Goals:
1. Develop linear time algorithms for suffix trees, suffix arrays
2. Generalize to linear time algorithm for BWT.

**Suffix Tries (Aho-Corasick Algorithm)**

- trie from "retrieval"
- useful data structure in other applications (text indexing)

**Example:**

```
1. AGATT$
2. GAAT$
3. AAT$
4. AT$
5. T$
6. $
```

Time complexity: $O(n^2)$ to build, $O(k)$ to query (query length $k$)

Space complexity: $O(n^2)$ (consider string $A^n C^n$)

Answer a query: traverse edge for each character in query ("threading"). Return true at leaf, false otherwise.

**Online algorithm for construction (Ukkonen):**

- Add state $I$ for empty suffix. Def. transition function $g$ (edges).
- Def. suffix function $f$ (links node to its suffix in the trie).
- Def. state $T$ for entire string.
Algorithm

\[ T \leftarrow \]

while \( g(\text{state}, s) \) undefined: [no edge]

add state \( r' \)

add edge \( g(\text{state}, s) = r' \) [extend longest suffix]

\[ r'' \leftarrow r' \]

\[ r \leftarrow f(r) \]

create suffix link \( f(g(\text{state}, s)) = g(\text{state}, s) \)

create suffix link \( f(r') = g(\text{state}, s) \)

Example:

\[ \Sigma \]

any char. in alphabet \( \Sigma \)

Suffix trees (Ukkonen's algorithm)

- compressed suffix tree: label edges with substrings (intervals)
- Adapt \( O(n) \) algorithm; \( O(n) \) space

Example:

\[ \Sigma \]

\[ \Sigma \]

\[ \Sigma \]

\[ \Sigma \]
Intuition for algorithm:

Define path from $t$ to root as boundary path. Define active pt: as lowest non-leaf on boundary path. Def. end point as lowest non-leaf with $g(s, t)$ defined.

Lemma: The alg. expands the path for states below the active point and creates new paths for states between active and end pts.

Consequences:
1. represent edges to leaves as open intervals $(k, \infty)$. No work to extend the path.
2. use suffix links to find where to split edges and create new paths.

Claim: This takes $O(n)$ time.

Proof sketch: Traversing to boundary path takes $O(n)$ time. Taking a suffix link reduces depth; taking a transition increases depth by 1. At each step we increase left endpoint & right endpoint → only do work $n$ times.
Sufix array (SA-IS)
- only store index of suffixes in lex. order
- closely related to BWT.
- Linear space, but smaller constant than suffix tree.

Example:

<table>
<thead>
<tr>
<th>6</th>
<th>$ \leftarrow \text{assume sentinel } &lt; \Sigma $</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>A A T $</td>
</tr>
<tr>
<td>1</td>
<td>A G A A T $</td>
</tr>
<tr>
<td>4</td>
<td>A T $</td>
</tr>
<tr>
<td>2</td>
<td>G A A T $</td>
</tr>
<tr>
<td>5</td>
<td>T $</td>
</tr>
</tbody>
</table>

Algorithm outline:

Divide: choose a subset of suffixes.
Conquer: build a suffix array on the subset.
Combine: reconstruct original suffix array (Induced Sorting)

Divide: classify suffixes.

\[
\begin{cases}
  S\text{-type}, i = n \text{ on } t_i = t_i+1 \text{ on } (t_i = t_{i+1} \text{ and } t_{i+1} \text{ is } S\text{-type}) \\
  L\text{-type} \text{ otherwise}
\end{cases}
\]

\(S^*\text{-type}, \text{ if } t_i \text{ is } S\text{-type} \text{ and } t_{i-1} \text{ is } L\text{-type}\)

Example:

<table>
<thead>
<tr>
<th>A G A A T $</th>
</tr>
</thead>
<tbody>
<tr>
<td>S L S S L S</td>
</tr>
<tr>
<td>(S^<em>) (S^</em>)</td>
</tr>
</tbody>
</table>
Conquer: build suffix array on $s^\ast$ suffixes

6 $\$
3 A A T $\$

Combine: suffixes appear in buckets keyed by first character.
All L-type suffixes appear before S-type suffixes.

1) Initialize:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>3</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

2) Add L-type suffixes:

for $i = 1$ to $n$
if $A_i > 0$ and $T_{A_i-1}$ is L-type:
add $i$ to head of bucket.

Example:

6 | 3 | -1| -1| -1| -1|
6 | 3 | -1| -1| -1| 5|
6 | 3 | -1| -1| 2 | 5|

3) Add S-type suffixes:

for $i = n$ to 1
if $A_i > 0$ and $T_{A_i-1}$ is S-type
add $i$ to tail of bucket
Example:

\[
\begin{array}{c|c|c|c|c}
6 & 3 & -1 & 2 & 5 \\
\uparrow & & & & \\
6 & 3 & -1 & 4 & 2 & 5 \\
\uparrow & & & & \\
6 & 3 & 1 & 4 & 2 & 5
\end{array}
\]

Remarks: Naive BWT takes \(O(n^2 \log n)\) time. Can modify this algorithm to do BWT in \(O(n)\) time.

Time complexity: \(T(n) = T\left(\frac{n}{2}\right) + O(n) = O(n)\) [Master theorem]