Suffix Arrays: maximum skipping

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).
Suffix array: querying

SA:

```
14 0 12 10 1 3 5 7 13 11 9 2 4 6 8
```

LCP = 2

Query:  a b a b a a

== < -- --

Pivot:  a b b a b a b $

data
Suffix array: querying

SA: 14 0 12 10 1 3 5 7 13 11 9 2 4 6 8

LCP = 4  abab$
LCP = 2  abbabab$

Query: a b a b a a
      ===== >

Pivot: a b a b a a

Pivot: a b a b a a$
Suffix array: querying

Min-LCP skipping uses what we learn about LCPs to skip character comparisons.

We can also precompute common prefixes between suffixes for even more skipping!
Suffix array

Terminology: length of common prefix between a query string and a suffix is an **LCP**

Length of common prefix between two suffixes of the same string is an **LCE**

We learn about **LCPs** during binary search; we can precompute **LCEs**
Suffix array

SA:

14 0 12 10 1 3 5 7 13 11 9 2 4 6 8

Query: ababa a

Query: abababa a

Pivot: abbbababa b$

LCP = 2
Suffix array

We can skip the first 2 character comparisons between query and new pivot!

What if we knew: LCE between the two pivots is 2
Suffix array

SA: 14 0 12 10 1 3 5 7 13 11 9 2 4 6 8

Query: abababa

Pivot: ababab$
Suffix array

**SA:**

```
 14  0  12  10  1  3  5  7  13  11  9  2  4  6  8
```

**Query:**

```
 a b a b a a
```

**Pivot:**

```
 a b a b $
```

**LCP = 4**
Suffix array

Skip first 4 character comparisons between query and new pivot!

What if we knew: LCE between the two pivots is 4
Suffix array

SA: [14, 0, 12, 10, 1, 3, 5, 7, 13, 11, 9, 2, 4, 6, 8]

LCP = 4

Query: ababab
Pivot: ababababababab$
Suffix array

**SA:**

```
14 0 12 10 1 3 5 7 13 11 9 2 4 6 8
```

**LCP = 4**

**Query:**  `a b a b a a`

```
====
```

**Pivot:**  `a b a b a b b a b a b b a b b a b b b`
Suffix array

If query was less than the previous pivot, it must be less than this pivot too; no character comparisons needed.
Suffix array

**No skipping**

<table>
<thead>
<tr>
<th>ababa $</th>
<th>== &lt; - - -</th>
<th>abbabab $</th>
</tr>
</thead>
<tbody>
<tr>
<td>ababa $</td>
<td>= = = &gt; -</td>
<td>abab $</td>
</tr>
<tr>
<td>ababa $</td>
<td>= = = = &lt;</td>
<td>ababababab $</td>
</tr>
<tr>
<td>ababa $</td>
<td>= = = = &lt;</td>
<td>abababababab $</td>
</tr>
</tbody>
</table>

**Min-LCP skipping**

<table>
<thead>
<tr>
<th>ababa $</th>
<th>== &lt; - - -</th>
<th>abbabab $</th>
</tr>
</thead>
<tbody>
<tr>
<td>ababa $</td>
<td>= = = &gt; -</td>
<td>abab $</td>
</tr>
<tr>
<td>ababa $</td>
<td>= = = = &lt;</td>
<td>ababababab $</td>
</tr>
<tr>
<td>ababa $</td>
<td>= = = = &lt;</td>
<td>abababababab $</td>
</tr>
</tbody>
</table>
### Suffix array

#### Min-LCP skipping

<table>
<thead>
<tr>
<th>ASCII</th>
<th>Min-LCP Skipping</th>
</tr>
</thead>
<tbody>
<tr>
<td>ababa</td>
<td>== &lt; - - -</td>
</tr>
<tr>
<td>ababb</td>
<td>==</td>
</tr>
<tr>
<td>ababa</td>
<td>=== &gt; -</td>
</tr>
<tr>
<td>abab</td>
<td>==</td>
</tr>
</tbody>
</table>

