DSC 190

DATA STRUCTURES & ALGORITHMS

Suffix Tries and Suffix Trees

Last Time

- We have seen **tries**.
- They provide for very fast prefix searches.
- But we don’t do a lot of prefix searches...

Today’s Lecture

- A way of using tries for solving much more interesting problems.

String Matching
(Substring Search)

- **Given**: a string, s, and a pattern string p
- **Determine**: all locations of p in s
- Example:

$$s = \text{"GATTACATACG"} \qquad p = \text{"TAC"}$$

Recall

- We've solved this with Rabin-Karp in $\Theta(|s| + |p|)$ expected time.
- What if we want to do *many* searches?
- Let's build a data structure for fast substring search.

Suffixes

- A **suffix** `p` of a string `s` is a contiguous slice of the form `s[t:]`, for some t.
- Examples:
  - `"ing"` is a suffix of `"testing"`
  - `"ting"` is a suffix of `"testing"`
  - `"di"` is **not** a suffix of `"san diego"`

A Very Important Observation

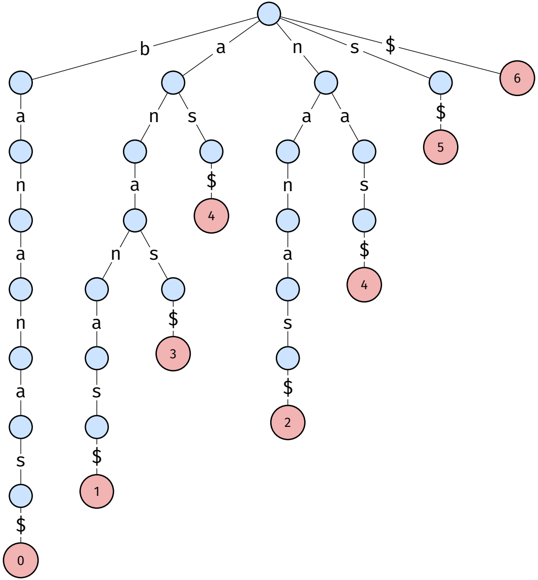
- *w* is a substring of *s* if and only if *w* is a **prefix** of some **suffix** of *s*.

```
s = "california"
p_1 = "ifo"
p_2 = "lif"
p_3 = "flurb"
```

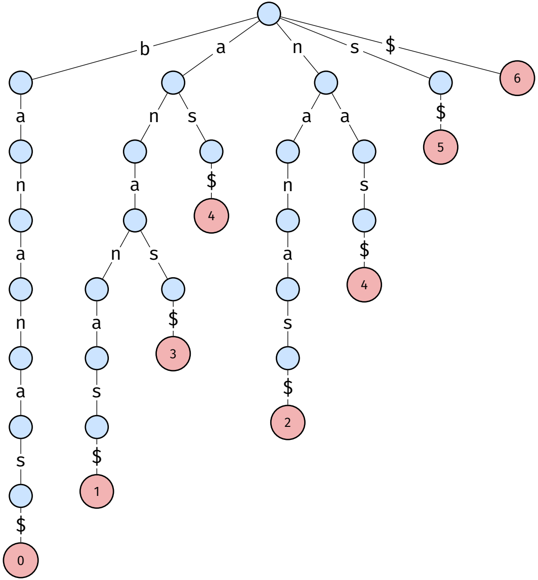
```
"california"
"alifornia"
"lifornia"
"ifornia"
"fornia"
"ornia"
"rnia"
"nia"
"ia"
"a"
""
```

Idea

- Last time: can do fast prefix search with `trie`.
- Idea for fast repeated substring search of s:
  - Keep track track of all suffixes of s in a trie.
  - Given a search pattern p, look up p in trie.
- A trie containing all suffixes of s is a **suffix trie** for s.



```
s[0:]: "bananas"
s[1:]: "ananas"
s[2:]: "nanas"
s[3:]: "anas"
s[4:]: "nas"
s[5:]: "as"
s[6:]: "s"
s[7:]: ""
```

