Advanced Algorithms – COMS31900

Pattern Matching part one

Suffix Trees

Raphaël Clifford

Slides by Benjamin Sach
**Exact pattern matching**

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

\[
\begin{array}{c}
\text{T} \\
\text{P}
\end{array}
\]

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

\[
\begin{array}{c}
n \\
m
\end{array}
\]

**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 $m$)

![Diagram](image)

**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 $m$)

$$
\begin{array}{c}
T \quad \begin{array}{ccccccc}
a & b & c & b & a & b & a & b & a & \cdots & a & b & a & c & a & b & a \\
\end{array} \\
\hline \\
P \quad \begin{array}{c}
6 & a & b & a \\
\end{array} \\
\hline \\
\end{array}
$$

**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 \( m \))

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

\[
\begin{array}{c}
P = \text{Valid} ; \\
10 \ a \ b \ a
\end{array}
\]

**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)
**Exact pattern matching**

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

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

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

**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)*
Exact pattern matching

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

![Text and Pattern Diagram]

**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 $m$)

\[
T = \text{a b c b a b a b a c a b a}
\]

\[
P = 6 \text{ a b a} \quad \checkmark
\]

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

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

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

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

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

\[
\begin{array}{cccc}
P & a & b & a \\
\end{array}
\]

\[
\begin{array}{c}
m \\
\end{array}
\]
Text indexing

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

![Diagram showing a text string $T$ and a query $P$]

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

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

e.g. 4, 6, 10
Text indexing

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

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

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

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

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...

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

4 6 10

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

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

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 occ

  - occ is the number of occurrences (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 $T$.

e.g. 4, 6, 10

- 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 occurrences (matches)

- We also want $O(n)$ space and fast preprocessing (prep.) time
The atomic suffix tree

\[ T = \begin{array}{c}
| b | a | n | a | n | a | s |
\end{array}
\]

---

\[ \overline{n} \]
The atomic suffix tree

$T$

```
b a n a n a s
   n
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
```
suffixes
The atomic suffix tree

\[ T \]

\[ b\ a\ n\ a\ n\ a\ s \]

suffix tree

\[ a\ b\ n\ s \]

suffixes

\[ b\ a\ n\ a\ n\ a\ s \]

\[ a\ n\ a\ n\ a\ s \]

\[ n\ a\ n\ a\ s \]

\[ a\ n\ a\ s \]

\[ n\ a\ s \]

\[ a\ s \]

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

T

bananas

suffixes

bananas

bananas

nanas

anas

as

s

suffix tree

bananas

nana

nana

nas

as

s

bananas
The atomic suffix tree

$T = \text{banana}$

Suffixes:
- banana
- anana
- anas
- nas
- as
- s
The atomic suffix tree
The atomic suffix tree

$T = \text{banana}$

suffixes

$n a s$

$a s$

$s$
The atomic suffix tree

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

- Suffixes:
  - \( b a n a n a s \)
  - \( a n a n a s \)
  - \( n a n a s \)
  - \( a n a s \)
  - \( n a s \)
  - \( a s \)
  - \( s \)
The atomic suffix tree
The atomic suffix tree

T

banana

suffixes

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

suffix tree

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>s</th>
</tr>
</thead>
</table>
The atomic suffix tree

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

\[ b\ a\ n\ a\ n\ a\ s \quad 0 \]

\[ a\ n\ a\ n\ a\ s \quad 1 \]

\[ n\ a\ n\ a\ s \quad 2 \]

\[ a\ n\ a\ s \quad 3 \]

\[ n\ a\ s \quad 4 \]

\[ a\ s \quad 5 \]

\[ s \quad 6 \]

suffixes

suffix tree

\[ a \quad n \quad s \]

\[ a \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ a \quad n \quad a \quad s \]

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

\[ a \quad a \quad n \quad s \]

\[ a \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]

\[ n \quad a \quad n \quad s \]
The atomic suffix tree

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

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

T

\[
\begin{array}{cccccc}
  b & a & n & a & n & a \\
\end{array}
\]

\[n\]
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

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
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
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
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

Given the text `T = bananas` and the pattern `P = banana`, we can see that `P` matches `T` at node 6.

However, the pattern `P' = naba` does not match `T`.

