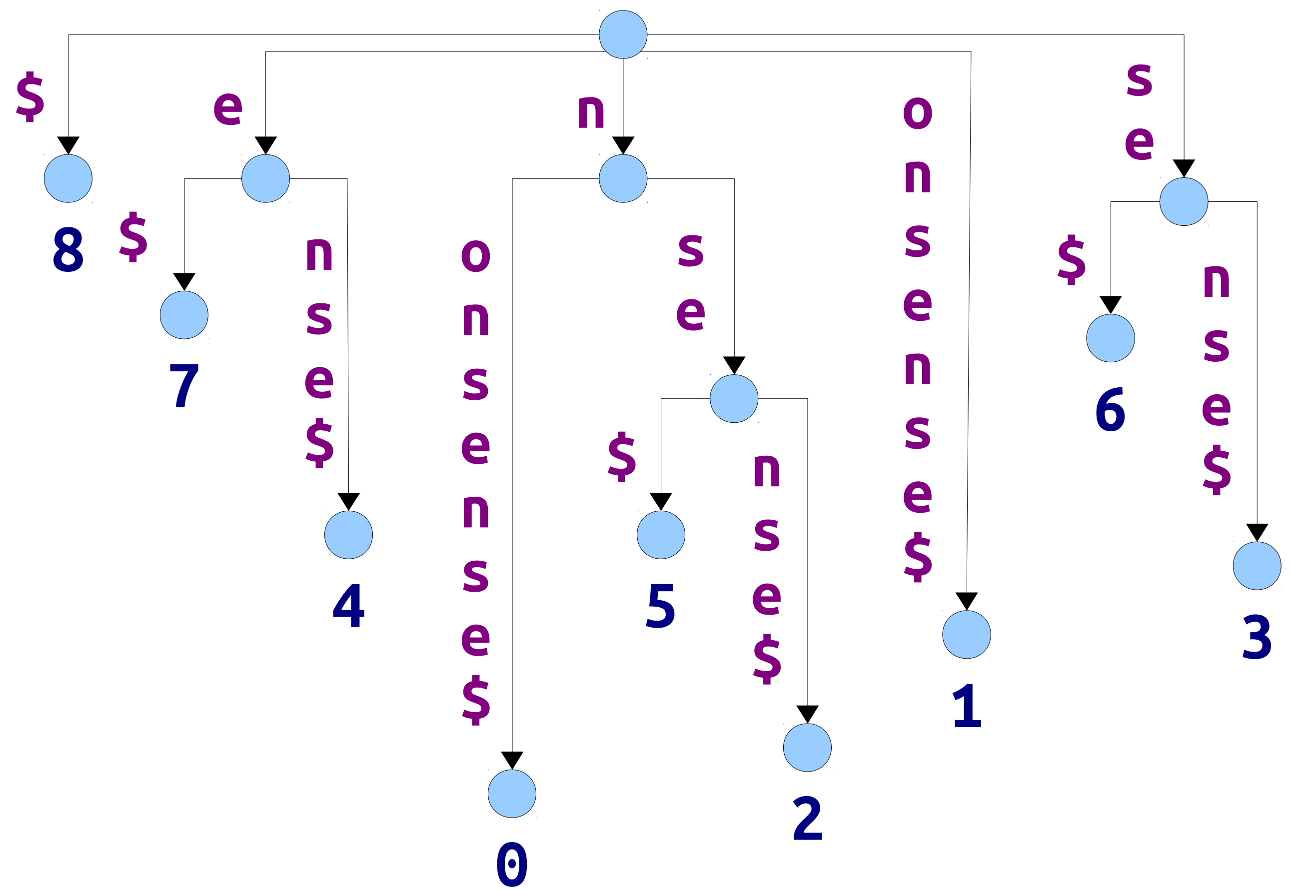
Suffix Trees

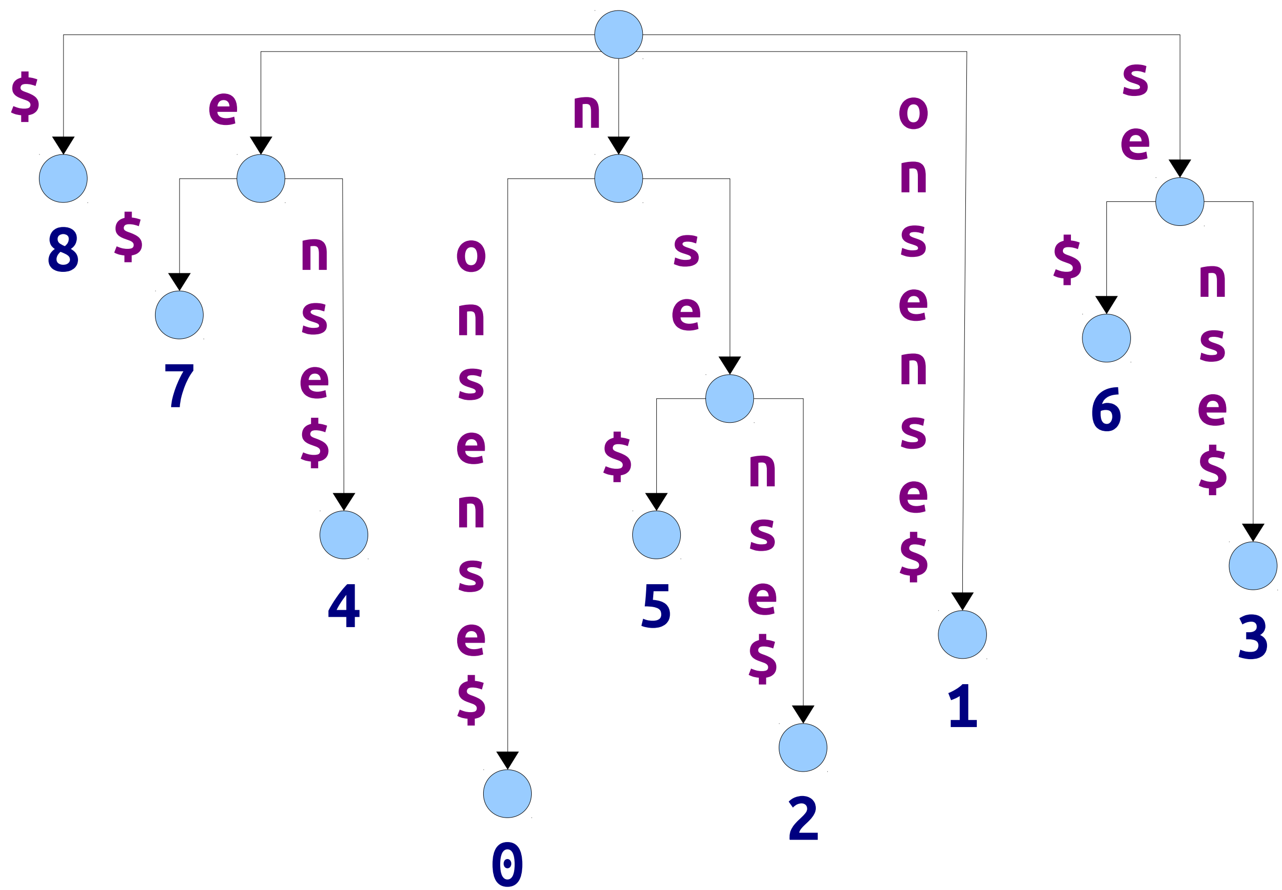
- A **suffix tree** for a string *T* is an Patricia trie of *T*$ where each leaf is labeled with the index where the corresponding suffix starts in *T*$.



nonsense$
012345678

Properties of Suffix Trees

- If $|T| = m$, the suffix tree has exactly $m + 1$ leaf nodes.
- For any $T \neq \varepsilon$, all internal nodes in the suffix tree have at least two children.
- Number of nodes in a suffix tree is $\Theta(m)$.