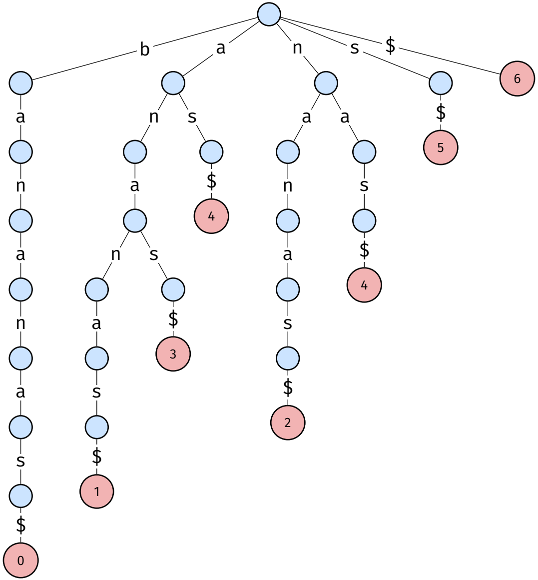


Substring Search

- Given pattern p, walk down suffix trie.
- If we fall off, return [ ].
- Else, do a DFS of that subtrie. Each leaf is a match.
- Time complexity: $\Theta(|p| + k)$, where k is number of nodes in the subtrie.

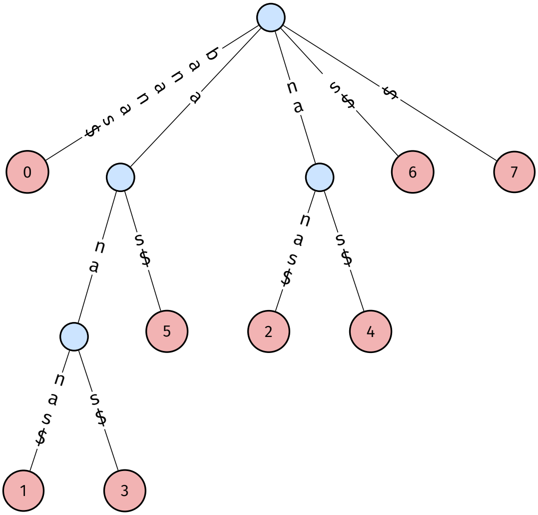
Problems

- In the worst case, a suffix tree for s has $\Theta(|s|^2)$ nodes.
  - Suffixes of length $|s|$, $|s| - 1$, $|s| - 2$, ...,
- So substring search can take $\Theta(|s|^2)$ time.
- Construction therefore takes $\Omega(|s|^2)$, too.
  - Naïve algorithm takes $\Theta(|s|^2)$ time.
- Takes $\Theta(|s|^2)$ storage.



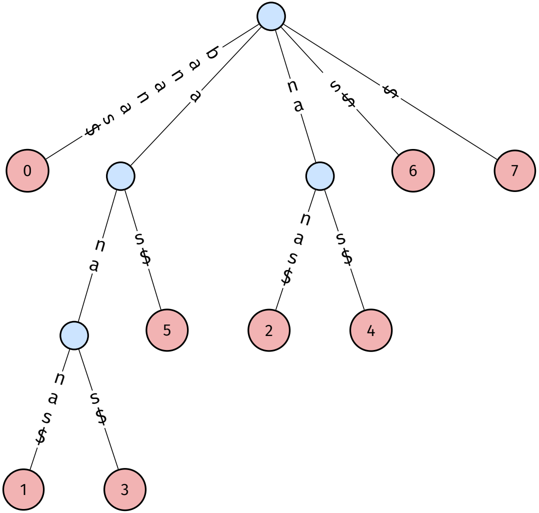
Silly Nodes

- A **silly node** has one child.
- Fix: “Collapse” silly nodes?



"Collapsing" Silly Nodes

```
s[0:]: "bananas"
s[1:]: "ananas"
s[2:]: "nanas"
s[3:]: "anas"
s[4:]: "nas"
s[5:]: "as"
s[6:]: "s"
s[7:]: ""
```



Suffix Trees

- This is a **suffix tree**[a].
- Internal nodes represent **branching words**.
- Leaf nodes represent **suffixes**.
- Leafs are labeled by starting index of suffix.

---

[a] Not to be confused with a **suffix trie**.

