Succinct RegEx

Lightening-fast Regular Expressions

Anurag Khandelwal, Rachit Agarwal, Ion Stoica

Regular Expressions

- Powerful and expressive tool for data analysis.
- Wide range of applications: text and document databases, XML databases, data mining, deep packet inspection and bioinformatics
- Traditional Databases: Via the `LIKE` operator.
- 25% of TPC-H queries contain `LIKE` operator.

Existing Techniques

- Scan-based approaches (NFA, DFA)
  - Can be slow for large volumes of data
- n-gram indexes
  - Avoid scans using index entries for tokens of length $\geq n$
  - If token length smaller than n, resort to scans
- Tree-based indexes (Suffix-Trees)
  - Efficient indexing of arbitrary length tokens
  - Suffer from large memory footprint
- Compressed indexes (CSTs, CSAs, FM-Indexes)
  - Support memory-efficient lookups of arbitrary length tokens
  - E.g., Succinct enables search directly on compressed data
- Compressed indexes are fast and memory-efficient; can be used as a black-box (could be far from optimal)

Black-box approach to RegEx

- Break RegEx into tokens; search for individual tokens
- Combine intermediate results based on operators
- **Operators:** Union('|'), Concat ('.'), Repeat ('*', '+'), Wildcard (".*", ".+").

Query: "Ahoy (matey!|hearties!)" →
Search("Ahoy ") = {0, 17, 56, 94, 109, ...}
Search("matey!") = {5, 44, 99, 134, ...}
Search("hearties!") = {22, 63, 75, 165, ...}
Search("matey!|hearties!") = {5, 22, 44, 63, 75, 99, ...}
Final Result: {5, 22, 99, ...}

- **Repeat** and **Wildcard** operators computed similarly.
- Performance depends on #occurrences of tokens
- Inefficient if tokens occur too frequently.

Succinct RegEx

Main Idea: Series of optimizations using Succinct's performance benchmarking and internal data structures.

[1] Counting #occurrences of tokens orders of magnitude faster than corresponding searches in Succinct
- Plan order of execution of RegEx operators based on cardinality of intermediate results

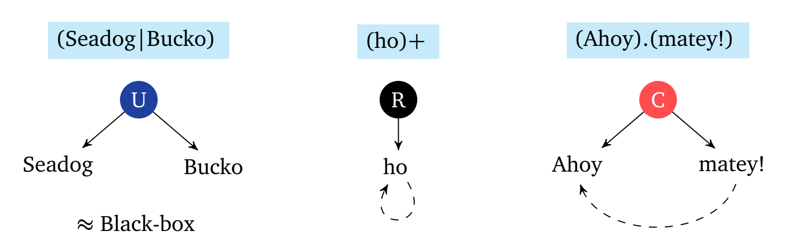
[2] Count time independent of input string length; Search time dominated by #occurrences of input string
- Concatenate tokens across operators (create longer tokens)
- Count time remains the same; Search time often reduces (longer tokens have fewer occurrences)

[3] Succinct query algorithm modified to incorporate operators without modifying internal data structures
- Get benefits of compressed data representation
- All the above optimizations when black-box approach is efficient (#occurrences of tokens small)
- When black-box approach inefficient, queries executed *across* RegEx operators directly on Succinct data structures

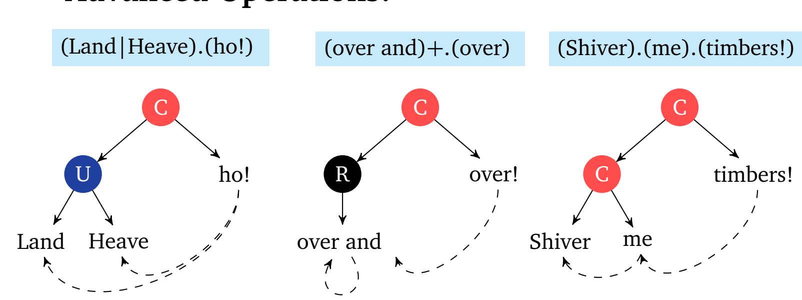
Executing RegEx *Directly* on Succinct

Main Idea: Continue search *across* RegEx operators

- **Basic Operations:**

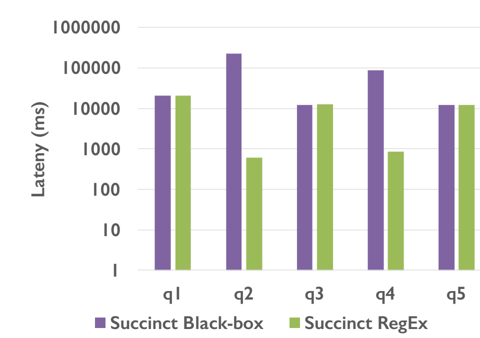


- **Advanced Operations:**



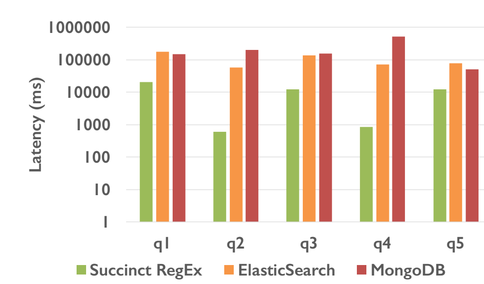
Preliminary Results

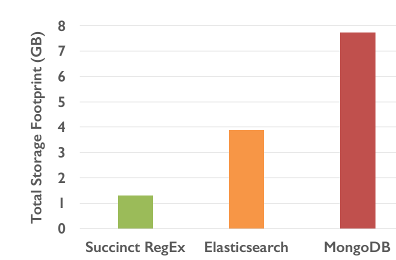
QID	RegEx
q1	.*(accounts).*
q2	.*((1993|1994)-02-01).*
q3	.*((123)+).*
q4	.*((1998-(01-01|02-01|03-01))+).*
q5	^(123)+



Succinct RegEx is at least as fast as Succinct Black-box; when optimizations help, $100 - 370\times$ speed up over Succinct Black-box approach

Succinct RegEx $6 - 94\times$ faster than existing systems on evaluated queries (data for all systems fits in memory).





Succinct RegEx achieves above gains in performance and functionality while using $3\times$ lower memory.

We could use your feedback:

- Current focus: Evaluation
- **RegEx workloads?**
- **Application-specific optimizations:** Many more optimizations can be done if set of RegEx queries constrained. Interesting?
- **More related work:** There has been tremendous amount of related work in database and networking community. What else should we compare against?

{anuragk, ragarwal, istoica}@eecs.berkeley.edu