**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

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

\[ \begin{array}{l}
P \ [ a \, n \, a ] \checkmark \\
P' \ [ n \, a \, b ] \times \\
\end{array} \]

How do you find a pattern?

start at the root and walk down the tree

\[ \ldots \text{matches occur at the leaves of the subtree} \]

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
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?

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 \hspace{1cm} 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?

$T \begin{array}{c|c}
   & \text{2} \\
\hline
   a & b
\end{array}$

4 nodes
how large is the atomic suffix tree?

2 nodes

4 nodes

9 nodes
how large is the atomic suffix tree?

- 2 nodes
- 4 nodes
- 6 nodes
how large is the atomic suffix tree?

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

- $T = ab$, 4 nodes
- $T = aab$, 9 nodes
- $T = aab$, 16 nodes
- $T = aaaaabbbbb$, 36 nodes
- $T = aaaaabbbbb$, 25 nodes
how large is the atomic suffix tree?

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

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

this is far too big!
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
Compact 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
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

*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

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

- There are at most $n$ leaves

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

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

- edges are now labelled with substrings

(instead of single characters)
Compacted suffix trees

\[ T \]

- 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

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

- 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?

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

- edges are now labelled with substrings

(instead of single characters)
Compacted suffix trees

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

- 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

  \((\text{instead of single characters})\)
**Main Idea** replace each non-branching path with a single edge

- edges are now labelled with substrings

*(instead of single characters)*
Compacted suffix trees

$T \quad bananas$

Diagram:

```
        a
       /\  
      na  s
     / \  /
   na   s s
  /   \ /  
 1    3  5
```

Node labels:
- 1: na
- 3: nas
- 5: bananas
- 0: na
- 2: nas
- 4: s
- 6: s

Compacted suffix trees

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

Compacted Suffix Tree of $T$
Compacted suffix trees

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

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

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

$T$: bananas

Diagram: A rooted tree with nodes labelled with substrings such as na, s, bananas, na, and s, with numbers 0, 2, 4, 5, 6, 1, and 3.
Compacted suffix trees

$T$ = bananas

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

**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

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
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

$T = \text{bananas}$
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?

$T$ uses $O(n)$ space

this doesn’t have $n$ leaves
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 $T$

$T$ \hspace{1cm} bananas

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 = \begin{array}{cccccc}
  b & a & n & a & n & a & s & \$
\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**

---

The diagram shows the compacted suffix tree of the string $ T $, which is a rooted tree with $ n $ leaves and uses $ O(n) $ space. Each internal node has two or more children, and each edge is labelled with a substring from the string. The tree structure is such that no two edges leaving the same node have the same first character.
Compacted suffix trees

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

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

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

T

banana

$
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

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

**T**

- bananas

**P**

- anana

---

How do you find a pattern?

- start at the root and walk down the tree

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

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
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

$T$ \hspace{2cm} \begin{array}{c} b \ a \ n \ a \ n \ a \ s \$ \end{array}

$P$ \hspace{2cm} \begin{array}{c} a \ n \ a \ n \ a \$ \end{array}$

$P'$ \hspace{2cm} \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

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

$P$ \begin{array}{c}
\text{a\ n\ a\ n\ a} \\
\underline{m}
\end{array} \quad \checkmark

$P'$ \begin{array}{c}
\text{n\ a} \\
\underline{}\underline{}\underline{}
\end{array} \quad \checkmark

How do you find a pattern?

start at the root and walk down the tree

\ldots 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

The text explains how to search for patterns in a compacted suffix tree. The tree is traversed starting from the root and walking down the tree. Matches occur at the leaves of the subtree.

Example:

- Tree T: "bananas$"
- Pattern P: "banana" 
- Pattern P': "na"

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
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

O(occ) because it has occ leaves
(and each internal node has at least two children)

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

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

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

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

Never actually do it like this.
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
  
  \(\text{as if you were matching a pattern}\)

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

*you should never actually do it like this*
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)*

You should never actually do it like this
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)
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)*

you should *never actually do it like this*
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)

You should never actually do it like this.
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)

**You should never actually do it like this**
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)}

you should \textit{never actually do it like this}
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)

never actually
do it like this
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)

You should never actually do it like this.
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)*

---

*You should never actually do it like this*
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
  
  \( \text{(as if you were matching a pattern)} \)