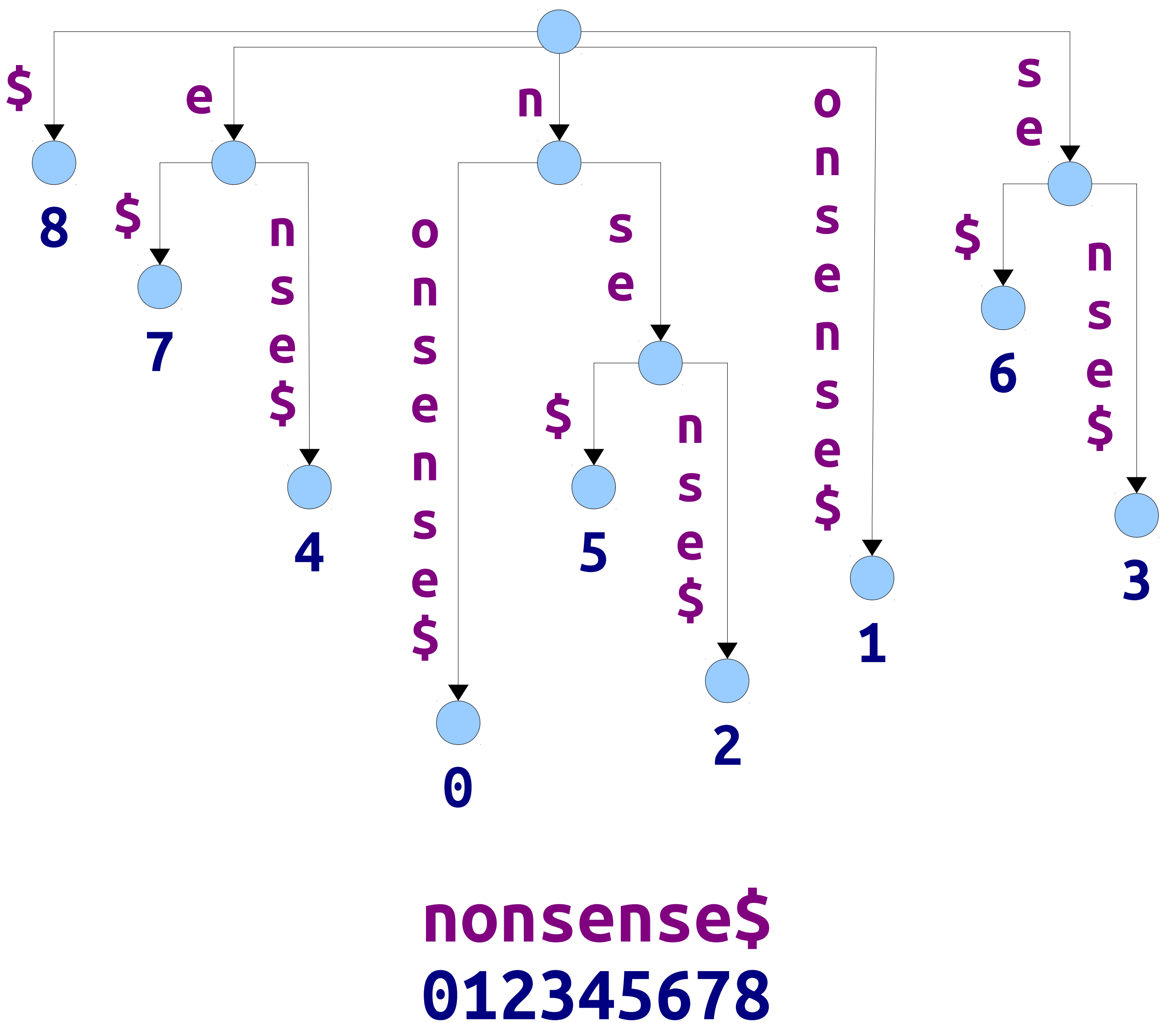


nonsense$
012345678

An Application: String Matching

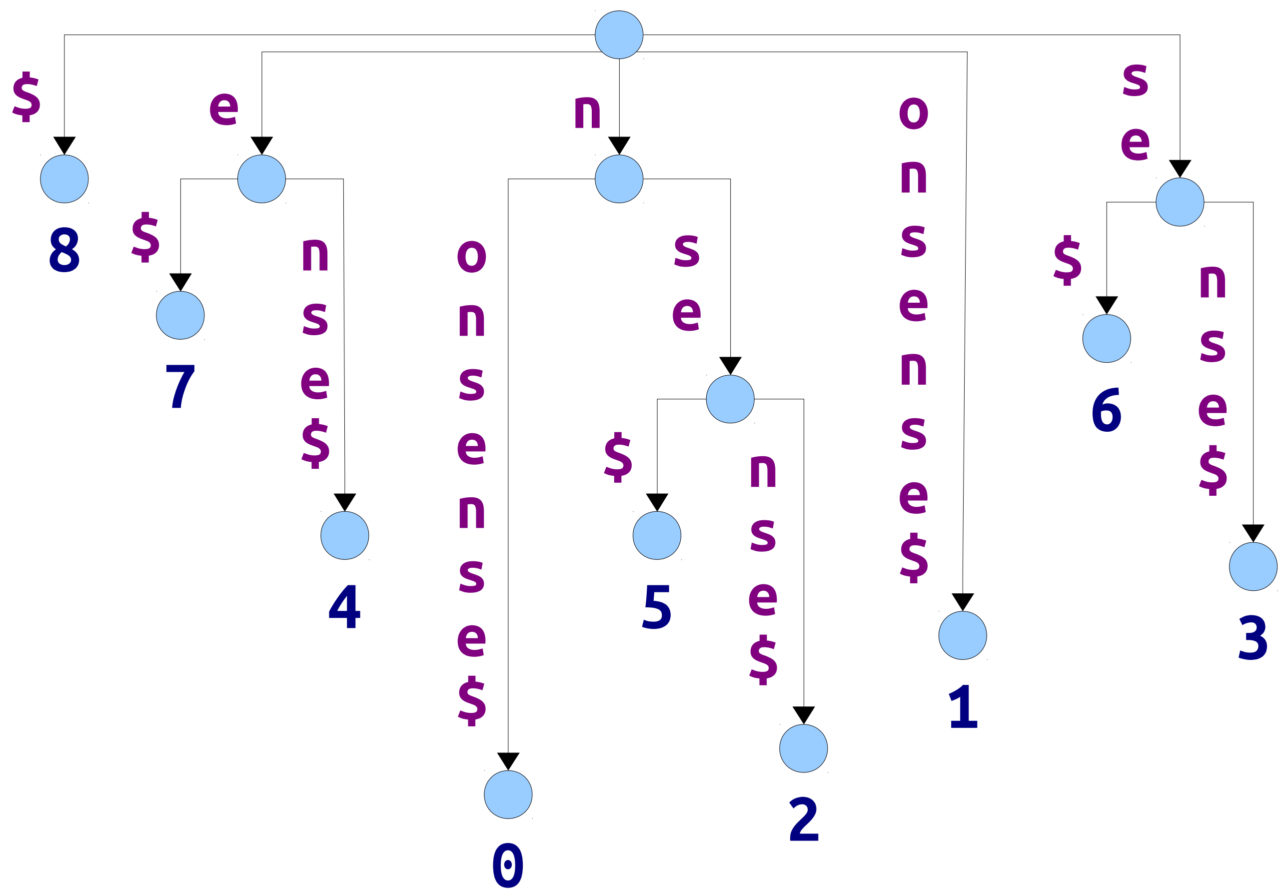
String Matching

- Given a suffix tree, can search to see if a pattern P exists in time $O(n)$.
- Gives an $O(m + n)$ string-matching algorithm.
- T can be preprocessed in time $O(m)$ to efficiently support binary string matching queries.



