CS4311
Design and Analysis of Algorithms

Suffix Tree and Suffix Array
About this tutorial

- Introduce two data structures for text indexing problem:

  Suffix Tree and Suffix Array
Text Indexing

String Matching problem:

Given a text $T$ and a pattern $P$, how to locate all occurrences of $P$ in $T$?

- **KMP algorithm** can solve this in $O(|T| + |P|)$ time $\Rightarrow$ optimal
- In some applications, $T$ is very long, and given in advance, and we will search different patterns against it later
- E.g., $T$ = Human DNA, $P$ = gene
Text Indexing

Text Indexing problem:

Suppose a text $T$ is known. Can we build a data structure for $T$, such that for any pattern $P$ given later, we can find all occurrences of $P$ in $T$ quickly?

• The data structure is called an index of $T$
• Target: search better than $O(|T|+|P|)$??

4
Text Indexing

• Two main kinds of text indexes:

  **Word-Based**: (for texts formed by words)
  • Used by most text search engine
  • E.g., Inverted Files

  **Full-Text**: (for texts with no word boundaries)
  • Used in indexing DNA
  • E.g., Suffix Tree, Suffix Array
Suffix Tree

• Let $T[1..n]$ be a text with $n$ characters
  • we assume $T[n]$ is a unique character
• For any $j$, $T[j..n]$ is called a suffix of $T$
  $\Rightarrow$ $T$ has exactly $n$ suffixes
• Weiner (1973) and McCreight (1976) independently invented the suffix tree
  • a tree formed by putting all suffixes of $T$ together
Suffix Tree of acacaac#
Definition of a Suffix Tree

- **Suffix tree** is an **edge-labeled compact tree** (no degree-1 nodes) with **n leaves**
  - each leaf ↔ suffix
  - leaf label ↔ starting pos of suffix
  - If we traverse from root to leaf \( k \):
    - edge labels along path ↔ suffix \( T[k..n] \)
  - edge-label to each child starts with different character
Searching in a Suffix Tree

Theorem: If a pattern $P$ occurs at position $j$ in $T$, $P$ is a prefix of $T[j..n]$

This suggests the searching algorithm below:

- Start from root of the suffix tree
- Traverse the suffix tree using $P$

→ What we are doing is to match $P$ with all suffixes of $T$ at the same time
Searching in a Suffix Tree

Theorem: Pattern \( P \) occurs in \( T \) if and only if all chars of \( P \) are matched in the traversal of the searching algorithm.

Questions:
1. How to locate the occurrences?
2. What is the searching time?
   \( O(|P|+r) \) time, where \( r \) = \#occurrences
Space Usage

• There are $O(n)$ nodes and $O(n)$ edges in the suffix tree
  ➔ $O(n)$ space?

• Each edge needs to store its label, which can contain $O(n)$ chars
  ➔ In the worst-case, total $O(n^2)$ chars

• Can we reduce space usage?
Space Usage

Observation: Each edge label must be equal to some substring of $T$

Clever Idea:

1. Store $T$, and
2. Replace each edge label by 2 integers, telling which substring it is equal to

$\Rightarrow$ Total space: $O(n)$
Suffix Tree of acacaac#
Suffix Array

- Although suffix tree takes $O(n)$ space, the hidden constant is quite large
  -> around $40n$ to $60n$ bytes

- Manber and Myers (1990) simplified the suffix tree, and invented the suffix array
  - An array storing the suffixes of $T$ in the “dictionary” order
**Suffix Array**

- The suffix array $SA$ for $T$ has $n$ entries
- For any $j$, $SA[j]$ stores the $j^{th}$ smallest suffix, based on alphabetical order
- Theorem: If $P$ occurs in $T$, its occurrences correspond to consecutive region in $SA$
### Suffix Array

Searching $P$ takes $O(|P| \log n)$ time using binary search.

**Space:**

We can represent each suffix by its starting position $\Rightarrow O(n)$ space.

In practice, around $14n$ bytes.