Pattern Matching part one

Suffix Trees

Benjamin Sach
Exact pattern matching

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

\[
\begin{array}{c}
T \\
 a \ b \ c \ b \ \underbrace{a \ b \ a \ b \ a}_{m} \ c \ a \ b \ a \\
\end{array}
\]

\[
\begin{array}{c}
P \\
 \underbrace{a \ b \ a}_{m} \\
\end{array}
\]

**Goal:** Find all the locations where $P$ matches in $T$

$P$ matches at location $i$ iff

for all $0 \leq j \leq m$ we have that $P[j] = T[i + j]$

(our strings are zero-indexed)
Exact pattern matching

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

\[
\begin{array}{c}
T & a & b & c & b & a & b & a & b & a & c & a & b & a \\
\hline
P & a & b & a & a & b & a & \checkmark
\end{array}
\]

**Goal:** Find all the locations where $P$ matches in $T$

$P$ matches at location $i$ iff

for all $0 \leq j \leq m$ we have that $P[j] = T[i + j]$

*(our strings are zero-indexed)*
**Exact pattern matching**

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

\[
\begin{array}{c}
| T & a & b & c & b & a & b & a & b & a & c & a & b & a \\
| P & 6 & a & b & a & \checkmark \\
\end{array}
\]

**Goal:** Find all the locations where $P$ matches in $T$

$P$ matches at location $i$ iff

for all $0 \leq j \leq m$ we have that $P[j] = T[i + j]$

*(our strings are zero-indexed)*
Exact pattern matching

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

<table>
<thead>
<tr>
<th>$T$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$b$</th>
<th>$a$</th>
<th>$b$</th>
<th>$a$</th>
<th>$b$</th>
<th>$a$</th>
<th>$c$</th>
<th>$a$</th>
<th>$b$</th>
<th>$a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
</tr>
</tbody>
</table>

**Goal:** Find all the locations where $P$ matches in $T$

$P$ matches at location $i$ iff

for all $0 \leq j \leq m$ we have that $P[j] = T[i + j]$

*(our strings are zero-indexed)*
Exact pattern matching

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

\[
\begin{array}{c}
T \\
\end{array}
\begin{array}{c}
a \ b \ c \ b \ a \ b \ a \ b \ a \ c \ a \ b \ a \\
\end{array}
\]

\[
\begin{array}{c}
P \\
\end{array}
\begin{array}{c}
6 \ a \ b \ a \\
m \end{array}
\]

**Goal:** Find all the locations where $P$ matches in $T$

$P$ matches at location $i$ iff for all $0 \leq j \leq m$ we have that $P[j] = T[i + j]$

*(our strings are zero-indexed)*
Exact pattern matching

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

$$
\begin{array}{c}
T \quad a \quad b \quad c \quad b \quad a \quad b \quad a \quad b \quad a \quad c \quad a \quad b \quad a \\
P \quad 6 \quad a \quad b \quad a \\
\end{array}
$$

**Goal:** Find all the locations where $P$ matches in $T$

$P$ matches at location $i$ iff for all $0 \leq j \leq m$ we have that $P[j] = T[i + j]$

(our strings are zero-indexed)

- A naive algorithm takes $O(nm)$ time
Exact pattern matching

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

- $T = a b c b a b a b a c a b a$
- $P = a b a$

**Goal:** Find all the locations where $P$ matches in $T$

$P$ matches at location $i$ iff

for all $0 \leq j \leq m$ we have that $P[j] = T[i + j]$

(our strings are zero-indexed)

- A naive algorithm takes $O(nm)$ time
- Many $O(n)$ time algorithms are known (for example KMP)
Text indexing

Preprocess a text string $T$ (length $n$) to answer pattern matching queries...
Text indexing

Preprocess a text string $T$ (length $n$) to answer pattern matching queries...

\[ T \quad \overbrace{a \ b \ c \ b \ a \ b \ a \ b \ a \ c \ a \ b \ a}^{n} \]

After preprocessing, a query is a pattern $P$ (length $m$),

\[ P \quad \overbrace{a \ b \ a}^{m} \]
Text indexing

Preprocess a text string $T$ (length $n$) to answer pattern matching queries...

![Diagram](image)

After preprocessing, a **query** is a pattern $P$ (length $m$), the output is a list of all matches in $T$.

![Diagram](image)
Preprocess a text string $T$ (length $n$) to answer pattern matching queries.


diagram

$T$

\[
\begin{array}{cccccccc}
  a & b & c & b & a & b & a & b & a & c & a & b & a \\
  \hline
  4 & 6 & 10
\end{array}
\]

diagram

After preprocessing, a **query** is a pattern $P$ (length $m$),

\[
\begin{array}{ccc}
  a & b & a \\
  \hline
  m
\end{array}
\]

diagram

the output is a list of all matches in $T$.

diagram

e.g. 4, 6, 10
Text indexing

Preprocess a text string $T$ (length $n$) to answer pattern matching queries...

$$T = \text{a b c b a b a b a c a b a}$$

After preprocessing, a **query** is a pattern $P$ (length $m$),

$$P = \text{a b a}$$

the output is a list of all matches in $T$. e.g. 4, 6, 10

- A naive algorithm takes $O(n)$ query time (using KMP)
Text indexing

Preprocess a text string $T$ (length $n$) to answer pattern matching queries...

$$T = \begin{array}{cccccccc}
  a & b & c & b & a & b & a & b & a & c & a & b & a \\
  4 & 6 & 10
\end{array}
$$

After preprocessing, a **query** is a pattern $P$ (length $m$),

$$P = \begin{array}{ccc}
  a & b & a \\
  m
\end{array}$$

the output is a list of all matches in $T$.