String Matching

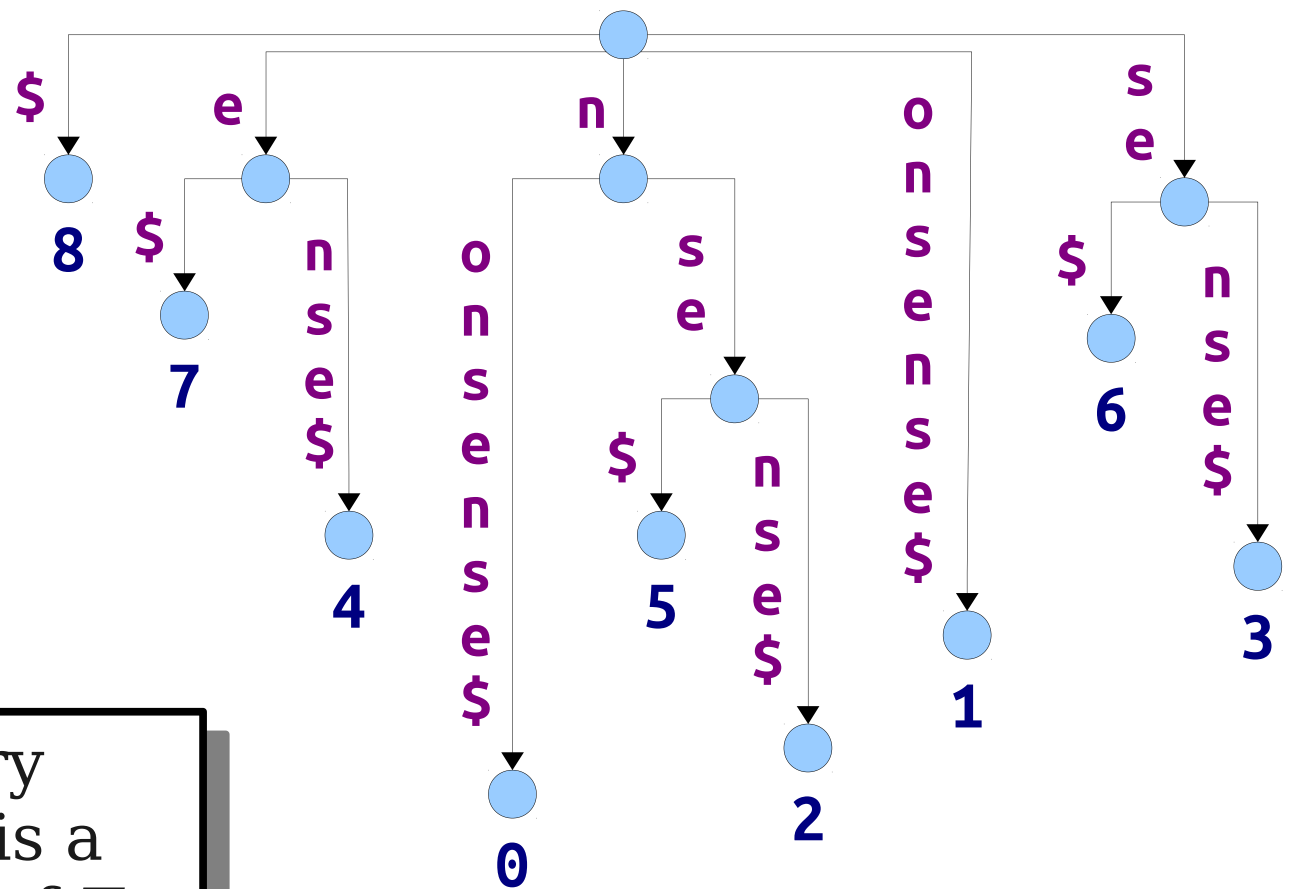
- **Claim:** After spending O(m) time preprocessing T\$, can find all matches of a string P in time O(n + z), where z is the number of matches.



nonsense$
012345678

String Matching

- **Claim:** After spending O(m) time preprocessing $T\$$, can find all matches of a string P in time O($n + z$), where z is the number of matches.

