String Algorithms and Data Structures

Suffix Arrays

CS 199-225
Brad Solomon

March 20, 2023
Informal Early Feedback Reminder

Will be closing the IEF form and opening a voting form soon!
Exact pattern matching \textit{w/ indexing}

\begin{align*}
\text{Preprocess (index)} & \approx O(|T|) \\
\text{Search Index} & \approx O(|P|) \\
\text{Find instances of } P \text{ in } T
\end{align*}
Exact pattern matching \(\textbf{w/ indexing}\)

There are many data structures built on \textit{suffixes}

Modern methods still use these today
Suffix Trie

A rooted tree storing a collection of suffixes as (key, value) pairs

The tree is structured such that:

- Each key is “spelled out” along some path starting at root
- Each edge is labeled with a character $c \in \Sigma$
- For given node, at most one child edge has label $c$, for any $c \in \Sigma$
- Each key’s value is stored at a leaf
Suffix Tree

A rooted tree storing a collection of suffixes as (key, value) pairs

The tree has many similarities to the trie but:

- Each edge is labeled with *a string s*
- For given node, at most one child edge *starts with character c*, for any $c \in \Sigma$
- Each internal node contains $>1$ children

$T = \text{abaaba}$$\$
Searching a suffix tree

How efficient is search?

$T = ababa$

$P = ab$

\[ T \text{ = abaaba$} \]

\[ P \text{ = ab} \]

\[ T \rightarrow a, ba, aba, aba$ \]

\[ P \rightarrow a, ba \]

\[ T \rightarrow abaaba$ \]

\[ P \rightarrow ab \]

\[ T \rightarrow abaaba$ \]

\[ P \rightarrow ab \]

\[ T \rightarrow abaaba$ \]

\[ P \rightarrow ab \]

\[ T \rightarrow abaaba$ \]

\[ P \rightarrow ab \]

\[ T \rightarrow abaaba$ \]

\[ P \rightarrow ab \]
## Suffix trie vs suffix tree: bounds

<table>
<thead>
<tr>
<th></th>
<th>Suffix trie</th>
<th>Suffix tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time: Does P occur?</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Time: Report $k$ locations of P</td>
<td>$O(n + m^2)$</td>
<td>$O(n + k)$</td>
</tr>
<tr>
<td>Space</td>
<td>$O(m^2)$</td>
<td>$O(m)$</td>
</tr>
</tbody>
</table>

$m = |T|$,  $n = |P|$,  $k = \#$ occurrences of $P$ in $T$
Suffix trees in the real world

Plagiarism Scan Report

Characters: 30  Words: 10  Sentences: 20  Speak Time: 1 Min

To be or not to be that is the question

X = ABAB  Y = AAB  Generalized Suffix Tree: ABAB$AAB#
Suffix trees in the real world

Genome A: TCGATGCGAGGATCATTA
Genome B: AAGTCGCGAGGATCACCG
Suffix trees in the real world: MUMmer


~ 4,000 citations  

http://mummer.sourceforge.net
Suffix trees in the real world: MUMmer

File containing genome (T)

File containing query (P)

Columns:
1. Match index in T
2. Match index in P
3. Length of exact match

Example by Ben Langmead
Suffix trees in the real world: MUMmer

For whole chromosome 1, took 2m:14s and used 3.94 GB memory
Suffix trees in the real world: constant factor

Suffix Trees are $O(|T|)$ but there's a hidden constant factor at work:

MUMmer constant factor $\approx 15.76$ bytes per nt

Suffix tree of human genome: $>45$ GB

‘Raw’ two-bit encoding $\approx 2$ bits per nt

Raw encoding of human genome: $\sim 0.75$ GB
Exact pattern matching *w/ indexing*

There are many data structures built on *suffixes*

More efficient to store, less efficient* to use
Lexicographic Order

A systematic way of organizing strings by the content and arrangement of its characters
Lexicographic Order

A systematic way of organizing strings by the **content** and arrangement of its characters

Strings are compared by their individual characters.

**Alphabetical Order**  \( A < B < \ldots < Z \)

**ASCII Order**  \( $ < 0 < A < a \)

<table>
<thead>
<tr>
<th>ASCII Value</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>$</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>48</td>
<td>0</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>65</td>
<td>A</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>97</td>
<td>a</td>
</tr>
</tbody>
</table>
Lexicographic Order

A systematic way of organizing strings by the content and **arrangement** of its characters

Characters are compared in order from left to right

A B C D
A B A B
B B
B B B
Lexicographic Order

A systematic way of organizing strings by the content and arrangement of its characters

What is the lexicographically smallest string?

A) “beep”  B) “zzz”

C) “aardvarks”  D) “apples”
Lexicographic Order

A systematic way of organizing strings by the **content** and **arrangement** of its characters.

What is the *lexicographically* smallest string?

