Lecture 9 - Trays, Mismatches and Wild cards, level ancestor

Last Time: Text T of length n, Pattern P of length m, both over alphabet \( \Sigma \)

### Suffix Trees (ST)
- \( O(n|\Sigma|) \) space or \( O(n) \) space
- \( O(m) \) query or \( O(m \log |\Sigma|) \) query

### Suffix Arrays (SA)
- \( O(n) \) space
- \( O(m + \log n) \) query

\[ T = \text{abababababa} \]
\[ \text{abababababa} \]

\[ 0123456789 \]

Idea: tree decomposition; we already saw heavy path, prefered path decompositions, we will also see separator decomposition (\( \leq \frac{1}{2} \)) and ladder decomposition today.

**SA:** 9 8 6 4 2 0 7 5 3 1 = leaves of ST

- Internal nodes of ST correspond to an interval of SA
- Time to search on interval \( I \) is \( O(m + \log |I|) \).

**Definition:** \( \varepsilon \)-node is:
- (1) has at least \( |\Sigma| \) leaves in its subtree.
- (2) all its children do not.

# of leaves in subtree of a \( \varepsilon \)-node \( \leq |\Sigma|^2 \) so time to search in SA interval of a \( \varepsilon \)-node is \( O(m + \log |I|) \)

- This takes care of \( \varepsilon \)-nodes
Remaining internal nodes partitioned into:

1. Branching-ε-node: has at least two children with subtree ≥ |ε|
2. Others

| Branching-ε-nodes is \(O(n/|ε|)\) so we can store an array of size |ε| in everyone.

The remaining nodes:

![Diagram showing branching and ε-nodes]

Navigation:
1. ε-node: jump to SA.
3. Others: look at single character, walk to this child or jump to SA.

Search time: \(O(n) + O(n + \log |ε|) = O(n + \log |ε|)\) to walk Branching-ε-nodes and single characters in interval and search SA

Space: \(O(n)\)
Approximate String Matching

Given a text $T$, find occurrences of a query pattern $p$ in $T$ with error $k$

- Hamming Distance: # of character mismatches
- Edit distance: # of edits (insertions, deletions, mismatches) needed for an exact match

"Online"

- $P$ & $T$ are given together.
- Construct suffix tree + LCA on $P#T$ \( \Rightarrow \) space \( O(|T|) \)
- For every location in $T$
  - Use $K$ LCA queries to check if $p$ occurs in this location.

<table>
<thead>
<tr>
<th>LCA(1,i)</th>
<th>1 2 3 4 5 6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ABCDEFG</td>
</tr>
<tr>
<td></td>
<td>ABXDEZ</td>
</tr>
</tbody>
</table>

Time: \( O(|T|+K) \)

"Offline"

- $T$ is given in advance, $p$ is given online.

- Best known Bounds: \cite{cole}

  Space and preprocessing: \( O(|T| (c \log |T|)^K / K!) \)

  Query time: \( O(k^1 + (c \log |T|)^K (\log \log |T| + \sum_{i=1}^{K} \# occurrences)) \) \( \frac{1}{K!} \)

  \( \text{only for edit distance} \)
We focus on subproblem of searching with wildcards:
- P contains at most K "don't care" characters (?? wildcards)
- Find "exact matches" (?? matches anything)

Solution: \( O(l_T l_g K l_T) \) space and preprocessing
\( O(2^{K l_g l_T} + |P| \# \text{occurrences}) \) for query

**Simple solution** \( \exists |P| \) query
- walk down suffix tree of \( T \) as usual
- upon \( ? \) in \( P \) branch \|\| \# ways

**Better solution** \( 2^K |P| \) query
- Heavy-Light decomposition on \( 	ext{suffix tree of } T \)

Each node in primary tree stores secondary tree on the union of light subtrees hanging off node, excluding first character
- recurs \( K \) times \( \implies \) \( K \# \text{ levels of trees} \)
- light depth \( \leq l_g l_T \)
  \[ \Rightarrow \] each leaf appears in \( \leq (K - 1)! \) trees
  \[ \Rightarrow \] \( O(l_T l_g K l_T) \) space & preprocessing

\( 2^{K l_g l_T} / |P| \) solution uses "level ancestor" data structure.

Determine in \( l_g l_T \) time on which of the two edges we branch.
Level Ancestor Problem:

- Preprocess static tree
- level ancestor query: find kth ancestor of node v.

[Berkman & Vishkin - TCAS 94, Dietz & Mehlhorn - ICALP 99, Ablstrap & Holm - ICALP 98, Bender & Farach-Colton - TCS 04]

Solutions:

1. **Lookup table:** on all possible queries
   - O(n^2) space
   - O(1) query

2. **Jump pointers:** (skip list) each node stores 1st, 2nd, kth, gth... ancestor
   - O(n log n) space
   - O(lgn) query

3. **Long path decomposition:** decompose tree by longest paths; store not the same as heavy path. Each path as an array + parent pointer
   - O(n) space
   - Query might still visit O(lg n) paths...
   - Notice: node of height h is on path of length ≥h. So:

4. **Ladder decomposition:** extend path of length k up by k levels.
   - Still O(n) space
   - Rh = height(v) = h ⇒ height(top node in v's ladder) ≥Rh
   - query: as above, but using ladders
     - each step at least doubles height ⇒ O(lgn) query

5. **Combine jump pointers & ladder decom.**
   - query: O(1). 1 jump pointer ⇒ height(v') > k/2
     - 1 ladder step ⇒ ladder(v') extends > k/2 above
     - => contains kth ancestor
   - Space: O(n lgn)
jump-pointer tuning: jump-pointers only at leaves
- start queries at leaves
- depth-d ancestor of u = depth-d ancestor of leaf(v)
- $O(n + L \lg n)$ space, where $L = \# \text{leaves}$
  
  ladder  jump
decomp pointers

ART decomposition:

$O(n \lg n)$ leaves $\Rightarrow$ using (6) we get $O(n)$ space
less than $\frac{n}{4} \lg n$ nodes

$\Rightarrow \# \text{distinct trees on } \frac{n}{4} \lg n \text{ nodes}$

$= C_{\frac{n}{4} \lg n} \leq 4 \frac{n}{4} \lg n = \sqrt{n}$

- lookup table for every possible bottom tree $\Rightarrow O(n \sqrt{n})$ space
- query: look in bottom tree, else get parent(root), query top
  $\Rightarrow O(1)$ query

* In our "don't cares" data structure we need ancestors in a
  compressed trie $\Rightarrow$ weights on the edges
  - total weight along any path $\leq n