• Homework 10 due Friday at 2pm
• Final Exam:
  • Friday, May 13 from 5:10pm to 7:00pm in WWH 101
  • Must bring a Laptop! (Email me asap if you do not have one!)
• It is possible to convert regexes to NFAs
  • And then convert NFAs to DFAs, thus the theoretical foundation for being able to convert regexes to programs
• Any regex can be converted to automata with $\varepsilon$-transitions
  • I.e., edges labeled $\varepsilon$ that contribute to the path but not the labels
• Automata with $\varepsilon$-transitions can be converted to equivalent automata without $\varepsilon$-transitions
• You can also convert from automata to regexes, but we won’t cover that today
Idea: build the automata from basis states:

- (a) Automaton for $\emptyset$.
- (b) Automaton for $\epsilon$.
- (c) Automaton for $x$. 
• Construct for operators

(a) Constructing the automaton for the union of two regular expressions.

(b) Constructing the automaton for the concatenation of two regular expressions.

(c) Constructing the automaton for the closure of a regular expression.
Exercise: Construct an automaton with ε-transitions for the regex a|bc*
Exercise: Construct an automaton with $\varepsilon$-transitions for the regex $a|bc^*$
• If we are in any state $S$, then we are effectively in any state we can get to from $S$ by following $\varepsilon$-transitions
• If a state can reach an accepted state with $\varepsilon$-transitions, then the combined state is also acceptable
• Identify **important states**: states that are entered via arcs with non-$\varepsilon$ symbols
• Only important states and the start state are needed
Exercise: How would you write a simple regular expression parser which can match the following:

- a literal character `c`
- any single character `.`
- zero or more occurrences of the previous character
A trie is a tree data structure for holding one or more strings
- Each node of the trie corresponds to a prefix of a stored string
- Can be thought of as a DFA, where accepted states represent stored strings
- Sometimes called a prefix tree
A **suffix trie** of a string $x$ is a trie made from all possible suffixes of $x$

- Each node corresponds to the prefix of a suffix (i.e., a substring)
- A special character (e.g., `$`) is added to the end of $x$ to ensure that all suffixes terminate at a leaf

**Can be too big!**
- Many nodes in the trie have only one child
• A suffix trie can be compressed into a **suffix tree** by allowing edges to have strings as labels instead of just letters
• Can be built in linear time
### Suffix Tries

**Path Label**

- **Path Label** of this vertex is ‘GA’

**Edge Label**

- ‘TAGACA$’ is an edge label

**Merge Vertices**

- Merge vertices with only 1 child

<table>
<thead>
<tr>
<th>i</th>
<th>Suffix</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>GATAGACA$</td>
</tr>
<tr>
<td>1</td>
<td>ATAGACA$</td>
</tr>
<tr>
<td>2</td>
<td>TAGACA$</td>
</tr>
<tr>
<td>3</td>
<td>AGACA$</td>
</tr>
<tr>
<td>4</td>
<td>GACA$</td>
</tr>
<tr>
<td>5</td>
<td>AC$</td>
</tr>
<tr>
<td>6</td>
<td>CA$</td>
</tr>
<tr>
<td>7</td>
<td>A$</td>
</tr>
<tr>
<td>8</td>
<td>$</td>
</tr>
</tbody>
</table>
Exercise: Suppose you have a tree $T$ where every non-leaf node has at least 2 children. If there are $n$ leaves give an upper bound on the total size of the tree.

Exercise: Draw the suffix tree for the string “aababb$". 
The suffix tree for the string “aababb$”: 

![Suffix Tree Diagram]
Exercise: Given a suffix tree for the text $T$ and a pattern string $P$, show how to find all $z$ occurrences of $P$ in $O(|P| + z)$ time.
String Matching

\( T = \text{'GATAGACA$'} \)
\( i = \text{'012345678'} \)

- \( P = \text{'A'} \rightarrow \text{Occurrences: 7, 5, 3, 1} \)
- \( P = \text{'GA'} \rightarrow \text{Occurrences: 4, 0} \)
- \( P = \text{'T'} \rightarrow \text{Occurrences: 2} \)
- \( P = \text{'Z'} \rightarrow \text{Not Found} \)
• You can also build a **generalized suffix tree** which stores a set of strings S
• Combine the suffix trees of each string in S
  • To distinguish between them, use different terminating symbols
  • Mark internal nodes which have leaves with different terminating symbols
  • Figure shows generalized suffix tree for GATAGACA$ and CATA#.
Exercise: (Longest common substring problem) Given a generalized suffix tree for the strings $x_1, \ldots, x_n$, find all substrings of maximum length that are common to every string.

Exercise: Given a list of strings $x_1, \ldots, x_n$, compute for every $k$ the length of the longest string common to $k$ of the $x_i$.

- I.e., $k = 3$ means find the length of the longest substring common to 3 of $x_1, \ldots, x_n$.
• Backstory: Knuth conjectured that suffix trees could not be built in linear time, or else there would be a linear-time solution to the longest common substring problem
  • In 1973, Weiner gave the first linear-time algorithm
  • In 1995, Ukkonen’s gave another linear-time algorithm

• Code will be posted online
Exercise: How would you find the longest repeated substring (i.e., a substring that occurs at least twice), given a suffix tree for a string?

Exercise: How would you use suffix trees to find the longest palindrome in a string?
Competitive Programming 6.5, 6.6