Aho-Corsick

- Given a dictionary of words D. We want to search a text S for all occurrences of any word in D. Let m be the sum of the lengths of all words in D. Let n be the length of S.
- Construct a trie from D. This costs $O(m)$.

  Each `TrieNode` has a `Dictionary{char, TrieNode}` of children. Use a boolean flag to mark which nodes are dictionary nodes. Each node will also have a `suffix TrieNode` (corresponding to longest suffix of the node in the trie) and a `dict_suffix TrieNode` (corresponding to longest suffix also in D). Each `TrieNode` is also associated with a character.
- Add suffix links. This costs $O(m)$. See below.
- Add dictionary suffix links. This costs $O(m)$. See below.

Pseudocode for computing suffix links.

```
# assume all suffixes are initially None
for child of root
    child.suffix = root
    Q.push(child)

while (Q is not empty)
    parent = Q.pop
    for child of parent
        key = child.char
        temp = parent.suffix
        while key not in temp.child_dict and temp is not root
            temp = temp.suffix
        if key in suffix.children_dict
            child.suffix = temp.children[key]
        Q.push(child)
```

Pseudocode for computing the dictionary suffix links.

```
# all dict_suffix links are initially None
Q.push(root)
while (Q is not empty)
    node = Q.pop
    if (suffix := n.suffix) is not None
        n.dict_suffix = suffix if suffix.is_dict_node
                                              else suffix.dict_suffix
    add node.children to Q
```

Since the trie is acyclic, we don't have to worry about maintaining a visited array in our breadth-first traversals.

- Now, we've constructed our Aho-Corasick trie T for D, so it's time to search S.
- Set state = root.
- For each character c of S, if we can append c to the current state to get a valid node, i.e. a child of the current state has character c, set the new state to that child. Otherwise, check the children of the suffix. Otherwise, check the child of the suffix's suffix, and so on until we reach the root.

  After we update the state, we output all dictionary entries that end at c's position in the text. Print every node reachable via dictionary suffix links. Print the current node, if the current node is itself is a dictionary node.