Efficient RegEx Queries on Compressed Data

Anurag Khandelwal, Rachit Agarwal, Ion Stoica

Regular Expressions
- Example: `\d{1,6}` (US Phone Numbers)
- Wide range of applications: text and document stores, bioinformatics, data mining, etc.

Existing Techniques
- Full Data Scans (NFA, DFA)
  - Do not scale with data size
  - E.g., MongoDB, Oracle, MySQL, etc
- m-gram Indexes with Partial Data Scans
  - Index token of length m for multiple or all values of m
  - Avoid scans for indexed tokens.
  - Partial scans for tokens not indexed.
  - Suffer from large memory footprint
    - E.g., ElasticSearch
- Succinct (Search on compressed data)
  - Memory-efficient search for arbitrary length tokens
  - Asymptotic search complexity similar to m-gram indexes
  - Can be used as a black box

Black-box RegEx
- Represent RegEx as an RTree
  - Leaves are tokens; Interior nodes are operators
- Search for leaves
- Traverse the RTree bottom-up, combining intermediate results at each node

Succinct
Main Idea: Transform the RTree for the RegEx such that:
- the black-box approach can be avoided for the Concat operator altogether
- Union, Repeat and Wildcard operators are pushed up the RTree

Pull-Up Union Transformation:

\[
\begin{align*}
(\text{RE}_1, \text{RE}_2, \text{RE}_3) &\rightarrow (\text{RE}_1, (\text{RE}_2, \text{RE}_3)) \\
(\text{RE}_1, \text{RE}_2, \text{RE}_3) &\rightarrow ((\text{RE}_1, \text{RE}_2), \text{RE}_3)
\end{align*}
\]

Pull-Up Wildcard Transformation:

\[
\begin{align*}
(\text{RE}_1, (\text{RE}_2, \text{RE}_3)) &\rightarrow (\text{RE}_1, (\text{RE}_2, \text{RE}_3)) \\
((\text{RE}_1, \text{RE}_2), \text{RE}_3) &\rightarrow (\text{RE}_1, (\text{RE}_2, \text{RE}_3))
\end{align*}
\]

Pull-Out Repeat Transformation:
Replace Repeat operator by Unions of Concat:

\[
\text{RE}^* = (\text{RE}^1, \text{RE}^2, \text{RE}^3, \ldots, \text{RE}^k)
\]
k: smallest integer for which \(\text{RE}^{k+1}\) has 0 occurrences.
- Use heuristic to upper bound value of k
- k can be large when RegEx contains character classes
- Use partial scans beyond threshold

Pull-Out Concat Transformation:
- Find Concat nodes whose children are tokens (T₁, T₂)
- Replace with new token \(T₁T₂\)

Transformations incorporated within Succinct data structures.

Results
Succinct has 8x smaller storage footprint than uncompressed data structures

<table>
<thead>
<tr>
<th>Query ID</th>
<th>Query Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1</td>
<td>Elasticsearch</td>
</tr>
<tr>
<td>W2</td>
<td>MongoDB</td>
</tr>
<tr>
<td>W3</td>
<td>ScanProsite</td>
</tr>
<tr>
<td>W4</td>
<td>Succinct</td>
</tr>
<tr>
<td>W5</td>
<td>Wikipedia Dataset</td>
</tr>
<tr>
<td>P1</td>
<td>Elasticsearch</td>
</tr>
<tr>
<td>P2</td>
<td>MongoDB</td>
</tr>
<tr>
<td>P3</td>
<td>ScanProsite</td>
</tr>
<tr>
<td>P4</td>
<td>Succinct</td>
</tr>
<tr>
<td>P5</td>
<td>Planet A Dataset</td>
</tr>
</tbody>
</table>

Open Source Release
- https://github.com/amplab/succinct-cpp
- For questions & feedback, contact us at:
  {anuragk, ragerwal, istoica}@berkeley.edu