A) “bah$”  B) “x”

C) “bb$”  D) “b$b”
Suffix Array

Suffix array of $T$ is an array of integers specifying lexicographic (alphabetical) order of $T$’s suffixes

$T = a b a a b a$  
$0 1 2 3 4 5 6$
### Suffix Array

Suffix array of $T$ is an array of integers specifying lexicographic (alphabetical) order of $T$’s suffixes.

- Suffix Array: $T = \text{abaaba}$
  - $0$:
  - $1$:
  - $2$:
  - $3$:
  - $4$:
  - $5$:
  - $6$:

<table>
<thead>
<tr>
<th>Index</th>
<th>Suffix</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$#$</td>
</tr>
<tr>
<td>1</td>
<td>baaba</td>
</tr>
<tr>
<td>2</td>
<td>ababa</td>
</tr>
<tr>
<td>3</td>
<td>aba</td>
</tr>
<tr>
<td>4</td>
<td>a</td>
</tr>
<tr>
<td>5</td>
<td>$#$</td>
</tr>
<tr>
<td>6</td>
<td>$#$</td>
</tr>
</tbody>
</table>

As with suffix tree, $T$ is part of index.

$m$ integers

Note: Red is not stored.
vector<int> build_sarray(string T)

Input:

\[ T: \text{CGTGC} \]
\[ \text{m suffixes} \]

Output:
vector<int> build_sarray(string T)

**Input:**

```
T: C G T G C $
0 1 2 3 4 5
```

**Output:**

```
m suffixes
5
4
0
3
1
2
```
Suffix array: build by sorting (from array)

Use your favorite sort, e.g., quickSort, heapSort, insertSort, …

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a b a a b a $</td>
<td>b a a b a $</td>
<td>a a b a $</td>
<td>a b a $</td>
<td>b a $</td>
<td>a $</td>
<td>$</td>
</tr>
</tbody>
</table>

std::sort()

Expected time:

IntroSort

$O(m \log m)$
Suffix array: build by sorting *suffixes*

Another idea: Use a sort algorithm that’s aware that the items being sorted are all suffixes of the same string

Original suffix array paper suggested an $O(m \log m)$ algorithm


Other popular $O(m \log m)$ algorithms have been suggested


There exist several $O(m)$ algorithms that *divide-and-conquer*


Suffix array: build by suffix tree

(a) Build suffix tree, (b) traverse in lexicographic order, (c) upon reaching leaf, append suffix to array
Suffix array: build by suffix tree

(a) Build suffix tree, (b) traverse in lexicographic order, (c) upon reaching leaf, append suffix to array
Assignment 7: a_sarray

Learning Objective:

Construct a suffix array by sorting suffixes

Implement exact pattern matching using a suffix array

Be as efficient or inefficient as you like!

Challenge yourself: Try to build in $O(m^2 \log m)$ or better.
Searching a suffix array

To find all exact matches using a suffix array:

\[ T = \text{abaaba}$ \]
\[ P = \text{baa} \]

<table>
<thead>
<tr>
<th>Starts with b?</th>
<th>6</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starts with b?</td>
<td>5</td>
<td>a $</td>
</tr>
<tr>
<td>Starts with b?</td>
<td>2</td>
<td>a a b a $</td>
</tr>
<tr>
<td>Starts with b?</td>
<td>3</td>
<td>a b a $</td>
</tr>
<tr>
<td>Starts with b?</td>
<td>0</td>
<td>a b a a b a $</td>
</tr>
<tr>
<td>Starts with b?</td>
<td>4</td>
<td>b a $</td>
</tr>
<tr>
<td>Starts with b?</td>
<td>1</td>
<td>b a a b a $</td>
</tr>
</tbody>
</table>
Searching a suffix array

To find all exact matches using a suffix array:

\[ T = \text{abaaba}\$
\]

\[ P = \text{baa}\$

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6</td>
<td>$</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>a $</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>a a b a $</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>a b a $</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>a b a a b a $</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>b a $</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>b a a b a $</td>
</tr>
</tbody>
</table>

Matches \text{baa}? → 4

Matches \text{baba}? → 1
Searching a suffix array

To find all exact matches using a suffix array:

1. Recreate suffix from int value

2. Compare each character in order

3. On mismatch, move to next suffix

What is our time complexity?

```
T = abaaba$
m = |T|
P = baa
n = |P|
```

```
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>$</td>
</tr>
<tr>
<td>5</td>
<td>a $</td>
</tr>
<tr>
<td>2</td>
<td>a a b a $</td>
</tr>
<tr>
<td>3</td>
<td>a b a $</td>
</tr>
<tr>
<td>0</td>
<td>a b a a b a $</td>
</tr>
<tr>
<td>4</td>
<td>b a $</td>
</tr>
<tr>
<td>1</td>
<td>b a a b a $</td>
</tr>
</tbody>
</table>
```

Return {1}
Searching a suffix array

To find all exact matches using a suffix array w/ binary search:

\[ T = \text{abaaba}\$ \quad m = |T| \]
\[ P = \text{baa} \quad n = |P| \]

