Jacobson’s Rank

Ben Langmead

Please sign guestbook (www.langmead-lab.org/teaching-materials) to tell me briefly how you are using the slides. For original Keynote files, email me (ben.langmead@gmail.com).
Jacobson's rank

Basic ideas:

When a bitvector is sparse enough, we can simply store answers for all 1-bits

When a bitvector is short enough, we can store all answers for all possible vectors and queries

Here we think in terms of $B \cdot \text{rank}_1$ queries, but $B \cdot \text{rank}_0$ queries are doable with same methods

Jacobson's rank

Split bitvector into **chunks** of $\log_2 n$ bits each

\[ T : \begin{array}{cccccccc} & & & & & & \cdots & \end{array} \]

\[ n \]

$\log_2 n \equiv (\log_2 n)^2$

I'll omit base-2 from logs from now on
Jacobson's rank

\[ \times \frac{n}{\log^2 n} \]
\[ \cdot \log^2 n \cdot \]

Store pre-calculated **cumulative rank** up to each chunk

\[ O \left( \log n \cdot \frac{n}{\log^2 n} \right) = O \left( \frac{n}{\log n} \right) = \tilde{o}(n) \]

bits to store

cum. rank

# chunks
So far, extra space is $\tilde{O}(n)$

Finding a rank can be decomposed:

(a) find what chunk it's in (division)

(b) look up cumulative rank

(c) find (relative) rank within chunk

(d) add (b) + (c)

TODO
Jacobson's rank

Say a chunk consists of $2 \log n$ sub-chunks, each of $1/2 \log n$ bits.
Jacobson's rank

Say each sub-chunk has a relative cumulative rank from beginning of chunk
Since chunk has $\log^2 n$ bits, a relative cum. rank needs $\log \log^2 n = O(\log \log n)$ bits.

$O\left(\frac{n}{\log n}\right)$ sub-chunks overall (across all chunks), for total of $O\left(n \cdot \log \log n / \log n\right)$ bits for relative cum. ranks.

$O\left(n \cdot \log \log n / \log n\right) = \tilde{o}(n)$
Jacobson's rank

$- \log^2 n -$  

$- \frac{1}{2} \log n -$  

$2 \log n -$

Extra space for cumulative ranks & relative cumulative ranks still $\tilde{O}(n)$; so far so good
Jacobson's rank

Finding a rank:
(a) find what chunk it's in (division)
(b) look up cumulative rank
(c) find rank within chunk
   (c.i) find what sub-chunk it's in
   (c.ii) look up relative cum. rank
   (c.iii) find rank within sub-chunk
(d) add (b) + (c.ii) + (c.iii)

$\log^2 n$

$1/2 \log n$

$2 \log n$

$- \log n$
Jacobson's rank

\[ \leftarrow \frac{1}{2} \log n \rightarrow \] Finding rank within a sub-chunk: two ways of thinking

**Way 1:** \( \frac{1}{2} \log n \) is \( \sim \) a machine word; use instructions like "population count" to find rank in \( O(1) \) time

**Way 2:** Lookup table

(Coming next)
Jacobson's rank

Say we naively store answers to all rank queries for all length-$x$ bitvectors. How many bits required?

$$2^x$$

possible bitvectors
Jacobson's rank

Say we naively store answers to all rank queries for all length-$x$ bitvectors. How many bits required?

$$2^x \cdot x$$

possible bitvectors possible offsets
Jacobson's rank

Say we naively store answers to all rank queries for all length-$x$ bitvectors. How many bits required?

$$2^x \cdot x \cdot \log x$$

possible possible answer
bitvectors offsets
Jacobson's rank

Say we naively store answers to all rank queries for all length-\(x\) bitvectors. How many bits required?

\[
2^x \cdot x \cdot \log x
\]

possible bitvectors possible offsets answer

Let \(x = 1/2 \log n\)

\[
\begin{align*}
O \left( 2^{1/2 \log n} \cdot 1/2 \log n \cdot \log 1/2 \log n \right) \\
= O \left( \sqrt{n \log n \log \log n} \right) = o(n)
\end{align*}
\]
Jacobson's rank

Finding a rank:

(a) find what chunk it's in (division)
(b) look up cumulative rank
(c) find rank within chunk
   (c.i) find what sub-chunk it's in
   (c.ii) look up relative cum. rank
   (c.iii) find rank within sub-chunk
(d) add (b) + (c.ii) + (c.iii)
## Bitvectors

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time</th>
<th>Space (bits)</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B \cdot \text{access}$</td>
<td>$O(1)$</td>
<td>$n$</td>
<td>Lookup</td>
</tr>
<tr>
<td>$B \cdot \text{rank}_1$</td>
<td>$O(1)$</td>
<td>$\tilde{o}(n)$</td>
<td>Jacobson</td>
</tr>
<tr>
<td>$B \cdot \text{select}_1$</td>
<td>$O(1)$</td>
<td>$\tilde{o}(n)$</td>
<td>?? 🦄 💜 🦄 ??</td>
</tr>
</tbody>
</table>