- A naive algorithm takes $O(n)$ query time (using KMP)
- We want a query time which depends only on $m$ and $\text{occ}$
  - $\text{occ}$ is the number of occurences (matches)
Text indexing

Preprocess a text string $T$ (length $n$) to answer pattern matching queries...

After preprocessing, a **query** is a pattern $P$ (length $m$),

the output is a list of all matches in $T$.

- A naive algorithm takes $O(n)$ query time (using KMP)
- We want a query time which depends only on $m$ and $occ$
  - $occ$ is the number of occurrences (matches)
- We also want $O(n)$ space and fast preprocessing (prep.) time
The atomic suffix tree

\[ T \]

\[ \begin{array}{c}
| b | a | n | a | n | a | s \\
\hline
| \underline{n} | n \end{array} \]
The atomic suffix tree

$T$

suffixes

b\textcolor{red}{a}n\textcolor{red}{a}n\textcolor{red}{a}\textcolor{red}{n}\textcolor{red}{a}s

b\textcolor{red}{a}n\textcolor{red}{a}n\textcolor{red}{a}\textcolor{red}{n}\textcolor{red}{a}s

\textcolor{red}{a}n\textcolor{red}{a}n\textcolor{red}{a}\textcolor{red}{n}\textcolor{red}{a}s

\textcolor{red}{n}a\textcolor{red}{n}\textcolor{red}{a}\textcolor{red}{n}\textcolor{red}{a}s

\textcolor{red}{a}\textcolor{red}{n}\textcolor{red}{a}\textcolor{red}{n}\textcolor{red}{a}s

\textcolor{red}{n}\textcolor{red}{a}\textcolor{red}{n}\textcolor{red}{a}s

\textcolor{red}{n}\textcolor{red}{a}s

\textcolor{red}{a}s

s
The atomic suffix tree
The atomic suffix tree

T

```
bananas
```

suffix tree

```
bananas

banana

ana

na

an

a

n

s

a

a

s

b

n

s
```

suffixes

```
a

a

s

as

nas

anana

bananas
```

```
```
The atomic suffix tree

$T = \text{banana}nas$

suffix tree

suffixes
The atomic suffix tree
The atomic suffix tree

\[ T = \text{banana} \]

<table>
<thead>
<tr>
<th>T</th>
<th>b a n a n a s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a n a n a s</td>
</tr>
<tr>
<td></td>
<td>n a n a s</td>
</tr>
<tr>
<td></td>
<td>a n a s</td>
</tr>
<tr>
<td></td>
<td>n a s</td>
</tr>
<tr>
<td></td>
<td>a s</td>
</tr>
<tr>
<td></td>
<td>s</td>
</tr>
</tbody>
</table>

suffix tree

- **suffixes**
The atomic suffix tree

Tuple $T = \text{bananas}$

Suffixes:
- $\text{bananas}$
- $\text{ananas}$
- $\text{nanas}$
- $\text{anas}$
- $\text{nas}$
- $\text{as}$
- $\text{s}$
The atomic suffix tree
The atomic suffix tree

\[ T \]

\[ b a n a n a s \]

\[ b a n a n a s \]

\[ a n a n a s \]

\[ n a n a s \]

\[ a n a s \]

\[ n a s \]

\[ a s \]

\[ s \]
The atomic suffix tree
The atomic suffix tree

T

\[ \text{bananas} \]

\[ \text{banana\,banana\,banana\,banana\,banana} \]

\[ \text{banana\,banana\,banana\,banana} \]

\[ \text{banana\,banana\,banana} \]

\[ \text{banana\,banana} \]

\[ \text{banana} \]

\[ \text{ban} \]

\[ \text{b} \]

\[ \text{s} \]

\[ \text{suffixes} \]

\[ \text{suffix tree} \]
The atomic suffix tree

- The suffix tree contains every suffix of $T$ as a root to leaf path
The atomic suffix tree

- The suffix tree contains every suffix of $T$ as a root to leaf path
- Every edge is labelled with a character from $T$
The atomic suffix tree contains every suffix of $T$ as a root to leaf path.

Every edge is labelled with a character from $T$.