- 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
  
  \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)
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)*

You should never actually do it like this.
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)

You should never actually do it like this.
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
  \((this \ may \ require \ you \ to \ break \ an \ edge \ in \ two)\)

\(T\)

\[\begin{align*}
T & \quad b \ a \ n \ a \ n \ a \ s \ $
\end{align*}\]

\(n\)

\[\begin{align*}
b & \quad a \ n \ a \ n \ a \ s \ $
\end{align*}\]

\[\begin{align*}
a & \quad n \ a \ n \ a \ s \ $
\end{align*}\]

\[\begin{align*}
n & \quad a \ n \ a \ s \ $
\end{align*}\]

\[\begin{align*}
a & \quad n \ a \ s \ $
\end{align*}\]

\[\begin{align*}
n & \quad a \ s \ $
\end{align*}\]

\[\begin{align*}
a & \quad s \ $
\end{align*}\]

\[\begin{align*}
s & \quad $
\end{align*}\]

\[\begin{align*}
\quad \quad 0 \ \checkmark
\end{align*}\]

\[\begin{align*}
\quad \quad 1 \ \checkmark
\end{align*}\]

\[\begin{align*}
\quad \quad 2 \ \checkmark
\end{align*}\]

\[\begin{align*}
\quad \quad 3 \ \checkmark
\end{align*}\]

\[\begin{align*}
\quad \quad 4 \ \checkmark
\end{align*}\]

\[\begin{align*}
\quad \quad 5 \ \checkmark
\end{align*}\]

\[\begin{align*}
\quad \quad 6
\end{align*}\]

\[\begin{align*}
\quad \quad 7
\end{align*}\]
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
  \[as \text{ (as if you were matching a pattern)}\]
- 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)

T

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

suffices

\[
\begin{array}{c|c|c|c|c|c|c|c}
|   | b & a & n & a & n & a & s & $ \\
\hline
|   | 0 & \checkmark \\
\hline
|   | 1 & \checkmark \\
\hline
|   | 2 & \checkmark \\
\hline
|   | 3 & \checkmark \\
\hline
|   | 4 & \checkmark \\
\hline
|   | 5 & \checkmark \\
\hline
|   | 6 & \\
\hline
|   | 7 & \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c|c|c}
|   | bananas & \text{na} & \text{nas} & \text{s} \\
\hline
|   | 0 & \\
\hline
|   | 2 & \\
\hline
|   | 4 & \\
\hline
|   | 1 & \\
\hline
|   | 3 & \\
\end{array}
\]
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)

You should never actually do it like this.
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)

you should never actually do it like this
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
  \[\text{(as if you were matching a pattern)}\]
- Add a new edge and leaf for the new suffix
  \[\text{(this may require you to break an edge in two)}\]

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
  \[(\text{as if you were matching a pattern})\]

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

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.

you should actually do it like this (or build a suffix array instead)

we assumed that the alphabet contained a constant number of symbols
Multiple text indexing

How can we index multiple texts?
How can we index multiple texts?
Multiple text indexing

$T \quad \text{bananas} \quad \text{apples} \quad 
\quad \text{apples} \quad 
\quad \text{n} \\
T_1 \quad \text{bananas} \quad \$ \quad n_1 \\
T_2 \quad \text{apples} \quad \& \quad n_2 \\
two \text{distinct unique symbols}$

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
Multiple text indexing

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** $O(m + occ)$ **time**
  
  *where occ is the number of matches*
The suffix array - a sneak preview

\[ T \quad b \quad a \quad n \quad a \quad n \quad a \quad s \]
The suffix array - a sneak preview

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

<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>
The suffix array - a sneak preview

$T$  

\[ b\ a\ n\ a\ a\ a\ s \]

\[ \text{suffix} \]

0  
\[ b\ a\ n\ a\ n\ a\ s \]

1  
\[ a\ n\ a\ a\ n\ s \]

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

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

\[ n \]

\begin{align*}
0 & \quad b a n a n a s \\
1 & \quad a n a n a s \\
2 & \quad n a n a s \\
3 & \quad a n a s \\
4 & \quad n a s \\
5 & \quad a s \\
6 & \quad s \\
\end{align*}
The suffix array - a sneak preview