Branching Words

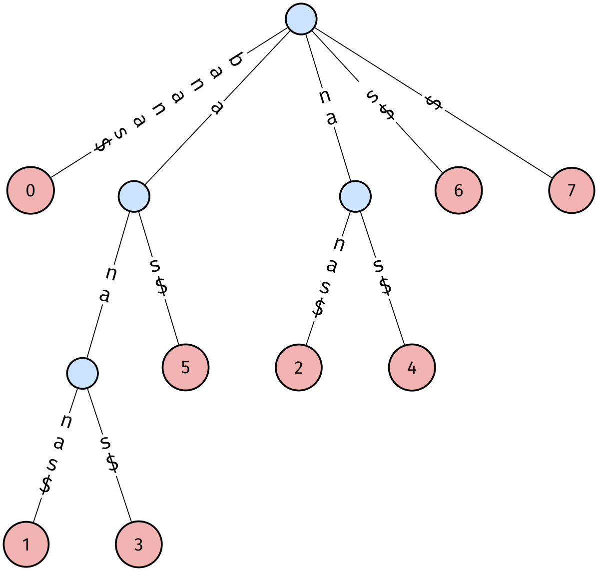
- Suppose s' is a substring of s.
- An **extension** of s' is a substring of s of the form:

$$s' + \text{one more character}$$

- Example: `s = "bananas"`,
  - `"ana"` → {`"anas"`, `"anan"`}
  - `"a"` → {`"an"`, `"as"`}
  - `"ban"` → {`"bana"`}

Branching Words

- A **branching word** is a substring of s with more than one extension.
- Example: `s = "bananas"`,
  - `"ana"` $\rightarrow$ {`"anas"`, `"anan"`} (**yes**)
  - `"a"` $\rightarrow$ {`"an"`, `"as"`} (**yes**)
  - `"ban"` $\rightarrow$ {`"bana"`} (**no**)



Branching Words

- "a", "ana", "na" are branching words in "bananas".
- Internal nodes of the suffix tree represent branching words.

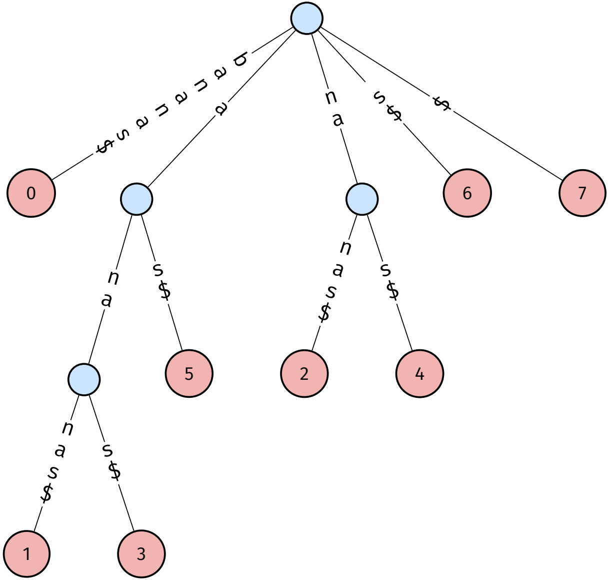
Number of Branching Words

- There are $O(|s|)$ branching words.

- Proof:
  - Remove all of the internal nodes (branching words).
  - Now there are $|s|$ forests (one for each suffix).
  - Add the internal nodes back, one at a time.
  - Each addition reduces number of forests by one.
  - After adding $|s| - 1$ internal nodes, forest has one tree.
  - Therefore there are at most $|s| - 1$ internal nodes.

Size of Suffix Trees

- A suffix tree for any string s has $\Theta(|s|)$ nodes.



Substring Search

- Given pattern p, walk down suffix trie.
- If we fall off, return [ ].
- Else, do a DFS of that subtree. Each leaf is a match.
- Time complexity: $\Theta(|p| + z)$, where z is number of matches.

Naïve Construction Algorithm

- First, build a suffix trie in $\Omega(|s|^2)$ time in worst case.
  - Loop through the $|s|$ suffixes, insert each into trie.
- Then “collapse” silly nodes.
- Takes $\Omega(|s|^2)$ time. **Bad**.