No two edges leaving the same node have the same label.
The atomic suffix tree

- The suffix tree contains every suffix of $T$ as a root to leaf path
- Every edge is labelled with a character from $T$
- No two edges leaving the same node have the same label
- Each leaf corresponds to a suffix (so there are $n$ leaves)
Searching in an atomic suffix tree
Searching in an atomic suffix tree

**How do you find a pattern?**
Searching in an atomic suffix tree

How do you find a pattern?
Searching in an atomic suffix tree

$T \begin{array}{cccccc}
    b & a & n & a & n & a & s \\
    \hline
    n
\end{array}$

$P \begin{array}{ccc}
    a & n & a \\
    \hline
    m
\end{array}$

How do you find a pattern?

start at the root and walk down the tree
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree
How do you find a pattern?

start at the root and walk down the tree
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree

... matches occur at the leaves of the subtree
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree
matches occur at the leaves of the subtree
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree
How do you find a pattern?

start at the root and walk down the tree

... matches occur at the leaves of the subtree
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree

... matches occur at the leaves of the subtree
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree
...matches occur at the leaves of the subtree
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree

We can decide whether $P$ matches somewhere in $O(m)$ time
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree

... matches occur at the leaves of the subtree

We can decide whether $P$ matches somewhere in $O(m)$ time

(we'll worry about outputting the matches later)
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree

We can decide whether $P$ matches somewhere in $O(m)$ time

*(we'll worry about outputting the matches later)*
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree

... matches occur at the leaves of the subtree

We can decide whether $P$ matches somewhere in $O(m)$ time

(we'll worry about outputting the matches later)

WARNING! How long does it take to find the correct child?

There could be $n$ edges here!

In this lecture we assume the alphabet size is a constant

This may be fine in some applications

(english text or DNA for example)

We can remove the assumption via the magic of hashing
Searching in an atomic suffix tree

How do you find a pattern?

start at the root and walk down the tree

... matches occur at the leaves of the subtree

We can decide whether $P$ matches somewhere in $O(m)$ time

(we'll worry about outputting the matches later)
how large is the atomic suffix tree?

There are at most $n$ leaves
how large is the atomic suffix tree?

There are at most $n$ leaves

that’s good right?
how large is the atomic suffix tree?

$T \ | \ a\ n\ a\ n\ a\ s$

There are at most $n$ leaves

that’s good right?

Unfortunately there can be lots of internal nodes
how large is the atomic suffix tree?

There are at most $n$ leaves

that's good right?

Unfortunately there can be lots of internal nodes
how large is the atomic suffix tree?

There are at most $n$ leaves

that’s good right?

Unfortunately there can be *lots* of internal nodes
how large is the atomic suffix tree?

There are at most $n$ leaves

that's good right?

Unfortunately there can be lots of internal nodes

7 characters
how large is the atomic suffix tree?

There are at most $n$ leaves

that's good right?

Unfortunately there can be *lots* of internal nodes

7 characters 23 nodes
how large is the atomic suffix tree?

There are at most $n$ leaves

that’s good right?

Unfortunately there can be lots of internal nodes

7 characters  23 nodes  that’s not so bad, right?
how large is the atomic suffix tree?
how large is the atomic suffix tree?
how large is the atomic suffix tree?
how large is the atomic suffix tree?

$T \begin{array}{c}
\text{a} \\
\text{b}
\end{array}$

4 nodes
how large is the atomic suffix tree?
how large is the atomic suffix tree?

- $T = a \ b$
  - 4 nodes

- $T = a \ a \ b \ b$
  - 9 nodes

- $T = a \ a \ a \ b \ b \ b$
  - 16 nodes
how large is the atomic suffix tree?

- $T[a, b]$ with 2 nodes
- $T[a, a, b, b]$ with 4 nodes
- $T[a, a, a, b, b, b]$ with 6 nodes
- $T[a, a, a, a, b, b, b, b, b, b]$ with 8 nodes
- $T[a, a, a, a, b, b, b, b, b, b, b, b, b, b, b, b]$ with 16 nodes
- $T[a, a, a, a, a, b, b, b, b, b, b, b, b, b, b, b, b, b, b, b, b, b, b, b, b, b, b, b, b, b, b, b, b, b, b]$ with 25 nodes
how large is the atomic suffix tree?

\[ T \begin{array}{c|c|c} a \mid b \end{array} \quad T \begin{array}{c|c|c|c} a \mid a \mid b \mid b \end{array} \quad T \begin{array}{c|c|c|c|c} a \mid a \mid a \mid b \mid b \mid b \mid b \end{array} \]

- 2 nodes
- 4 nodes
- 6 nodes
- 8 nodes
- 9 nodes
- 16 nodes
- 10 nodes
- 36 nodes
- 25 nodes
how large is the atomic suffix tree?

An atomic suffix tree can have \(((n/2) + 1)^2\) nodes.
how large is the atomic suffix tree?

An atomic suffix tree can have \(((n/2) + 1)^2\) nodes

this is far too big!
Compacted suffix trees

Why is the atomic suffix tree so big?
Why is the atomic suffix tree so big?

because it has long paths like this one
Compacted suffix trees

Why is the atomic suffix tree so big?

Main Idea replace each non-branching path with a single edge
Compacted suffix trees

Why is the atomic suffix tree so big?

Main Idea replace each non-branching path with a single edge

- edges are now labelled with substrings
Why is the atomic suffix tree so big?

Main Idea replace each non-branching path with a single edge
- edges are now labelled with substrings
  (instead of single characters)
Compacted suffix trees

Main Idea
replace each non-branching path with a single edge
- edges are now labelled with substrings
  (instead of single characters)
Compacted suffix trees

Main Idea replace each non-branching path with a single edge

- edges are now labelled with substrings
  
  (instead of single characters)
Compacted suffix trees

Main Idea replace each non-branching path with a single edge

- edges are now labelled with substrings

(instead of single characters)
Compacted suffix trees

Main Idea replace each non-branching path with a single edge

- edges are now labelled with substrings
  
  (instead of single characters)

- There are at most $n$ leaves
- Every internal node has two or more children

so there are $O(n)$ edges
Main Idea: replace each non-branching path with a single edge.