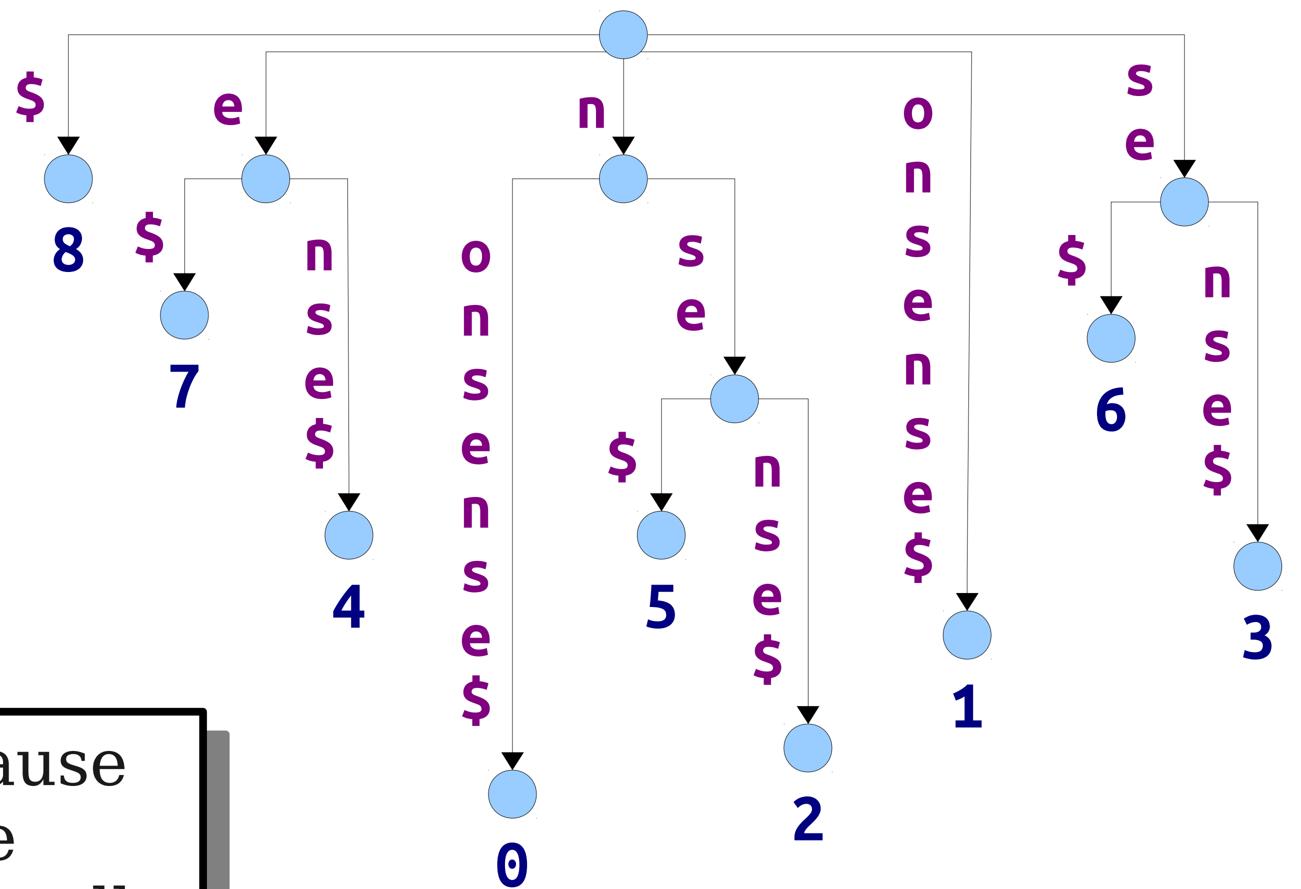


nonsense$
012345678

> **Observation 1:** Every occurrence of P in T is a prefix of some suffix of T.

String Matching

- **Claim:** After spending O(m) time preprocessing T\$, can find all matches of a string P in time O($n + z$), where z is the number of matches.