Faster Construction

- There are (surprisingly) $O(|s|)$ algorithms for constructing suffix trees.
- For instance, Ukkonen's Algorithm.

Single Substring Search

Rabin-Karp

- Rolling hash of window.
- $\Theta(|s| + |p|)$ time.

Suffix Tree

- Construct suffix tree; $\Theta(|s|)$ time.
- Search it; $\Theta(|p| + z)$ time.
- Total: $\Theta(|s| + |p|)$, since $z = O(|s|)$.

Multiple Substring Search

Multiple searches of s with different patterns, $p_1, p_2, \ldots$

Rabin-Karp

- First search: $\Theta(|s| + |p_1|)$.
- Second search: $\Theta(|s| + |p_2|)$.

Suffix Tree

- Construct suffix tree; $\Theta(|s|)$ time.
- First search: $\Theta(|p_1| + z_1)$ time.
- Second search: $\Theta(|p_2| + z_2)$ time.
- Typically $z \ll |s|$

Suffix Trees

- Many other string problems can be solved efficiently with suffix trees!

DSC 190

DATA STRUCTURES & ALGORITHMS

Longest Repeated Substring

Repeating Substrings

- A substring of s is **repeated** if it occurs more than once.
- Example: `s = "bananas"`.
  - `"na"`
  - `"ana"`

Repeating Substrings in Genomics

- A repeated substring in a DNA sequence is interesting.
- It's a “building block” of that gene.

GATTACAGTAGCGATGATTACAGGTGATTACA

GATTACAGTAGCGAT**GATTACA**GGT**GATTACA**

Longest Repeated Substrings

- The longer a repeated substring, the more interesting.
- **Given**: a string, s.
- **Find**: a repeated substring with longest length.

Brute Force

- Keep a dictionary of substring counts.
- Loop a window of size 1 over s.
- Loop a window of size 2 over s.
- Loop a window of size 3 over s, etc.
- $\Theta(|s|^2)$ time.

Suffix Trees

- We'll do this in $\Theta(|s|)$ time with a suffix tree.

Branching Words & Repeated Substrings

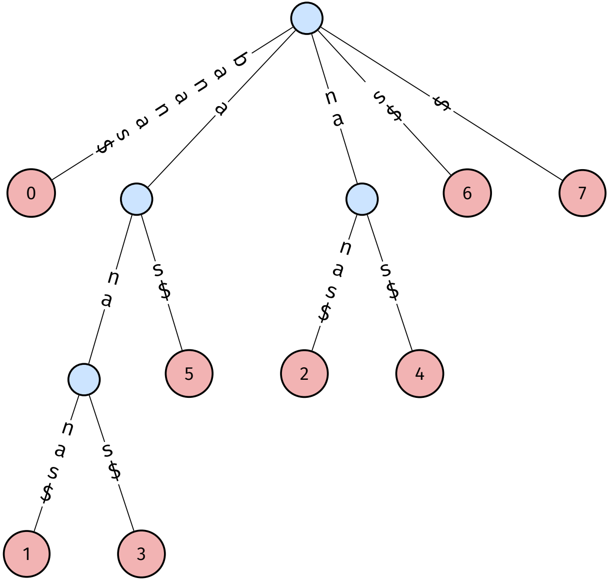
- Recall: a branching word is a substring with more than one extension.
- If a substring is repeated, is it a branching word?
- **No**. Example: `"barkbark"`.
  - `"bar"` is repeated, **not** branching: {`"bark"`}.
  - `"bark"` is repeated, **is** branching: {`"barkb"`, `"bark$"`}.

Claim

- If a substring w is repeated but not a branching word, it can't be the **longest**.
- Why? Since it isn't branching, it has one extension: w'.
- w' must also repeat, since w repeats.
- w' is longer than w, so w can't be the longest.

Claim

- A longest repeated substring must be a branching word.
- Therefore, must be an internal node of the suffix tree of s.



LRS

- Build suffix tree in $\Theta(|s|)$ time.
- Do a DFS in $\Theta(|s|)$ time.
- Keep track of "deepest" internal node. (Depth determined by number of characters.)
- This is a longest repeated substring; found in $\Theta(|s|)$ time.