- Edges are now labelled with substrings (instead of single characters).

There are at most $n$ leaves.

Every internal node has two or more children.

So there are $O(n)$ edges.

Don’t the edges take up lots of space?
Compacted suffix trees

$T$  
\[ \begin{array}{cccccc}
  b & a & n & a & n & a & s \\
  \hline
  n
\end{array} \]

- There are at most $n$ leaves
- Every internal node has two or more children

so there are $O(n)$ edges

don't the edges take up lots of space?

we only store the end points

**Main Idea** replace each non-branching path with a single edge

- edges are now labelled with substrings

  \[(\text{instead of single characters})\]
Compacted suffix trees

Main Idea: replace each non-branching path with a single edge

- edges are now labelled with substrings

(instead of single characters)

- There are at most $n$ leaves
- Every internal node has two or more children

so there are $O(n)$ edges

don’t the edges take up lots of space?

we only store the end points

we actually store (4, 6)
Compacted suffix trees

$T \quad b\,a\,n\,a\,n\,a\,s$

$$n$$

Diagram of a compacted suffix tree for the string "bananas".
Compacted suffix trees

$T = \text{banana\,n\,as}$

Compacted Suffix Tree of $T$
Compacted suffix trees

$T \begin{array}{c} b \ a \ n \ a \ n \ a \ s \\ n \end{array}$

Compacted Suffix Tree of $T$

- A rooted tree with $n$ leaves
Compacted Suffix Trees

$T = \text{bananas}$

Compacted Suffix Tree of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
Compacted suffix trees

$T$  
\[ b a n a n a s \]

Compacted Suffix Tree of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring
Compacted suffix trees

Compacted Suffix Tree of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring
- No two edges leaving the same node have the same first character
Compacted suffix trees

$T$

\[ b a n a n a s \]

$\overline{n}$

Compacted Suffix Tree of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring
- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$
Compacted suffix trees

A rooted tree with $n$ leaves

Every internal node has two or more children

Every edge is labelled with a substring

No two edges leaving the same node have the same first character

Each leaf is labelled with a location in $T$

Any root-to-leaf path spells out the corresponding suffix
Compacted suffix trees

**Compacted Suffix Tree** of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring
- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$
- Any root-to-leaf path spells out the corresponding suffix

Uses $O(n)$ space
Compacted suffix trees

Compacted Suffix Tree of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring
- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$
- Any root-to-leaf path spells out the corresponding suffix

Sanity Check

Does the compacted suffix tree always exist?

Uses $O(n)$ space
Compacted suffix trees

Compacted Suffix Tree of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring
- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$
- Any root-to-leaf path spells out the corresponding suffix

Sanity Check

Does the compacted suffix tree always exist?

$T \quad b \quad b$

this doesn't have $n$ leaves

Uses $O(n)$ space
Compacted suffix trees

Compacted Suffix Tree of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring
- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$
- Any root-to-leaf path spells out the corresponding suffix

Sanity Check

Does the compacted suffix tree always exist?

- This doesn't have $n$ leaves
- This has $n$ leaves

Uses $O(n)$ space
Compacted suffix trees

Compacted Suffix Tree of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring
- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$
- Any root-to-leaf path spells out the corresponding suffix

Uses $O(n)$ space
Compacted suffix trees

Step one: Add a $\$$(unique symbol) to $T$

$T$  
\[
\begin{array}{c|c|c|c|c|c}
T & b & a & n & a & n & a & s \\
\hline
n
\end{array}
\]

Compacted Suffix Tree of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring
- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$
- Any root-to-leaf path spells out the corresponding suffix

Uses $O(n)$ space
Compacted suffix trees

**Step one:** Add a $\$$(unique symbol) to $T$

$T = \text{banana}\$s$

**Compacted Suffix Tree of $T$**

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring
- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$
- Any root-to-leaf path spells out the corresponding suffix

Uses $O(n)$ space
Compacted suffix trees

**Step one:** Add a $$(\text{unique symbol})$$ to $$T$$

$$T$$: $b a n a n a s$$

**Compacted Suffix Tree** of $$T$$

- A rooted tree with $$n$$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring
- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $$T$$
- Any root-to-leaf path spells out the corresponding suffix

Uses $$O(n)$$ space
Compacted suffix trees

Step one: Add a $ (unique symbol) to $T$

$T$  

Compacted Suffix Tree of $T$

- A rooted tree with $n$ leaves
- Every internal node has two or more children
- Every edge is labelled with a substring
- No two edges leaving the same node have the same first character
- Each leaf is labelled with a location in $T$
- Any root-to-leaf path spells out the corresponding suffix

Uses $O(n)$ space

This is normally just called a suffix tree
Searching in a compacted suffix tree

$T = \text{banana@nas}$

Diagram of the compacted suffix tree with nodes labeled for easy reference.
Searching in a compacted suffix tree

How do you find a pattern?
Searching in a compacted suffix tree

How do you find a pattern?
Searching in a compacted suffix tree

How do you find a pattern?

start at the root and walk down the tree
Searching in a compacted suffix tree

$T \quad b\ a\ n\ a\ n\ a\ s\ \$ 

$P \quad a\ n\ a\ n\ a\ n\ a$ 

How do you find a pattern?

start at the root and walk down the tree
Searching in a compacted suffix tree

How do you find a pattern?

start at the root and walk down the tree
Searching in a compacted suffix tree

How do you find a pattern?

start at the root and walk down the tree
Searching in a compacted suffix tree

How do you find a pattern?