Sort the suffixes lexicographically

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

<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

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

Sort the suffixes lexicographically

- The symbols themselves 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

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

Sort the suffixes **lexicographically**

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

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

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

Sort the suffixes **lexicographically**

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

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

\[
egin{array}{c}
\text{a a} \quad < \quad \text{b a}
\end{array}
\]
The suffix array - a sneak preview

Sort the suffixes lexicographically

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

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

\[
\begin{array}{cc}
\text{a a} & \text{b a}
\end{array}
\]
The suffix array - a sneak preview

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

**Sort the suffixes lexicographically**

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

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

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

Sort the suffixes **lexicographically**

- The symbols themselves 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

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

Sort the suffixes lexicographically

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

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

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

Sort the suffixes lexicographically

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

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

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

Sort the suffixes lexicographically

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

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

\[
\begin{align*}
\text{a a} & < \text{b a} < \text{b c} < \text{b c a} \\
\text{(in a ‘tie’, the shorter string is smaller)}
\end{align*}
\]
The suffix array - a sneak preview

Sort the suffixes lexicographically

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

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

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

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

$T = b a n a n a s$

Sort the suffixes lexicographically

- The symbols themselves 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 themselves 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 themselves 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 themselves 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 themselves must have an order throughout we will use alphabetical order just a fancy name for the order the strings would appear in a dictionary

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

$$\begin{align*}
  \text{\texttt{aa}} &< \text{\texttt{ba}} < \text{\texttt{bc}} < \text{\texttt{bca}} \\
\end{align*}$$

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

Sort the suffixes lexicographically

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

Just a fancy name for the order the strings would appear in a dictionary

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

\[
\begin{align*}
\text{a a} & < \text{b a} < \text{b c} < \text{b c a} \\
\text{(in a ‘tie’, the shorter string is smaller)}
\end{align*}
\]

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

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

Sort the suffixes lexicographically

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

Sort the suffixes lexicographically

Suffix Array

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

Sort the suffixes lexicographically

Suffix Array

1 3 5 0 2 4 6
The suffix array - a sneak preview

Sort the suffixes lexicographically

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

| 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           |

Suffix Array

| 1  | 3  | 5  | 0  | 2  | 4  | 6  |
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

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

Sort the suffixes lexicographically

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

Suffix Array: 1 3 5 0 2 4 6

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
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**

- **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

\[ T = \textbf{ba}\textbf{n}\textbf{a}\textbf{n}\textbf{a}\textbf{n}\textbf{a}\$ \]

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

\textbf{Suffix Array}

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

\[ n \]

\textit{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

- The $ symbol is added to make sure the tree exists.
- The $ symbol 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}{cccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\end{array}

\begin{array}{c}
\text{Suffix Array} \\
1 & 3 & 5 & 0 & 2 & 4 & 6 \\
\end{array}

\begin{array}{cccccc}
\begin{array}{cccccc}
1 & 3 & 5 & 0 & 2 & 4 & 6 \\
\end{array}
\end{array}

\begin{array}{cccccc}
\begin{array}{cccccc}
1 & 3 & 5 & 0 & 2 & 4 & 6 \\
\end{array}
\end{array}

\begin{array}{cccccc}
\begin{array}{cccccc}
1 & 3 & 5 & 0 & 2 & 4 & 6 \\
\end{array}
\end{array}

\begin{array}{cccccc}
\begin{array}{cccccc}
1 & 3 & 5 & 0 & 2 & 4 & 6 \\
\end{array}
\end{array}

\begin{array}{cccccc}
\begin{array}{cccccc}
1 & 3 & 5 & 0 & 2 & 4 & 6 \\
\end{array}
\end{array}

\begin{array}{cccccc}
\begin{array}{cccccc}
1 & 3 & 5 & 0 & 2 & 4 & 6 \\
\end{array}
\end{array}

\begin{array}{cccccc}
\begin{array}{cccccc}
1 & 3 & 5 & 0 & 2 & 4 & 6 \\
\end{array}
\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).
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

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

We start by constructing the suffix tree for the string `bananas
```plaintext
T
b a n a n a s
```
with `$` added as a unique symbol.

- The `$` is the smallest symbol in the alphabet.

The suffix array is:
```
Suffix Array
1 3 5 0 2 4 6
```

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)

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 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 contains a constant number of symbols.