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

\[
\begin{align*}
h_1(x_1) & \rightarrow 1 \\
h_1(x_2) & \rightarrow 0 \\
h_1(x_3) & \rightarrow 1
\end{align*}
\]
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, asking how spaced out they are (like $k^{th}$ minimum value)
Bitvectors

Might represent set of all words in a document
Bitvectors

Could represent "one-hot" encoding of string

4 parallel bitvectors

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

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

Idea is easy; can be hard in practice if we compress \( B \)

We’ll index starting at 0

\[ B \]
\begin{array}{c}
0 \\
1 \\
1 \\
0 \\
1 \\
0 \\
0 \\
0 \\
1 \\
0 \\
1 \\
\end{array}

\( B . \text{access}(2) = 1 \)

\( B . \text{access}(7) = 0 \)
Bitvectors

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

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

Note that rank is counting 0s/1s up to \textit{but not including} offset \( i \)
Bitvectors

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

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

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

(Don't forget: rank counts up to \textit{but not including} given offset)
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?
\[ B \cdot \text{select}_1 (B \cdot \text{rank}_1(i) - 1) \]

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

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

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, m)\]

\([0, n)\]
Bitvectors

Idea 1: Pre-calculate all answers

\[
\begin{align*}
B &= \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \\
R_1 &= \begin{bmatrix} 0 & 0 & 1 & 2 & 2 & 3 & 3 & 4 & 4 & 4 & 4 & 4 & 5 \end{bmatrix}
\end{align*}
\]
Bitvectors

Idea 1: Pre-calculate all answers

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

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

\[ [0, n) \]

\[ [0, m) \]

\[ O(m \log n + n \log m) \text{ bits required} \]
## 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>$B \cdot \text{access}$</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>$O(n \log m)$</td>
<td>Pre-calculate $R_1$</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>
</tbody>
</table>
Bitvectors

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

$O(m \log n)$ 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>$B \cdot \text{rank}_1$</td>
<td>$O(\log m)$</td>
<td>$O(m \log n)$</td>
<td>Binary search on $S_1$</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>
</tbody>
</table>
## Bitvectors

Is this possible?

<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>$B . \text{rank}_1$</td>
<td>$O(1)$</td>
<td>$\tilde{o}(n)$</td>
<td>?? 🦄unicorn??</td>
</tr>
<tr>
<td>$B . \text{select}_1$</td>
<td>$O(1)$</td>
<td>$\tilde{o}(n)$</td>
<td>?? 🦄unicorn??</td>
</tr>
</tbody>
</table>