Topics

- Keyword trees
- Suffix trees
- Constructing suffix trees

Exact pattern matching recap

- Text T
  - $|T| = m$
  - Ex: "I like bananas and potatoes"
- Pattern P
  - $|P| = n$
  - Ex: "banana"
- Find all occurrences of P in T
  - Naïve: $O(mn)$
  - Possible: $O(m + n)$

Matching sets of patterns

- Text T
  - $|T| = m$
  - Ex: "I like bananas and potatoes"
- Patterns $P = \{P_1, P_2, \ldots, P_z\}$
  - $|P| = \sum |P_i| = n$
  - Ex: {"banana", "potato", "pottery", "poetry", "other", "theater", "tattoo"}
- Find all occurrences of any $P_i \in P$ in T
  - z separate linear time searches: $O(n + zm)$
  - Possible: $O(n + m + k)$, k is the number of occurrences; How?

Keyword trees encode pattern sets

$P = \{"banana", "potato", "pottery", "poetry", "other", "theater", "tattoo"\}$

Keyword trees can be built in $O(n)$

$P = \{"banana", "potato", "pottery", "poetry", "other", "theater", "tattoo"\}$
- Partial tree $K_i$
  - Encodes patterns $\{P_1, \ldots, P_i\}$

1
Naïve matching takes $O(nm)$

NaiveKeywordSearch($T, P$):
1. $K =$ BuildKeywordTree($P$)
2. for $i$ in len($T$):
   1. 

Ex: “I like bananas and potatoes”

Speeding up keyword tree searches

- Possible in $O(n + m + k)$
  - $O(n)$: Keyword tree construction
  - $O(m)$: Scan each position in $T$ once (constant)

- Solution
  - Identify substrings whose suffix is prefix of other pattern
  
  Ex: (“other”, “theater”), $T =$ “otheater”

Failure functions for keyword trees

- Node label $L(v)$
  - Concatenation of letters from root to $v$
  - Ex: $L(w) =$ “oth”

- $lp(v)$
  - Length of suffix of $v$ that is prefix of other pattern
  - Ex: $lp(w) = 2$
  - There is a unique node corresponding to this suffix; Why?

- Use link to prefix node to track failures
  - Failure link for $w$?

Failure links speed up keyword search

Keyword trees appropriate with fixed $P$

- Pattern set $P$ known
  - Preprocessing of $P$ to keyword tree
  - Speeds up search in unknown texts

- What if $T$ is known but $P$ is unknown?

Reverse keyword trees speed search

- Index words in text
  - Ex: “I like bananas and potatoes”

- Tree creation: $O(m)$

- Amortized search: $O(n)$

- Limitations?
### Topics
- Keyword trees
- Suffix trees
- Constructing suffix trees

### Suffix trees encode all suffixes in T
- **Text** $T = t_1t_2\ldots t_m$
  - Suffix set $S = \{t_1t_2\ldots t_m, t_2t_3\ldots t_m, \ldots, t_{m-1}t_m, t_m\}$
  - Ex: $T = \text{gctgca}$, $S = \{\text{gctgca, ctgca, tgc, gca, ca, a}\}$
- **Suffix tree of $T$**
  - Rooted directed tree
  - $m$ leaves numbered 1 to $m$
  - Internal nodes (not root) have at least 2 children
  - Edges labeled with substring of $T$
  - All edges from same node start with different letters
  - For any leaf $i$, the path from root to leaf spells the $S_i$ suffix

### Suffix tree example
- $T = \text{gctgca}$

### Searching with suffix trees
Find all occurrences of "gca", "c", and "gct" in "gctgca"

### Suffix tree search algorithm

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>$S = \text{BuildSuffixTree}(T)$</td>
<td>$O(</td>
</tr>
<tr>
<td>2.</td>
<td>Use $P$ to navigate $S$</td>
<td>$O(_)$</td>
</tr>
<tr>
<td>3. if end of $P$:</td>
<td></td>
<td>$O(_)$</td>
</tr>
</tbody>
</table>
| 4. else: | | $O(\_)$

