**Suffix Trees**

- String Dictionaries
- Tries
- Suffix Trees
- Suffix Sorting

**String Dictionaries**

String dictionary problem. Let $S$ be a string of characters from alphabet $\Sigma$. Preprocess $S$ into data structure to support:

- `search(P)`: Return the starting positions of all occurrences of $P$ in $S$.

**Example.**

- $S = \text{yabbadabbado}$
- $\text{search(abba)} = \{1, 6\}$
Tries

- Tries [Fredkin 1960]. Retrieval. Store a set of strings in a rooted tree such that:
  - Each edge is labeled by a character. Edges to children of a node are sorted from left-to-right alphabetically.
  - Each root-to-leaf path represents a string in the set. (obtained by concatenating the labels of edges on the path).
  - Common prefixes share same path maximally.
  - Prefix free.
  - Append special character $ < \text{any character in } \Sigma$ to each string.

  $\implies$ Each leaf correspond to a unique string.

- Suffix trie.
  - Trie of all suffixes of a string.

\[ \begin{array}{c}
\text{Trie of all suffixes for yabbadabbado$} \\
\text{Space. } O(n^2) \\
\text{Preprocessing. } O(n^2)
\end{array} \]

Search(P):

- Process P from left-to-right while doing top-down search of trie:
  - At each node identify (unique) edge matching next character in P.
  - If no such edge, P is not a substring of S.
  - Report labels of all leaves below final node.

- Example.
  - search(abba) = \{1,6\}

- Time.
  - Top-down search +
  - Time for reporting leaves

  $\implies O(m + occ)$

\[ \begin{array}{c}
\text{Example.} \\
\text{search(abba) = \{1,6\}}
\end{array} \]

Theorem. We can solve the string dictionary problem in

- $O(n^2)$ space and preprocessing time.
- $O(m + occ)$ time for queries.
Suffix Trees

- String Dictionaries
- Tries
- Suffix Trees
- Suffix Sorting

The compact trie of all suffixes of S.
- Chains of nodes with single child are compacted into a single edge.

Suffix Trees

- Space.
  - Number of edges + space for edge labels
  - \( \Rightarrow O(n) \) space
- Preprocessing. \( O(sort(n,|\Sigma|)) \)
  - \( sort(n,|\Sigma|) \) = time to sort n characters from an alphabet \( \Sigma \).
  - Search(P): as before.

Suffix Trees

- Theorem. We can solve the string dictionary problem in
  - \( O(n) \) space and \( sort(n,|\Sigma|) \) preprocessing time.
  - \( O(m + occ) \) time for queries.
Suffix Trees

• Applications.
  • Approximate string matching problems
  • Compression schemes (Lempel-Ziv family, ...)
  • Repetitive string problems (palindromes, tandem repeats, ...)
  • Information retrieval problems (document retrieval, top-k retrieval, ...)
  • ...
Suffix Sorting

- **Suffix sorting.** Given string S of length n over alphabet Σ, compute the sorted lexicographic order of all suffixes of S.
- **Theorem [Kasai et al. 2001].** Given the sorted lexicographic order of suffixes of S, we can construct the suffix tree for S in linear time.
- How do we sort suffixes?

Sorting Small Universes

- Let X be a sequence of n integers from a universe U = {0, 1, ..., u-1}.
- How fast can we sort if the size of the universe is not too big?
  - U = {0, 1}?
  - U = {0, ..., n-1}?
  - U = {0, ..., n^3 - 1}?

Suffix Sorting

- **Goal.** Compute the lexicographic order of all suffixes of S fast.
- **Warm up.** Sorting small universes.
- **Solution in 3 steps.**
  - Solution 1: Radix sorting
  - Solution 2: Prefix doubling
  - Solution 3: Difference cover sampling

Sorting Small Universes

- **Radix Sort [Hollerith 1887].** Sort sequence X of n integers from U = {0, ..., n^3-1}.
  - Write each x ∈ X as a base n integer (x₁, x₂, x₃): x = x₁·n² + x₂·n + x₃
  - Sort X according to rightmost (least significant) digit
  - Sort X according to middle digit
  - Sort X according to leftmost (most significant) digit
  - Each sort should be **stable**.
  - Final result is the sorted sequence of X.

- **Positional number systems.** The base-n representation of x is x written in base n.
- **Example.**
  - (10)₁₀ = (1010)₂ = (1·2³ + 0·2² + 1·2¹ + 0·2⁰)
  - (107)₁₀ = (212)₇ = (2·7² + 1·7¹ + 2·7⁰)
n = 10, U = \{0, ..., n^3 - 1 = 999\}

- Theorem. We can sort n integers from a universe U = \{0, ..., n^3 - 1\} in O(n) time.
- Theorem. We can sort n integers from a universe U = \{0, ..., n^k - 1\} in O(kn) time.