- start at the root and walk down the tree

$T = b a n a n a s$

$P = a n a n a$

---

Remember that an edge is actually stored as a pair $we're$ actually looking in $T$. 

---
Searching in a compacted suffix tree

*How do you find a pattern?*

start at the root and walk down the tree
Searching in a compacted suffix tree

$T$  
<table>
<thead>
<tr>
<th>b a n a n a s $</th>
</tr>
</thead>
</table>

$n$

$P$  
<table>
<thead>
<tr>
<th>a n a n a a</th>
</tr>
</thead>
</table>

$m$

How do you find a pattern?

start at the root and walk down the tree
Searching in a compacted suffix tree

How do you find a pattern?

start at the root and walk down the tree
Searching in a compacted suffix tree

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree
Searching in a compacted suffix tree

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree
Searching in a compacted suffix tree

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree
Searching in a compacted suffix tree

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree
Searching in a compacted suffix tree

How do you find a pattern?

- start at the root and walk down the tree
- ...matches occur at the leaves of the subtree
Searching in a compacted suffix tree

How do you find a pattern?

start at the root and walk down the tree

... matches occur at the leaves of the subtree
Searching in a compacted suffix tree

How do you find a pattern?

- start at the root and walk down the tree
- matches occur at the leaves of the subtree
Searching in a compacted suffix tree

$T$ | b a n a n a s $
---|-------------------------------
    | n

$P$ | a n a n a a
   | ----- m

$P'$ | n a

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree
Searching in a compacted suffix tree

\[ T \begin{array}{c}
  b & a & n & a & n & a & s & \$
\end{array}
\]

\[ P \begin{array}{c}
  a & n & a & n & a
\end{array}
\]

\[ P' \begin{array}{c}
  n & a
\end{array}
\]

How do you find a pattern?

- start at the root and walk down the tree
- matches occur at the leaves of the subtree
Searching in a compacted suffix tree

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree
Searching in a compacted suffix tree

How do you find a pattern?

start at the root and walk down the tree

... matches occur at the leaves of the subtree
Searching in a compacted suffix tree

How do you find a pattern?

start at the root and walk down the tree

...matches occur at the leaves of the subtree

T: banana$ 

P: banana 

P': na 

how big is this subtree?
Searching in a compacted suffix tree

**How do you find a pattern?**

start at the root and walk down the tree

...matches occur at the leaves of the subtree

---

**T**

```
banana$s

---

n
```

**P**

```
banana

---

m
```

**P'**

```
n$a
```

---

**how big is this subtree?**

\[ O(occ) \] because it has \( occ \) leaves
Searching in a compacted suffix tree

How do you find a pattern?

- start at the root and walk down the tree
- matches occur at the leaves of the subtree

We can find all the matches in $O(m + \text{occ})$ time (by looking at the whole subtree)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  
  \((as\ if\ you\ were\ matching\ a\ pattern)\)

- Add a new edge and leaf for the new suffix
  
  \((this\ may\ require\ you\ to\ break\ an\ edge\ in\ two)\)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

• Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)

• Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

1. Insert the suffixes one at a time (longest first)
   - Search for the new suffix in the partial suffix tree
     (as if you were matching a pattern)
   - Add a new edge and leaf for the new suffix
     (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree 
  \((\text{as if you were matching a pattern})\)

- Add a new edge and leaf for the new suffix
  \((\text{this may require you to break an edge in two})\)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree 
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix 
  (this may require you to break an edge in two)

we actually store this as \( (0, 7) \)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)
- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)
- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree (as if you were matching a pattern)
- Add a new edge and leaf for the new suffix (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)
- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  \[\text{(as if you were matching a pattern)}\]
- Add a new edge and leaf for the new suffix
  \[\text{(this may require you to break an edge in two)}\]
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)
- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)
- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree 
  
  \textit{(as if you were matching a pattern)}

- Add a new edge and leaf for the new suffix 
  
  \textit{(this may require you to break an edge in two)}
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix
  
  *(this may require you to break an edge in two)*
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)
- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  
  \textit{(as if you were matching a pattern)}

- Add a new edge and leaf for the new suffix
  
  \textit{(this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)
- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)
- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix

  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree (as if you were matching a pattern)
- Add a new edge and leaf for the new suffix (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  *(as if you were matching a pattern)*
- Add a new edge and leaf for the new suffix
  *(this may require you to break an edge in two)*
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix
  
  *(this may require you to break an edge in two)*
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)
- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree (as if you were matching a pattern)
- Add a new edge and leaf for the new suffix (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)
- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree (as if you were matching a pattern)
- Add a new edge and leaf for the new suffix (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

