Bitvectors and RSA queries

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).
Bitvectors

Bitvectors are no stranger to us; Bloom filters!

Now we consider bitvectors where slots have *meaning*

*Navigating* between slots also meaningful
Bitvectors

Does this bitvector have a "meaning?"

What if its name was is_prime? 😊

How might we query it?

E.g. next-highest-prime

E.g. designing a 2-universal hash, we want smallest prime (leftmost 1) greater than some number
Bitvectors

What if the vector really was a Bloom filter?

Why might want to "navigate" it?

Say we are counting 1s to estimate cardinality

Might want to "jump" between 1s, ask how spaced out they are ($k^{th}$ minimum value)
Bitvectors

Might represent words in a document

<table>
<thead>
<tr>
<th></th>
<th>abinoam</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>abiogenesis</td>
</tr>
<tr>
<td>0</td>
<td>abiological</td>
</tr>
<tr>
<td>0</td>
<td>abiosis</td>
</tr>
<tr>
<td>0</td>
<td>abiotic</td>
</tr>
<tr>
<td>0</td>
<td>abiotically</td>
</tr>
<tr>
<td>0</td>
<td>abiotrophy</td>
</tr>
<tr>
<td>0</td>
<td>abirritate</td>
</tr>
<tr>
<td>0</td>
<td>abishag</td>
</tr>
<tr>
<td>0</td>
<td>abit</td>
</tr>
<tr>
<td>0</td>
<td>abitibi</td>
</tr>
<tr>
<td>0</td>
<td>abiu</td>
</tr>
<tr>
<td>1</td>
<td>abject</td>
</tr>
</tbody>
</table>
Bitvectors

Could be a "one-hot" encoding of string

Navigating bitvectors = navigating the occurrences of characters in the string

https://www.encodeproject.org/documents/5284a1d9-7eb6-44d9-8979-b90408332b12/@@download/attachment/Anshul_slides.pdf
**Bitvectors**

How do we navigate / query bitvectors?

Proposal: "RSA" (Rank, Select, Access)
Bitvectors

\[ B \text{ . access}(i) = B[i] \]

Conceptually trivial, but harder if we compress \( B \)

(more later)

Indexing starts at 0
Bitvectors

\[ B \cdot \text{rank}_1(i) = \sum_{j=0}^{i-1} B[j] \]

\[ B \cdot \text{rank}_0(i) = i - B \cdot \text{rank}_1(i) \]

Rank counts up to
\textit{but not including} offset \( i \)
Bitvectors

\[ B \cdot \text{select}_1(i) = \max \{ j \mid B \cdot \text{rank}_1(j) = i \} \]

\[ B \cdot \text{select}_0(i) = \max \{ j \mid B \cdot \text{rank}_0(j) = i \} \]
Bitvectors

\[ B \text{. access}(\ldots) \]
\[ B \text{. rank}(\ldots) \]
\[ B \text{. select}(\ldots) \]

Let \(|B| = n\) and let \(m\) equal the number of set bits.
Bitvectors

What does this do?

\[ B \cdot \text{select}_1(B \cdot \text{rank}_1(i) - 1) \]

\[ B \cdot \text{rank}_1(i) = \sum_{j=0}^{i-1} B[j] \]

\[ B \cdot \text{select}_1(i) = \max\{ j \mid B \cdot \text{rank}_1(j) = i \} \]

Gives offset of next-earliest set bit -- *predecessor*
Bitvectors

How to implement $B \cdot \text{rank}_1$ & $B \cdot \text{select}_1$?

Idea 0: linear scans over $B$

Can we be more efficient?
Bitvectors

Idea 1: Pre-calculate all answers

\[ B \cdot \text{rank}_1 \]

\[ B \cdot \text{select}_1 \]

\[ [0, n) \quad [0, m) \]

\[ S_1 \]

\[
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 \\
1 & 2 & 2 & 2 \\
0 & 3 & 3 & 3 \\
1 & 4 & 4 & 4 \\
0 & 4 & 4 & 4 \\
0 & 5 & 5 & 5 \\
\end{array}
\]
## Bitvectors

**Idea 1: Pre-calculate all answers**

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Space (bits)</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B . \text{access}$</td>
<td>$O(1)$</td>
<td>$n$</td>
<td>Lookup</td>
</tr>
<tr>
<td>$B . \text{select}_1$</td>
<td>$O(1)$</td>
<td>$O(m \log n)$</td>
<td>Pre-calculate $S_1$</td>
</tr>
<tr>
<td>$B . \text{rank}_1$</td>
<td>$O(1)$</td>
<td>$O(n \log m)$</td>
<td>Pre-calculate $R_1$</td>
</tr>
</tbody>
</table>
Bitvectors

Idea 2: Pre-calculate all answers for $B \cdot \text{select}_1$

- Discard $R_1$, keep only $S_1$

- Rank queries can be answered with binary search on $S_1$

$O(m \log n)$ bits. $B \cdot \text{rank}_1$ is $O(\log m)$ time.
## Bitvectors

### Idea 2: Pre-calculate all answers for $B \cdot \text{select}_1$

<table>
<thead>
<tr>
<th></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>$B \cdot \text{select}_1$</td>
<td>$O(1)$</td>
<td>$O(m \log n)$</td>
<td>Pre-calculate $S_1$</td>
</tr>
<tr>
<td>$B \cdot \text{rank}_1$</td>
<td>$O(\log m)$</td>
<td>$O(m \log n)$</td>
<td>Binary search on $S_1$</td>
</tr>
</tbody>
</table>
## Bitvectors

Coming soon:

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Space (bits)</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B . \text{access}$</td>
<td>$O(1)$</td>
<td>$n$</td>
<td>Lookup</td>
</tr>
<tr>
<td>$B . \text{select}_1$</td>
<td>$O(1)$</td>
<td>$	ilde{O}(n)$</td>
<td>?🧚🦄🧚?</td>
</tr>
<tr>
<td>$B . \text{rank}_1$</td>
<td>$O(1)$</td>
<td>$	ilde{O}(n)$</td>
<td>?🧚🦄🧚?</td>
</tr>
</tbody>
</table>