- Larger universes?
- Theorem [Han and Thorup 2002]. We can sort n integers in O(n \log \log n) time or \(O(n (\log \log n)^{1/2})\) expected time.

Suffix Sorting

- Suffix sorting. Given string S of length n over alphabet \(\Sigma\), compute the sorted lexicographic order of all suffixes of S.
- For simplicity assume \(|\Sigma| = O(n)\)

Solution 1: Radix Sort

- Radix Sort.
  - Generate all suffixes (pad with $).  
    
    yabbadabbado$
    abbadabbado$
    abbabbadabado$
    abadabado$
    badabbado$
    bbadabado$
    
- Time. \(O(n^2)\)
Solution 2: Prefix Doubling

- Prefix doubling [Manber and Myers 1990]. Sort substrings (padded with $) of lengths 1, 2, 4, 8, ..., n. Each step uses radix sort on pair from previous step.

| 5 | Y  | 6 | 51 | Ya | 10 | 84 | Yabb  |
| 4 | a  | 1 | 12 | Ab | 1  | 13 | Abba  |
| 3 | b  | 2 | 22 | Bb | 6  | 42 | Bbad  |
| 2 | a  | 3 | 21 | Ba | 4  | 35 | Bada  |
| 1 | 2  | 3 | 21 | Ad | 2  | 21 | Adab  |
| Y | 5 | 51 | Da | 7  | 54 | Dabb  |
| 1 | a  | 1 | 12 | Ab | 1  | 13 | Abba  |
| 2 | b  | 4 | 22 | Bb | 6  | 42 | Bbad  |
| 2 | a  | 3 | 21 | Ba | 5  | 35 | Bada  |
| 1 | 2  | 3 | 21 | Ad | 3  | 27 | Adob  |
| 3 | d  | 6 | 34 | Do | 8  | 60 | Dob$  |
| 4 | o  | 7 | 40 | Do$| 9  | 70 | Dob$  |
| 0 | 3  | 0 | 00 | D$ | 0  | 00 | D$   |

- Time. O(n log n)

Solution 3: Difference Cover Sampling

- DC3 Algorithm [Karkkainen et al. 2003]. Sort suffixes in three steps:
  - Step 1. Sort sample suffixes.
    - Sample all suffixes starting at positions $i = 1 \mod 3$ and $i = 2 \mod 3$.
    - Recursively sort sample suffixes.
  - Step 2. Sort non-sample suffixes.
    - Sort the remaining suffixes (starting at positions $i = 0 \mod 3$).
  - Step 3. Merge.
    - Merge sample and non-sample suffixes.

Step 1: Sort Sample Suffixes

Step 1: Sort Sample Suffixes
Step 1: Sort Sample Suffixes

Step 2: Sort Non-Sample Suffixes
### Step 3: Merge

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>d</td>
<td>a</td>
<td>d</td>
</tr>
<tr>
<td>3</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>4</td>
<td>y</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>d</td>
<td>o</td>
<td>d</td>
<td>o</td>
</tr>
<tr>
<td>5</td>
<td>d</td>
<td>b</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
</tr>
<tr>
<td>6</td>
<td>d</td>
<td>b</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
</tr>
<tr>
<td>7</td>
<td>o</td>
<td>b</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>8</td>
<td>s</td>
<td>b</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
</tr>
</tbody>
</table>
Solution 3: Difference Cover Sampling

- DC3 Algorithm. Sort suffixes in three steps:
  - Step 1. Sort sample suffixes.
    - Sample all suffixes starting at positions $i = 1 \mod 3$ and $i = 2 \mod 3$. $O(n)$
    - Recursively sort sample suffixes. $T(2n/3)$
  - Step 2. Sort non-sample suffixes.
    - Sort the remaining suffixes (starting at positions $i = 0 \mod 3$). $O(n)$
  - Step 3. Merge.
    - Merge sample and non-sample suffixes. $O(n)$
  - $T(n) =$ time to suffix sort a string of length $n$ over alphabet of size $n$

- Time. $T(n) = T(2n/3) + O(n) = O(n)$

Solution 3: Difference Cover Sampling

- Theorem. We can suffix sort a string of length $n$ over alphabet $\Sigma$ of size $n$ in time $O(n)$.
- Larger alphabets?
- Theorem. We can suffix sort a string of length $n$ over alphabet $\Sigma$ $O(sort(n, |\Sigma|))$ time.
- Bound is optimal.

Suffix Trees

- String Dictionaries
- Tries
- Suffix Trees
- Suffix Sorting