<table>
<thead>
<tr>
<th>Suffixes</th>
<th>Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>bananas</td>
<td>0</td>
</tr>
<tr>
<td>na</td>
<td>5</td>
</tr>
<tr>
<td>$nas$</td>
<td>1</td>
</tr>
<tr>
<td>$a$s</td>
<td>3</td>
</tr>
<tr>
<td>$s$</td>
<td>7</td>
</tr>
<tr>
<td>$n$</td>
<td>2</td>
</tr>
<tr>
<td>$a$s</td>
<td>4</td>
</tr>
</tbody>
</table>

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix
  *(this may require you to break an edge in two)*
Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  
  *(as if you were matching a pattern)*

- Add a new edge and leaf for the new suffix

  *(this may require you to break an edge in two)*
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  
  \( \text{(as if you were matching a pattern)} \)

- Add a new edge and leaf for the new suffix
  
  \( \text{(this may require you to break an edge in two)} \)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)
- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

- Search for the new suffix in the partial suffix tree
  (as if you were matching a pattern)

- Add a new edge and leaf for the new suffix
  (this may require you to break an edge in two)

This takes $O(n)$ time per suffix…
Naively constructing a compacted suffix tree

Insert the suffixes one at a time (longest first)

• Search for the new suffix in the partial suffix tree

(\textit{as if you were matching a pattern})

• Add a new edge and leaf for the new suffix

(\textit{this may require you to break an edge in two})

This takes $O(n)$ time per suffix... so $O(n^2)$ time in total
The (compacted) suffix tree of a (length $n$) text uses $O(n)$ space.

Finding all matches of a pattern $P$ of length $m$ takes $O(m + \text{occ})$ where $\text{occ}$ is the number of matches.

Suffix trees can be built in $O(n)$ time but we have only seen the $O(n^2)$ time method.

We assumed that the alphabet contained a constant number of symbols.
The suffix array - a sneak preview

T: b a n a n a s

n
The suffix array - a sneak preview

\[ T = \text{banana} \]

\[ n \]

0  \[ b\ a\ n\ a\ n\ a\ s \]
1  \[ a\ n\ a\ n\ a\ s \]
2  \[ n\ a\ n\ a\ s \]
3  \[ a\ n\ a\ s \]
4  \[ n\ a\ s \]
5  \[ a\ s \]
6  \[ s \]
The suffix array - a sneak preview

$T = \text{banana}$

$\text{suffix array}$

0: $b\ an\ an\ an\ s$
1: $a\ an\ an\ an\ s$
2: $n\ an\ an\ a\ s$
3: $a\ an\ a\ s$
4: $n\ a\ s$
5: $a\ s$
6: $s$
The suffix array - a sneak preview

\[ T = \text{banana} \]

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>b</td>
<td>a</td>
<td>n</td>
<td>a</td>
<td>n</td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>n</td>
<td>a</td>
<td>n</td>
<td>a</td>
<td>s</td>
</tr>
<tr>
<td>2</td>
<td>n</td>
<td>a</td>
<td>n</td>
<td>a</td>
<td>s</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>n</td>
<td>a</td>
<td>s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>n</td>
<td>a</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>a</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The suffix array - a sneak preview

Sort the suffixes lexicographically

<table>
<thead>
<tr>
<th>Index</th>
<th>Suffix</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>b a n a n a s</td>
</tr>
<tr>
<td>1</td>
<td>a n a n a s</td>
</tr>
<tr>
<td>2</td>
<td>n a n a s</td>
</tr>
<tr>
<td>3</td>
<td>a n a s</td>
</tr>
<tr>
<td>4</td>
<td>n a s</td>
</tr>
<tr>
<td>5</td>
<td>a s</td>
</tr>
<tr>
<td>6</td>
<td>s</td>
</tr>
</tbody>
</table>
The suffix array - a sneak preview

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

Sort the suffixes lexicographically

- The symbols must have an order throughout we will use alphabetical order
The suffix array - a sneak preview

Sort the suffixes lexicographically

- The symbols must have an order throughout we will use alphabetical order

In lexicographical ordering we sort strings based on the first symbol that differs:
The suffix array - a sneak preview

Sort the suffixes lexicographically

- The symbols must have an order

throughout we will use alphabetical order

In lexicographical ordering we sort strings based on the first symbol that differs:

\[
\begin{align*}
&aa < ba \\
&ab < ba \\
&ba < ba
\end{align*}
\]
The suffix array - a sneak preview

Sort the suffixes lexicographically

- The symbols must have an order throughout we will use alphabetical order

In lexicographical ordering we sort strings based on the first symbol that differs:

\[ \text{a} \text{a} < \text{b} \text{a} \]
### The suffix array - a sneak preview

<table>
<thead>
<tr>
<th>Suffix Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
</tr>
<tr>
<td>$b a n a n a s$</td>
</tr>
</tbody>
</table>

#### Sort the suffixes lexicographically

- The symbols must have an order throughout we will use alphabetical order.