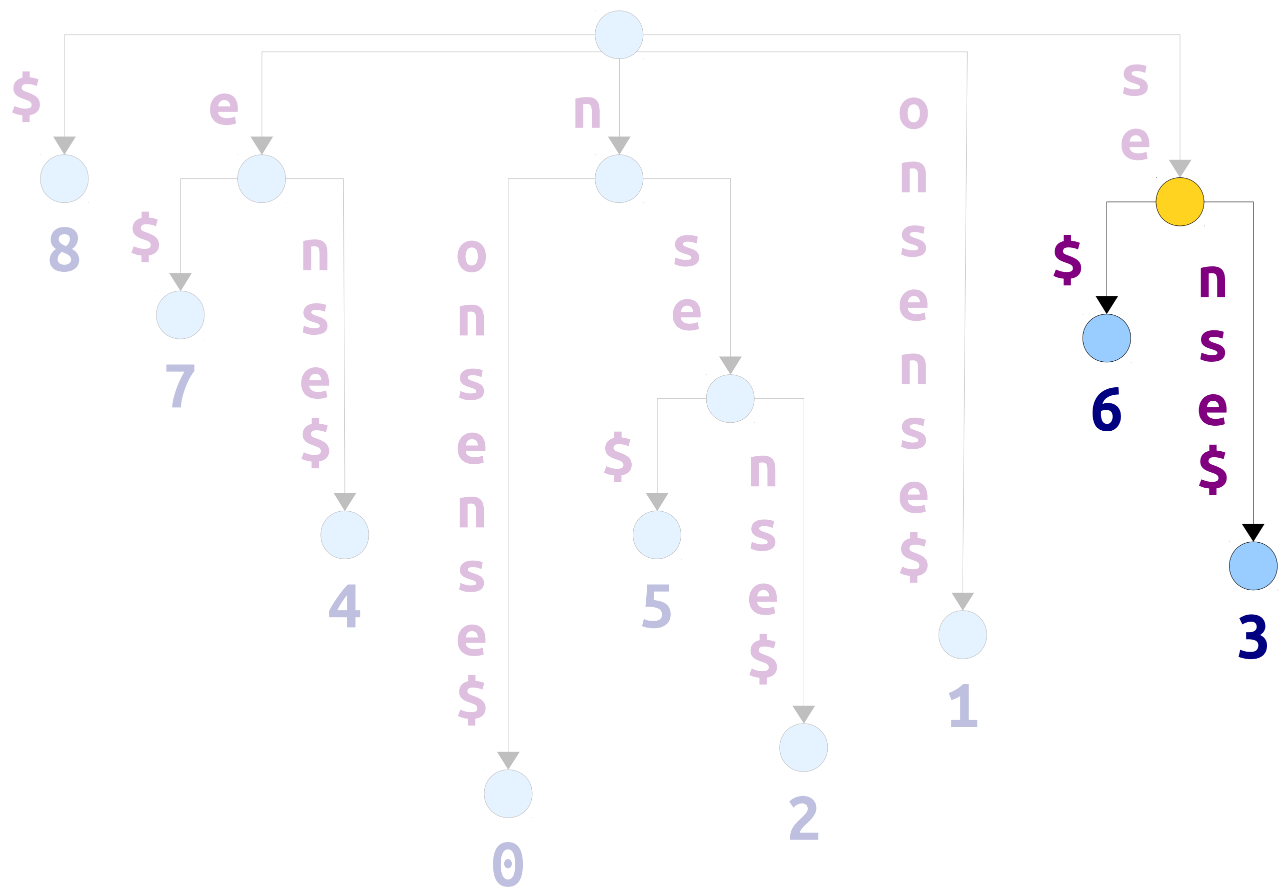


nonsense$
012345678

> **Observation 2:** Because the prefix is the same each time (namely, P), all those suffixes will be in the same subtree.

String Matching

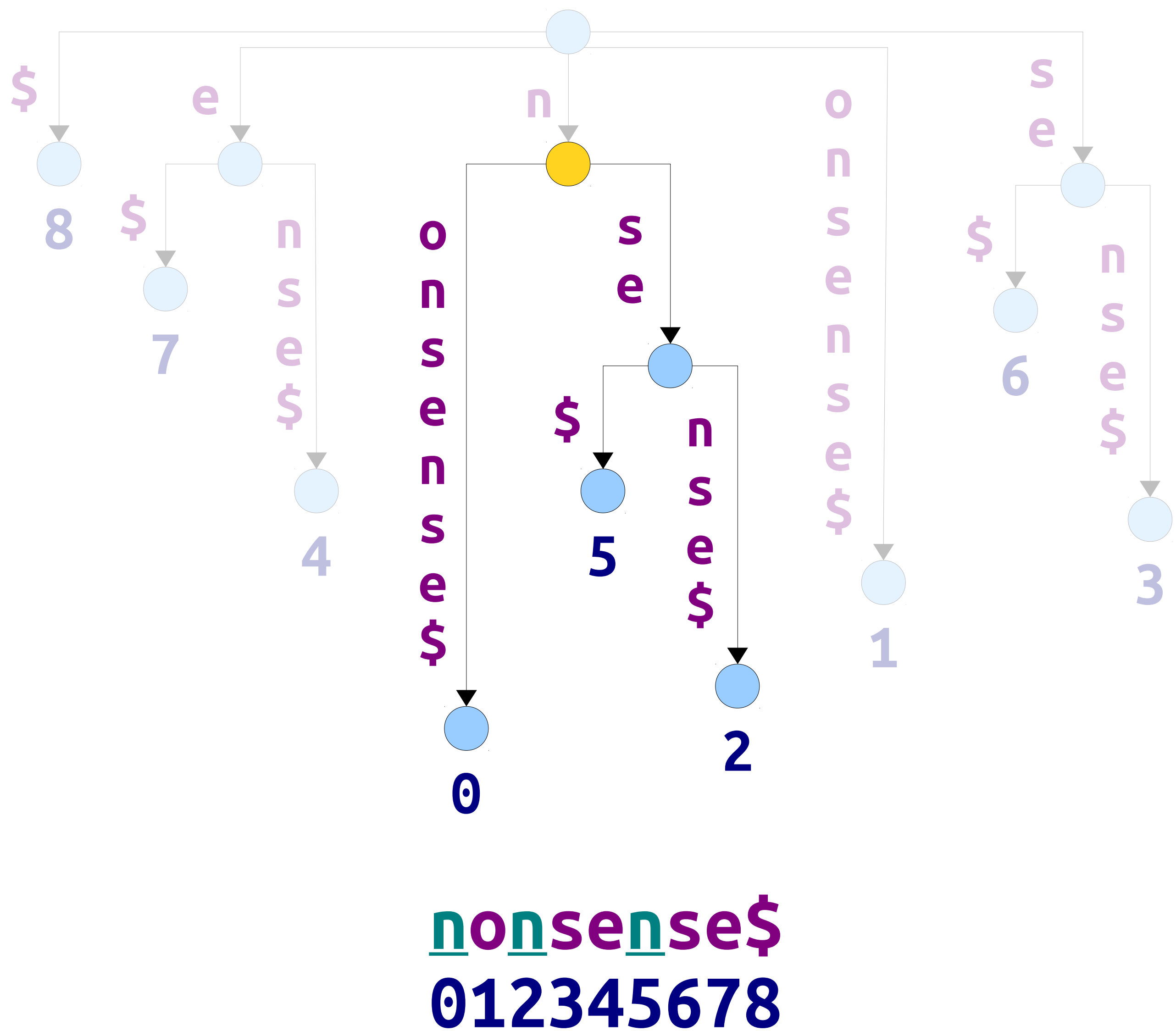
- **Claim:** After spending O(m) time preprocessing T\$, can find all matches of a string P in time O(n + z), where z is the number of matches.