#### Max skipping

<table>
<thead>
<tr>
<th>ASCII</th>
<th>Max skipping</th>
</tr>
</thead>
<tbody>
<tr>
<td>ababa</td>
<td>== &lt; - - -</td>
</tr>
<tr>
<td>ababb</td>
<td>==</td>
</tr>
<tr>
<td>ababa</td>
<td>=== &gt; -</td>
</tr>
<tr>
<td>abab</td>
<td>==</td>
</tr>
</tbody>
</table>

The suffix array is a fundamental data structure used in string processing, particularly in text indexing and searching algorithms. The Min-LCP skipping uses the minimum LCP (Longest Common Prefix) to optimize the search, while Max skipping uses the maximum LCP for a similar purpose.
Suffix array: less-than case

Say we know the query is less than the previous pivot with some LCP. We also know the LCE between the pivots.

- **LCP < LCE**
  - Query must also be less than next pivot; recurse left
  - New LCP is same as old LCP
  - New LCE is between red & new pivots

- **LCP > LCE**
  - Query must be greater than next pivot; recurse right
  - New LCP is same as old LCP
  - New LCE is between blue & new pivots

- **LCP = LCE**
  - Skip first LCP characters, then continue comparisons, updating LCP and deciding recursion as usual.
  - New LCE will involve red pivot
Suffix array: greater-than case

Say we know the query is **greater** than the previous pivot with some LCP. We also know the LCE between the pivots.

- **LCP < LCE**
  - Query must also be **greater** than next pivot; recurse right
  - New LCP is same as old LCP
  - New LCE is between red & new pivots

- **LCP > LCE**
  - Query must be **less** than next pivot; recurse left
  - New LCP is same as old LCP
  - New LCE is between blue & new pivots

- **LCP = LCE**
  - Skip first LCP characters, then continue comparisons, updating LCP and deciding recursion as usual.
  - New LCE will involve red pivot
Suffix array: less-than case

Say we know the query is less than the previous pivot with some LCP. We also know the LCE between the pivots.

- **LCP < LCE**
  - Query must also be less than next pivot; recurse left
  - New LCP is same as old LCP
  - New LCE is between red & new pivots

- **LCP > LCE**
  - Query must be greater than next pivot; recurse right
  - New LCP is same as old LCP
  - New LCE is between blue & new pivots

- **LCP = LCE**
  - Skip first LCP characters, then continue comparisons, updating LCP and deciding recursion as usual.
  - New LCE will involve red pivot
Suffix array

We must precompute all possible LCEs that might be needed in the previous computation
Suffix array

Store pre-computed LCEs at nodes (pivots), conditioned on whether previous pivot was to the left or right
Store pre-computed LCEs at nodes (pivots), conditioned on whether previous pivot was to the left or right.

Approximately triples size of the data structure (!)
Suffix array

Like naive & min-LCP strategies, max skipping performs $O(\log m)$ bisections

Once a character of P matches it "stays matched."
Comparisons in a given round don't look back farther than the last mismatch.

Our query time bound therefore improves from $O(n \log m)$ to $O(n + \log m)$

Max skipping

```
ababa
==<----
ababab$
```

```
ababa
==<>-
abab$
```

```
ababa
====<
abababababab$
```

```
ababa
====<
abababababab$
```
Suffix array: summary

Naive binary search on suffix array takes $O(n \log m)$ time, in contrast to $O(n)$ for suffix tree query

Min-LCP skipping helps, using what we learned in previous rounds to skip character comparisons

Requires no additional space beyond SA $+ T$

Not $O(n + \log m)$, but efficient in practice

Max skipping additionally stores $\sim 2m$ pre-computed LCEs, tripling structure size, but improving time to $O(n + \log m)$
Suffix array: summary

Whether query takes $O(n \log m)$ or $O(n + \log m)$ time, there's no escaping the $\log m$; the price of binary search!