### Handling strings with overlapping prefix/suffixes
- Append unique termination symbol
  - Ex: "$\$"
  - $T = \text{gctgca}\$"
Generalized suffix trees

- Leaves encode string ID and position (ID, pos)
- Applications
  - Find all occurrences of P in T (Ex: P = "gc")
  - Substring problem: Given S, find all T ∈ T such that S is substring of T
  - Find longest common substring of T₁ and T₂ (Ex: DNA contamination)

Topics

- Keyword trees
- Suffix trees
- Constructing suffix trees

Naïve suffix tree construction

- NaïveBuildSuffixTree(T): Running time: O(__)
  1. S = rootNode
  2. for i in len(T):
     1. Add suffix T[i:] + "$" to S

Ex: T = "gcgcac" → implicit suffix tree, I₅

Adding suffixes to growing tree

1. Use T[i:] + "$" to navigate S
2. Split edge above mismatch

Ex: T = "gcgcac", T[4:] + "$" = "gcac$"

Going from quadratic to linear construction

- Naïve algorithm scans last letter m² times
- Linear: constant #evaluations of each letter
  - Build suffix tree by extending suffixes from start
    - Given suffix tree for prefix T[1:i]
    - Build suffix tree for prefix T[1:i+1] by extending existing suffixes
    - Ex: T[1:i] = "gcgc", T[1:i+1] = "gcgcac"

High level suffix extension (Ukkonen) algorithm

- HighLevelUkkonen(T):
  1. Construct tree I₁
  2. for i in range(|T| - 1):
     1. for j in range(i + 1):
        1. Find end of path labeled T[j:i] in I₁
        2. Extend that path by T[j] if needed
  3. O(|T|)

Total: O(__)
Three suffix extension rules

- $T[j:i] = \beta$ is suffix of $T[1:i]$
- End of $\beta$ found, extend such that $T[j+1]$ is in tree
  1. $\beta$ ends at leaf
     - Append $T[j+1]$ to label
  2. No path from end of $\beta$ starts with $T[j+1]$, but path continues
     - Create new leaf edge from end of $\beta$ and label with $T[j+1]$
     - Number leaf with $j$
  3. Tree contains $T[j+1]$
     - Do nothing

T = "gctgcg"

Implicit suffix tree, $I_5$

Speeding up extensions: Suffix links

- Where to continue next extension?

Definition (Suffix link):

- $x\alpha$ is string
  - $x$ single character
  - $\alpha$ substring (possibly empty)
- For internal node $v$ labeled $x\alpha$, if another node $s(v)$ labeled $\alpha$
  - Pointer from $v$ to $s(v)$
  - Suffix link

Ex: Extending with "a"

Suffix link properties (Lemma 6.1.1)

- New internal node labeled $xa$ in extension $j$
  - Either path $\alpha$ ends in internal node
  - Or internal node labeled $\alpha$ created in extension $j + 1$ in same phase

Proof:

- Extension rule 2 applies
  - In ext. $j$, path $xa$ continued with character other than $T[j+1]$ (ex: $c$; that is, tree contains path $xac$ and $ac$)
  - Two cases
    1. $\alpha$ continues with only $c$
      - Extension rule 2 creates node $s(v)$ at end of $\alpha$ (in ext. $j + 1$)
    2. $\alpha$ continues with additional characters
      - Node $s(v)$ already exists

Suffix link properties (Corollary 6.1.1)

Any newly created internal node will have an outgoing suffix link by the end of next extension

- Tree $I_j$ has no internal nodes
- Phase $i + 1$:
  - When new internal node $v$ created, correct suffix link $s(v)$ found in next extension (Lemma 6.1.1).
  - No internal node in last extension (single character)
  - All internal nodes known by end of phase so $I_{i+1}$ has all its suffix links

- If internal node $v$ has path-label $xa$, then there is a node $s(v)$ with path label $\alpha$

Following suffix links

Ext. 1: always extends leaf 1
- Pointer gives O(1)

Ext. $j$:
- Go up to closest node (internal $v$, or root), path $\gamma$
- Internal: Go to $s(v)$; follow $\gamma$; extend
- Root ($\gamma = x\alpha$): Follow path $\alpha$ and extend