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

In lexicographical ordering we sort strings based on the first symbol that differs:

- $a a < b a$
The suffix array - a sneak preview

\[ T \quad b \quad a \quad n \quad a \quad n \quad a \quad s \]

Sort the suffixes lexicographically

- The symbols must have an order throughout we will use alphabetical order

In lexicographical ordering we sort strings based on the first symbol that differs:

\[ a \ a \quad < \quad b \ a \quad < \quad b \ c \]
The suffix array - a sneak preview

Sort the suffixes lexicographically

- The symbols must have an order

throughout we will use alphabetical order

In lexicographical ordering we sort strings based on the first symbol that differs:

\[
\begin{align*}
\text{a a} & < \text{b a} < \text{b c}
\end{align*}
\]
The suffix array - a sneak preview

Sort the suffixes lexicographically

- The symbols must have an order throughout we will use alphabetical order

In lexicographical ordering we sort strings based on the first symbol that differs:

\[
\begin{align*}
\text{a a} & \ < \ \text{b a} \ < \ \text{b c}
\end{align*}
\]
The suffix array - a sneak preview

Sort the suffixes lexicographically

- The symbols must have an order throughout we will use alphabetical order

In lexicographical ordering we sort strings based on the first symbol that differs:

```
a a  <  b a  <  b c  <  b c a
```
The suffix array - a sneak preview

Sort the suffixes lexicographically

- The symbols must have an order
throughout we will use alphabetical order

In lexicographical ordering we sort strings based on the first symbol that differs:

\[
\begin{align*}
& a \ a < b \ a < b \ c < b \ c \ a \\
\end{align*}
\]
The suffix array - a sneak preview

Sort the suffixes lexicographically

- The symbols must have an order throughout we will use alphabetical order

In lexicographical ordering we sort strings based on the first symbol that differs:

\[
\begin{align*}
\text{a a} & \ < \ \text{b a} \ < \ \text{b c} \ < \ \text{b c a} \\
\end{align*}
\]

(in a ‘tie’, the shorter string is smaller)
The suffix array - a sneak preview

Sort the suffixes lexicographically

The symbols must have an order throughout we will use alphabetical order

In lexicographical ordering we sort strings based on the first symbol that differs:

\[
\begin{align*}
\text{aa} & < \text{ba} < \text{bc} < \text{bc}a \\
\end{align*}
\]

(in a ‘tie’, the shorter string is smaller)
The suffix array - a sneak preview

Sort the suffixes lexicographically

- The symbols must have an order

Throughout we will use alphabetical order

In lexicographical ordering we sort strings based on the first symbol that differs:

\[
a a < b a < b c < b c a
\]

(in a ‘tie’, the shorter string is smaller)
The suffix array - a sneak preview

Sort the suffixes lexicographically

- The symbols must have an order throughout we will use alphabetical order

In lexicographical ordering we sort strings based on the first symbol that differs:

\[
\begin{align*}
\text{a a} & < \text{b a} & < \text{b c} & < \text{b c a} \\
\end{align*}
\]

(in a ‘tie’, the shorter string is smaller)
The suffix array - a sneak preview

Sort the suffixes lexicographically

- The symbols must have an order throughout we will use alphabetical order

In lexicographical ordering we sort strings based on the first symbol that differs:

```
| a a |   | b a |   | b c |   | b c a |
```

(in a 'tie', the shorter string is smaller)
The suffix array - a sneak preview

Sort the suffixes lexicographically

- The symbols must have an order throughout we will use alphabetical order

In lexicographical ordering we sort strings based on the first symbol that differs:

```
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
a | a | a | b | a | b | c | b | c | a
```

(in a ‘tie’, the shorter string is smaller)
The suffix array - a sneak preview

Sort the suffixes lexicographically

- The symbols must have an order throughout we will use alphabetical order

In lexicographical ordering we sort strings based on the first symbol that differs:

\[
\begin{align*}
  a & \ a & < & \ b & \ a & < & \ b & \ c & < & \ b & \ c & \ a \\
\end{align*}
\]

(in a ‘tie’, the shorter string is smaller)

If the symbols don’t have a natural order, we use their binary representation in memory
The suffix array - a sneak preview

Sort the suffixes lexicographically.
The suffix array - a sneak preview

<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>$T$</td>
<td>$b$</td>
<td>$a$</td>
<td>$n$</td>
<td>$a$</td>
<td>$n$</td>
<td>$a$</td>
<td>$s$</td>
</tr>
</tbody>
</table>

Sort the suffixes lexicographically

<table>
<thead>
<tr>
<th></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>$a$</td>
<td>$n$</td>
<td>$a$</td>
<td>$n$</td>
<td>$a$</td>
<td>$s$</td>
</tr>
<tr>
<td></td>
<td>$a$</td>
<td>$n$</td>
<td>$a$</td>
<td>$s$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$a$</td>
<td>$s$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$b$</td>
<td>$a$</td>
<td>$n$</td>
<td>$a$</td>
<td>$n$</td>
<td>$a$</td>
</tr>
<tr>
<td></td>
<td>$n$</td>
<td>$a$</td>
<td>$n$</td>
<td>$a$</td>
<td>$s$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$n$</td>
<td>$a$</td>
<td>$s$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$s$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The suffix array - a sneak preview

Sort the suffixes lexicographically

<table>
<thead>
<tr>
<th>T</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>b a n a n a s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<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>a n a n a s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a n a s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b a n a n a s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n a n a s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n a s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Suffix Array

