.g. useful in Bioinformatics)
- Ranked search (for a set of strings, i.e. a collection of documents)
  - Find documents which contain a patter most frequently
  - Find documents which contain a pattern and rank them according to their weight
  - ...

Text Indexing

Contents

- Classical Text Indexes
  - Suffix Trees
  - String-B-Trees
- Compressed Text Indexes
  - Inverted Indexes
  - Compressed Suffix Arrays (based on Wavelet Trees,)
  - Compressed Suffix Trees
  - Lempel-Ziv Indexes
- Search Engines
  - based on Inverted Indexes
  - based on Self-Indexes

Text Indexing

Succinct Data Structure Library

Compressed Text Indexes are composed of basic components which will also be covered in the lecture and are implemented in the SDSL library:

- (compressed) bitvectors
- rank and select data structures
- wavelet trees
- balanced parentheses structures
- range minimum query structures
- range search structures
- ...

Text Indexing

Succinct Data Structure Library

The SDSL library is a C++ template library (design close to STL) which

- implement ideas from about 50 research papers
- makes it easy to compose complex index structures
- is well optimized (e.g. exploits bit-parallelism and SSE instructions)
- contains benchmarks and unit tests
- will be used in the practical part of the lecture
- is available at `https://github.com/simongog/sdsl-lite`

Tutorial slides are available on the github website.

Text Indexing

Project for additional ECTS points

Implement a little project which uses text compressed text indexes. E.g. an effcient

- code search engine
- version control search engine

Text Indexing

Literature

- Full-Text Index Survey artircle of Navarro and Mäkinen [2]
- Wavelet Tree Survey by Navarro [1]
- and other recent articles

Literature

[1] Gonzalo Navarro. Wavelet trees for all. *J. Discrete Alg.*, 25:2–20, 2014.

[2] Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Comp. Surv.*, 39(1), 2007.