nonsense$
012345678

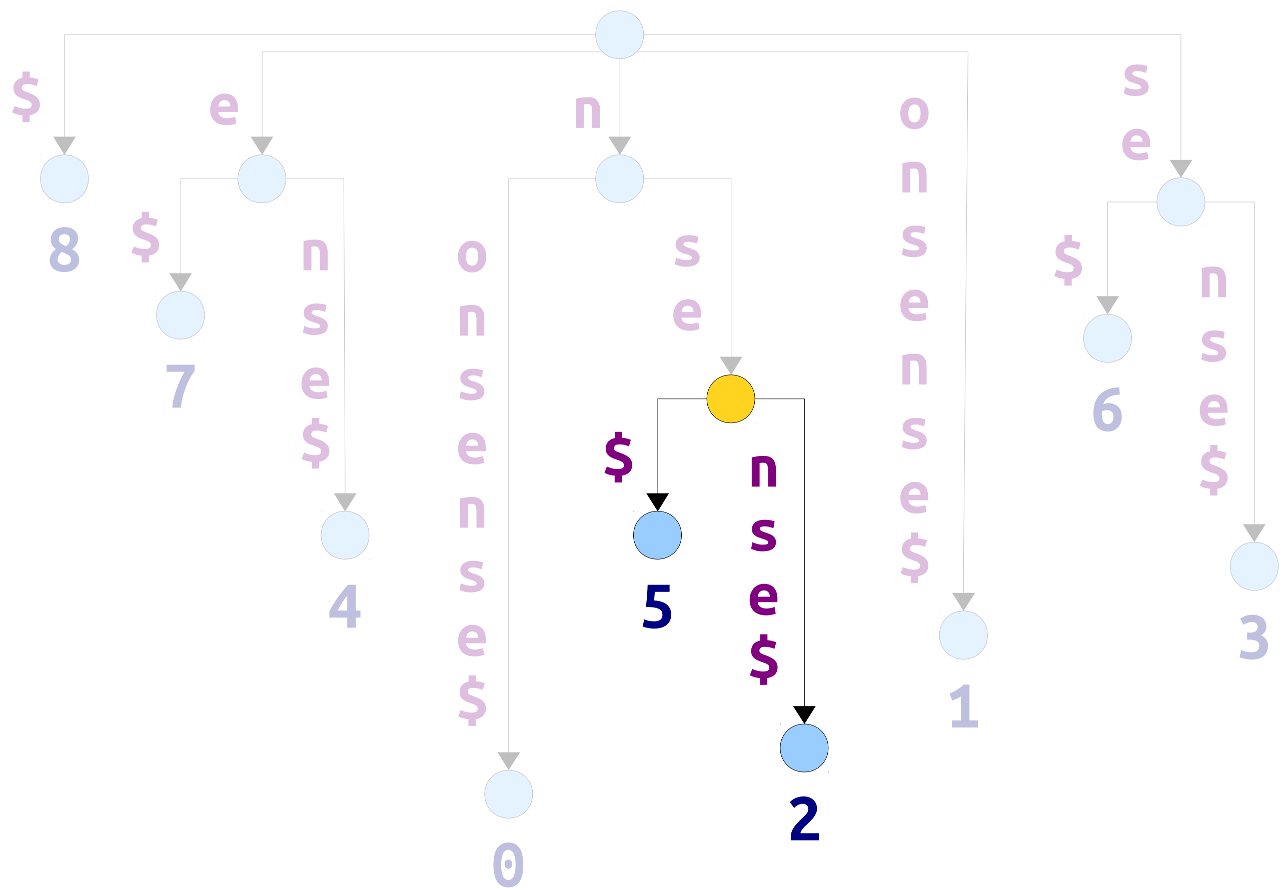
String Matching

- **Claim:** After spending O(m) time preprocessing T\$, can find all matches of a string P in time O(n + z), where z is the number of matches.