<table>
<thead>
<tr>
<th>Suffix Array</th>
<th>1</th>
<th>3</th>
<th>5</th>
<th>0</th>
<th>2</th>
<th>4</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n</td>
<td></td>
<td></td>
<td>n</td>
<td></td>
<td></td>
<td>n</td>
</tr>
</tbody>
</table>
The suffix array - a sneak preview

Sort the suffixes lexicographically
The suffix array - a sneak preview

Sort the suffixes lexicographically

Suffix Array

\[ T \begin{array}{ccccccc}
  b & a & n & a & a & n & s \\
  0 & 1 & 2 & 3 & 4 & 5 & 6
\end{array} \]

\[ \begin{array}{ccccccc}
  1 & a & n & a & a & n & s \\
  3 & a & n & a & s \\
  5 & a & s \\
  0 & b & a & n & a & a & n & s \\
  2 & n & a & n & a & s \\
  4 & n & a & s \\
  6 & s
\end{array} \]
The suffix array - a sneak preview

Sort the suffixes lexicographically

The suffix array is much smaller than the suffix tree (in terms of constants)
The suffix array - a sneak preview

Sort the suffixes lexicographically

The suffix array is much smaller than the suffix tree (in terms of constants)
Constructing the Suffix Array from the Suffix Tree

Recall that we added a unique symbol $\$$ to make sure the tree exists.

- the $\$$ is the smallest symbol in the alphabet.
To get the Suffix array perform a depth-first search (in lexicographical order)

recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet
Constructing the Suffix Array from the Suffix Tree

recall that we added a unique symbol \$ to make sure the tree exists
- the \$ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

Recall that we added a unique symbol $\$$ to make sure the tree exists.
- the $\$$ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

Recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

recall that we added a unique symbol $ to make sure the tree exists

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

T
\[
\begin{array}{ccccccc}
\text{b} & \text{a} & \text{n} & \text{a} & \text{n} & \text{a} & \text{s} \\
0 & 1 & 2 & 3 & 4 & 5 & 6
\end{array}
\]

Suffix Array
\[
\begin{array}{ccccccc}
1 & 3 & 5 & 0 & 2 & 4 & 6 \\
\end{array}
\]

recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

Recall that we added a unique symbol $ to make sure the tree exists.
- The $ is the smallest symbol in the alphabet.

To get the Suffix array perform a depth-first search (in lexicographical order).
Constructing the Suffix Array from the Suffix Tree

Recall that we added a unique symbol $\$ to make sure the tree exists
- the $\$ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

Recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

recall that we added a unique symbol $\$$ to make sure the tree exists
- the $\$$ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

recall that we added a unique symbol $ to make sure the tree exists
  - the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

recall that we added a unique symbol $\$ to make sure the tree exists
- the $\$ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

Recall that we added a unique symbol $ to make sure the tree exists.

- The $ is the smallest symbol in the alphabet.

To get the Suffix array perform a depth-first search (in lexicographical order).
Constructing the Suffix Array from the Suffix Tree

recall that we added a unique symbol $ to make sure the tree exists

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

 recall that we added a unique symbol $ to make sure the tree exists

 - the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

Recall that we added a unique symbol $\$ to make sure the tree exists
- the $\$ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
recall that we added a unique symbol $\$$ to make sure the tree exists

- the $\$$ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

```
T
b a n a n a s
0 1 2 3 4 5 6
```

Suffix Array
```
1 3 5 0 2 4 6
```

Recall that we added a unique symbol $ to make sure the tree exists.
- The $ is the smallest symbol in the alphabet.

To get the Suffix array perform a depth-first search (in lexicographical order).
Constructing the Suffix Array from the Suffix Tree

Recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

recall that we added a unique symbol $ to make sure the tree exists

- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

\[ T = \text{banana}s \]

Suffix Array

\[
\begin{array}{ccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\end{array}
\]

recall that we added a unique symbol \$ to make sure the tree exists

- the \$ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)
Constructing the Suffix Array from the Suffix Tree

recall that we added a unique symbol $ to make sure the tree exists
- the $ is the smallest symbol in the alphabet

To get the Suffix array perform a depth-first search (in lexicographical order)

this takes $O(n)$ time
The (compacted) suffix tree of a (length $n$) text uses $O(n)$ space

- Finding all matches of a pattern $P$ of length $m$ takes $O(m + \text{occ})$
  
  where $\text{occ}$ is the number of matches

- Suffix trees can be built in $O(n)$ time
  
  but we have only seen the $O(n^2)$ time method

we assumed that the alphabet contained a constant number of symbols
How can we index multiple texts?
Multiple text indexing

How can we index multiple texts?
Multiple text indexing

How can we index multiple texts?
How can we index multiple texts?
How can we index multiple texts?
How can we index multiple texts?

- build a generalised suffix tree in $O(n_1 + n_2)$ space
How can we index multiple texts?

- **build a generalised suffix tree in** $O(n_1 + n_2)$ **space**
- **using the linear time method** (which we omitted), **this takes** $O(n_1 + n_2)$ **time**
How can we index multiple texts?

- build a generalised suffix tree in $O(n_1 + n_2)$ space
- using the linear time method (which we omitted), this takes $O(n_1 + n_2)$ time
- Finding all matches of a pattern $P$ of length $m$ still takes $O(m + \text{occ})$ time
  where $\text{occ}$ is the number of matches