String Matching

Pattern: CCATT
Text: ACTGCCATTCCTTAGGCCATGTG

- Brute force & variant
- Automata-like strategies
- Suffix trees

To do:
[CLRS] 32
Supplements #5
Some Applications

• Text & programming languages
  – Spell-checking
  – Linguistic analysis
  – Tokenization
  – Virus scanning
  – Spam filtering
  – Database querying
• DNA sequence analysis
• Music identification & analysis

Brute Force Exact Match

Pattern: CCATT
Text: ACTGCCATTCCTTAGGCCATGTG

Algorithm?
Example of worst case?
Running time?
Rabin-Karp, 1981

Pattern:  **CCATT**

Text:  **ACTGCCATTCCTTAGGCCATGTG**

Hash pattern & $|P|$-substrings
- Compare hashes.
- Compare strings only when hashes match.
- $h(xb)$ easily computed from $h(ax)$, $b$

Best and worst cases?

---

**Intuition for Better Algorithms**

Pattern:  **CCATT**

Text:  **CCAGCCATTCCTTAGGCCATGTG**

After failing match at 1$^{\text{st}}$ position, what should we do?
Quick Overview of Two Algorithms

Knuth-Morris-Pratt, 1977
– Use previous intuition.
– Preprocessing builds shift table based upon pattern prefixes. $O(|P|)$
– Each text character compared once or twice. $O(|T|)$

Boyer-Moore, 1977 & Turbo Boyer-Moore, 1992
– Use previous intuition, along with similar heuristics. Complicated.
– Match from right end of pattern.
– Can often skip some text characters. $O(|T|)$, with $O\left(\frac{|T|}{|P|}\right)$ best case.

Finite Automata Matching Example

Pattern: CCATT

Text: CCAGCCATTCTCTAGGCGATGTG

$O(|P| \cdot |\Sigma|)$ to build
Regular Expressions

\((0 + 1)(00 + 01 + 10 + 11)^*\)

Syntax: \(\emptyset, \epsilon, a, rs, r + s, r^*\)
Regular Expression to Finite Automaton

\[ rs \]

\[ r + s \]

\[ r^* \]

Eliminating \(\varepsilon\)
Eliminating Non-Determinism

Each state is a set of the old states.

Can Minimize

States unreachable or equivalent? $O(|Q|^2)$
Finite Automaton Approach Summary

Preprocessing expensive for REs.
Matching is flexible and still linear.

Suffix Tree (Trie)

Text: CCAGAT

How many leaves, nodes, edges? Total space?
How to search for a pattern? Running time?

Each edge labeled with two indices.
**Require Suffixes to End at Leaves**

Text: CCAGAT

Reason: Simplicity. Distinguish nodes & leaves.
Example text that would break that?

---

**Forcing Suffixes to End at Leaves**

Text: CCAGAC$

---
How Would You Create the Tree?

Text: CCAGAC$

Suffix Tree Construction Example

Text: nananabanana$
Suffix Tree Construction Example

Text: nananabanana$

Match nanabanana$, nananabanana$: 4 comparisons
**Suffix Tree Construction Example**

Text: \text{nana\text{\textbackslash{}text{\textdollar}}}banana$

- \text{ana}$
- \text{nana}$
- \text{banana}$
- \text{nabanana}$

Match \text{ana\text{\textdollar}}}banana$, \text{ananabana\textdollar})): 3 redundant comps

Next ...

Match \text{nabanana\textdollar}}, \text{nabanana\textdollar})): 2 redundant comps

Match \text{abanana\textdollar}}, \text{abanana\textdollar})): 1 redundant comp

**First Algorithm Improvement: No Redundant Matching**

When inserting $aXY$, $nanabanana$

If we discover $aX$ is a prefix in tree, $nana$

Then $X$ will also be a prefix in tree, $ana$

So, don’t bother matching to verify.
Suffix Tree Construction Example

Text: nananabana$

Match nanabana$, nananabana$: 4 comparisons

Suffix Tree Construction Example

Text: nananabana$

No matching. Just form new node and adjust edge string indices.
Suffix Tree Construction Example

Text: \textit{nana}banana$

No matching. Just form new node and adjust edge string indices.

Suffix Tree Construction Example

Text: \textit{nana}banana$

No matching. Just form new node and adjust edge string indices.
Suffix Tree Construction Example

Text: nananabanana$

$ananabanana

anana is there, but it takes work to match & follow links.
Suffix Tree Construction Example

Text: `nananabanana$

`nana` is there, but it takes work to match & follow links.

Second Algorithm Improvement: Suffix Links

Text: `nananabanana$

Each internal node for $xA$ has pointer to node for $A$. 
Suffix Tree Construction Example

Text: nananab\textcolor{red}{ana}$

Have to follow links for match on first time.
Need to create suffix link for new node.

Suffix Tree Construction Example

Text: nananab\textcolor{red}{ana}$

Need to create suffix link for new node.
Use parent’s suffix link to get close.
**Suffix Tree Construction Example**

Text: `nananabana$`

![Image of Suffix Tree]

Already found node. No searching or matching.
Suffix Tree Construction Example

Text: nananabanana$

Follow suffix link.

Suffix Tree Construction Example

Text: nananabanana$

Follow suffix link.
Suffix Tree Construction Example

Text: nananabanan$ a 

banana$ na \$ na banana$

banana$ na \$ na banana$

banana$ na \$ na banana$

banana$ na \$ na banana$

Follow suffix link.

Done.

$O(c|T|)$, but roughly how big is $c$?
Almost Correct Analysis of Construction

Two indices: \( i \) & \( j \)

\( j \): Each increment takes \( O(1) \) time.
  – Just search for one more character.

\( i \): Each increment takes \( O(1) \) time.
  – Follow suffix link, or from root, get to suffix match.
  – Possibly split node & create suffix link.
  – Add one edge & leaf.

\( i, j \) each incremented \( n \) times \( \rightarrow O(n) \) total.

Creating & Following Suffix Links

When creating new node, need new suffix link.
Follow parent’s suffix link to get close, then search down.

Text: Z\( aB\)CDEF

\( k = \) index of next suffix link
Correct Analysis of Construction

Three indices: $i \backslash k \backslash j$ ($k$ not part of alg.)

$j$: Each increment takes $O(1)$ time.

$k$: Follow some number $l$ of links in $O(l)$ time.
In increments $k$ by at least $l$.

$i$: Each increment takes $O(1)$ time in addition to that considered for $k$.

$i, j, k$ each incremented at most $n$ times $\rightarrow O(n)$ total.

Some Applications of Suffix Trees

Search for fixed patterns
Search for regular expressions
Find longest common substrings, longest repeated substrings
Find most commonly repeated substrings
Find maximal palindromes
Find Lempel-Ziv decomposition (for text compression)

As used in

Bioinformatics
Data compression
Data clustering
<table>
<thead>
<tr>
<th>Supplementary Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Exact String Matching Algorithms</strong></td>
</tr>
<tr>
<td>&gt;30 algorithms, with animations</td>
</tr>
<tr>
<td><strong>Course on string matching (Biosequencing)</strong></td>
</tr>
<tr>
<td><strong>Wikipedia: Suffix trees</strong></td>
</tr>
<tr>
<td><strong>Tutorial on Suffix trees</strong></td>
</tr>
<tr>
<td><strong>Tutorial on Suffix trees with applet &amp; code</strong></td>
</tr>
<tr>
<td><strong>Notes on string matching and building suffix trees</strong></td>
</tr>
</tbody>
</table>

Suffix tree slides adapted from those by Guy Blelloch, CMU 15-853.