Single extension algorithm (SEA) ($j \geq 2$)

1. Find first node $v$ above end of $T[j−1:i]$
   - At most one edge
   - Either internal node (with suffix link)
   - Or root
   - $\gamma$ is string between $v$ and end of $T[j−1:i]$
2. $v$ internal?
   - Yes: go to $s(v)$, walk down following $\gamma$
   - No (root): follow path $T[j:i]$ from root
3. Extend with $T[j+1]$ using extension rules
4. If extension $j−1$ created internal node $w$
   - String $\alpha$ must end in node $s(w)$ (Lemma 6.1.1)
   - Create suffix link from $w$ to $s(w)$
Suffix link in itself does not give speedup

- γ – backtrack and γ – traversal takes O(|γ|)
- Skip/count trick reduces traversal to O(|nodes|)
  - One edge e from s(v) must start with γ
  - Two possibilities (|e| = g'; |γ| = g)
    1. g' > g: skip to g on edge
    2. g' <= g:
      1. skip to node
      2. g = g' - g
      3. h = h + g'
      4. find edge corresponding to character γ[h]

Skip/count trick gives O(m^2) bound

- Def. Node depth of u: nodes from root to u
- Given suffix link (v, s(v))
  - Node depth of v is at most 1 greater than node depth of s(v)
  - All ancestors of v (except root) have suffix links to distinct ancestors of s(v)
  - Only extra depth can come from node labeled x
- Bound # node depth decreases to bound # edge traversals
  - Max node depth: m
  - Max decreases of node depth in phase: 2m
    - Up-walk decrease by 1
    - Suffix-link traversal decrease by at most 1
    - Down-walk increase by at least 1
  - Edge traversals bounded by 3m
  - Down-walking is O(m)

Edge-label compression

- Use index pairs to label edges
  - Start and end in T
  - Ex: T = "gctgcg"
- Extension rules (phase i + 1):
  1. Change label on leaf edge from (p, i) to (p, i + 1)
  2. Label new edge with (i + 1, i + 1)
- Reduces space from O(m^2) to O(m)

Observation 1: Rule 3 applies to all following extensions

- Rule 3: Tree contains T[j:i+1]
  - Tree also contains all T[j+1:i+1],... T[i+1:i+1]
- Trick 2:
  - End phase when rule 3 first applies
  - Remaining extensions done implicitly

Observation 2: Once a leaf, always a leaf

- Once leaf created (and labeled j), that leaf will remain
  - Rule 1 will always apply to extension
- For any phase i
  - Initial series of extensions with rule 1 or 2
    - j last such extension
    - j <= j_{i+1}
- Trick 3:
  - Use "current end" symbol e on leaves
  - Label new leaf edges (i + 1, e)
  - Extensions 1 to j, takes O(1)
  - Explicit extensions from j + 1

Single phase algorithm (SPA)

1. e = i + 1
2. for j in range(j_i+1, i+1):
   1. Use SingleExtensionAlgorithm for explicit extensions
   2. if “rule 3” applies:
      1. break
   3. j_{i+1} = j - 1
Ukkonen's algorithm runs in $O(m)$

- Implicit extensions is $O(1)$ per phase
  - $O(\_\_\_)$ overall
- At most $2m$ explicit extensions
  - Current extension $j^*$
  - $j^*$ never decreases
  - Can remain identical between two phases
- Node skips of explicit extensions bounded by $O(m)$
  - Node depth does not change between explicit extensions
  - Bound on depth decrease gives $O(m)$ skips

Creating true suffix tree

- Append $\$\$ to $T$ and run algorithm
  - Ex: $T = “gctgc”$
  - Replace $e$ with $m$: $O(m)$ tree traversal

Summary

- Ukkonen's construction builds suffix tree in $O(m)$
- Scans from start to end
- Uses properties of tree and problem to speed construction
  - Suffix links
  - Skip/count
  - Implicit extensions of suffixes/prefixes (Rule 3)
  - Implicit extensions of leaves (Rule 1)