Match here? $ \rightarrow $

<table>
<thead>
<tr>
<th>Offset</th>
<th>Suffix</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>$</td>
</tr>
<tr>
<td>5</td>
<td>a $</td>
</tr>
<tr>
<td>2</td>
<td>a a b a $</td>
</tr>
<tr>
<td>3</td>
<td>a b a $</td>
</tr>
<tr>
<td>0</td>
<td>a b a a b a a $</td>
</tr>
<tr>
<td>4</td>
<td>b a $</td>
</tr>
<tr>
<td>1</td>
<td>b a a b a a $</td>
</tr>
</tbody>
</table>

Return \{1\}
Searching a suffix array

To find all exact matches using a suffix array w/ binary search:

\[ T = \text{abaaba}$ \quad m = |T| \]
\[ P = \text{aba} \quad n = |P| \]

Binary search match!

Return \{3\}!

But what about our other match?
Searching a suffix array

To find all exact matches using a suffix array w/ binary search:

1. Pick suffixes using binary search
2. Compare suffixes as normal
3. After match, check neighbors

Assume we have $k=m$ matches

What is our time complexity?

$T = \text{abaaba}\$
$m = |T|$
$P = \text{aba}$
$n = |P|$

Return $\{0,3\}$
Searching a suffix array

How can we do better?

\[ T = \text{abaaba}\$ \quad m = |T| \]
\[ P = a \quad n = |P| \]

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$6$</td>
<td>$$$</td>
</tr>
<tr>
<td>$5$</td>
<td>$a $$</td>
</tr>
<tr>
<td>$2$</td>
<td>$a \ a \ b \ a $$</td>
</tr>
<tr>
<td>$3$</td>
<td>$a \ b \ a $</td>
</tr>
<tr>
<td>$0$</td>
<td>$a \ b \ a \ a \ b \ a $</td>
</tr>
<tr>
<td>$4$</td>
<td>$b \ a $</td>
</tr>
<tr>
<td>$1$</td>
<td>$b \ a \ a \ b \ a $</td>
</tr>
</tbody>
</table>
Range Search

Given a collection of objects, $C$, with comparable values and an object of interest, $q$, find the first instance(s) of $q \in C$.

Output: Range of indices matching $q$ if it exists, $(-1, -1)$ otherwise
if mid == q:
    # Match case:
    # Treat like query is larger
    # Remember last match!

elif mid > q:
    # query is smaller case
else:
    # query is larger case

# Final Return Snippet
if saw_match:
    return last_match
else:
    return -1
if mid == q:
    # Match case:
    # Treat like query is smaller
    # Remember last match!

elif mid > q:
    # query is smaller case
else:
    # query is larger case

# Final Return Snippet
if saw_match:
    return last_match
else:
    return -1
Searching a suffix array

How can we do better?

1. Identify the *first* and *last* matches to $P$ w/ binary search
2. Return all values in that range!

Assume we have $k=m$ matches

What is our time complexity?

$T = \text{abaaba}\$

$P = a$

$m = |T|$

$n = |P|$

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a b a a a b a $</td>
<td>Last</td>
</tr>
<tr>
<td>3</td>
<td>a b a $</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>a a b a $</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>a $</td>
<td>First</td>
</tr>
<tr>
<td>6</td>
<td>$</td>
<td></td>
</tr>
</tbody>
</table>
Assignment 7: a_sarray

Learning Objective:

Construct a suffix array by sorting suffixes

Implement exact pattern matching using a suffix array

Be as efficient or inefficient as you like!

Challenge yourself: Try to search in $O(n \log m + k)$
Suffix tree vs suffix array: size

$O(m)$ space, like suffix tree

Is “constant factor” worse, better, same?
32-bit integers sufficient for human genome, so fits in
~4 bytes/base \( \times \) 3 billion bases \( \approx 12 \text{ GB} \). Suffix tree is >45 GB.
Suffix tree vs suffix array: search

<table>
<thead>
<tr>
<th>Suffix Array</th>
<th>Suffix Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(n \log m + k)^*$</td>
<td>$O(n + k)$</td>
</tr>
<tr>
<td>6 $$</td>
<td>6</td>
</tr>
<tr>
<td>5 a $</td>
<td>a ba$</td>
</tr>
<tr>
<td>2 a a b a $</td>
<td>a b a$</td>
</tr>
<tr>
<td>3 a b a $</td>
<td>a b a a b a $</td>
</tr>
<tr>
<td>0 a b a a b a $</td>
<td>b a $</td>
</tr>
<tr>
<td>4 b a $</td>
<td>b a a b a $</td>
</tr>
<tr>
<td>1 b a a b a $</td>
<td></td>
</tr>
</tbody>
</table>

* Can be improved to $O(n + \log m)$, (See Gusfield 7.17.4)
Suffix *arrays* in the real world: MUMmer


~ 4,000 citations

http://mummer.sourceforge.net
Exact pattern matching \textit{w/ indexing}

There are many data structures built on \textit{suffixes}

The FM index is a compressed self-index (smaller* than original text)!