String Matching

- **Claim:** After spending O(m) time preprocessing T\$, can find all matches of a string P in time O(n + z), where z is the number of matches.



nonsense$
012345678

Finding All Matches

- To find all matches of string P, start by searching the tree for P.
- If the search falls off the tree, report no matches.
- Otherwise, let v be the node at which the search stops, or the endpoint of the edge where it stops if it ends in the middle of an edge.
- Do a DFS and report all leaf numbers found. The indices reported this way give back all positions at which P occurs.

Claim: The DFS to find all leaves in the subtree corresponding to prefix P takes time $O(z)$, where z is the number of matches.

Proof: If the DFS reports z matches, it must have visited z different leaf nodes.

Since each internal node of a suffix tree has at least two children, the total number of internal nodes visited during the DFS is at most $z - 1$.

During the DFS, we don't need to actually match the characters on the edges. We just follow the edges, which takes time $O(1)$.

Therefore, the DFS visits at most $O(z)$ nodes and edges and spends $O(1)$ time per node or edge, so the total runtime is $O(z)$. ■

Another Application:

Longest Repeated Substring

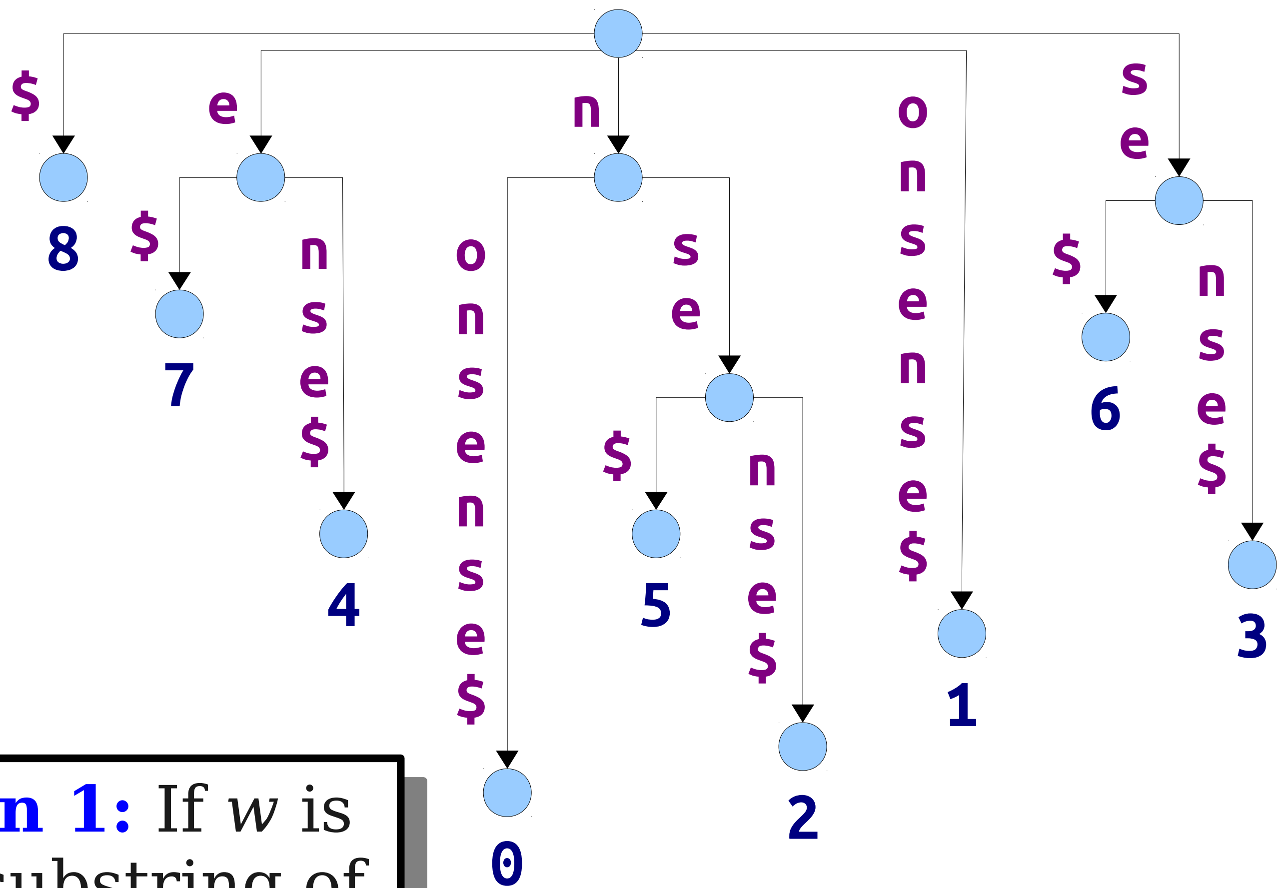
Longest Repeated Substring

- Consider the following problem:

  Given a string T, find the longest substring w of T that appears in at least two different positions.

- Applications to computational biology: more than half of the human genome is formed from repeated DNA sequences!

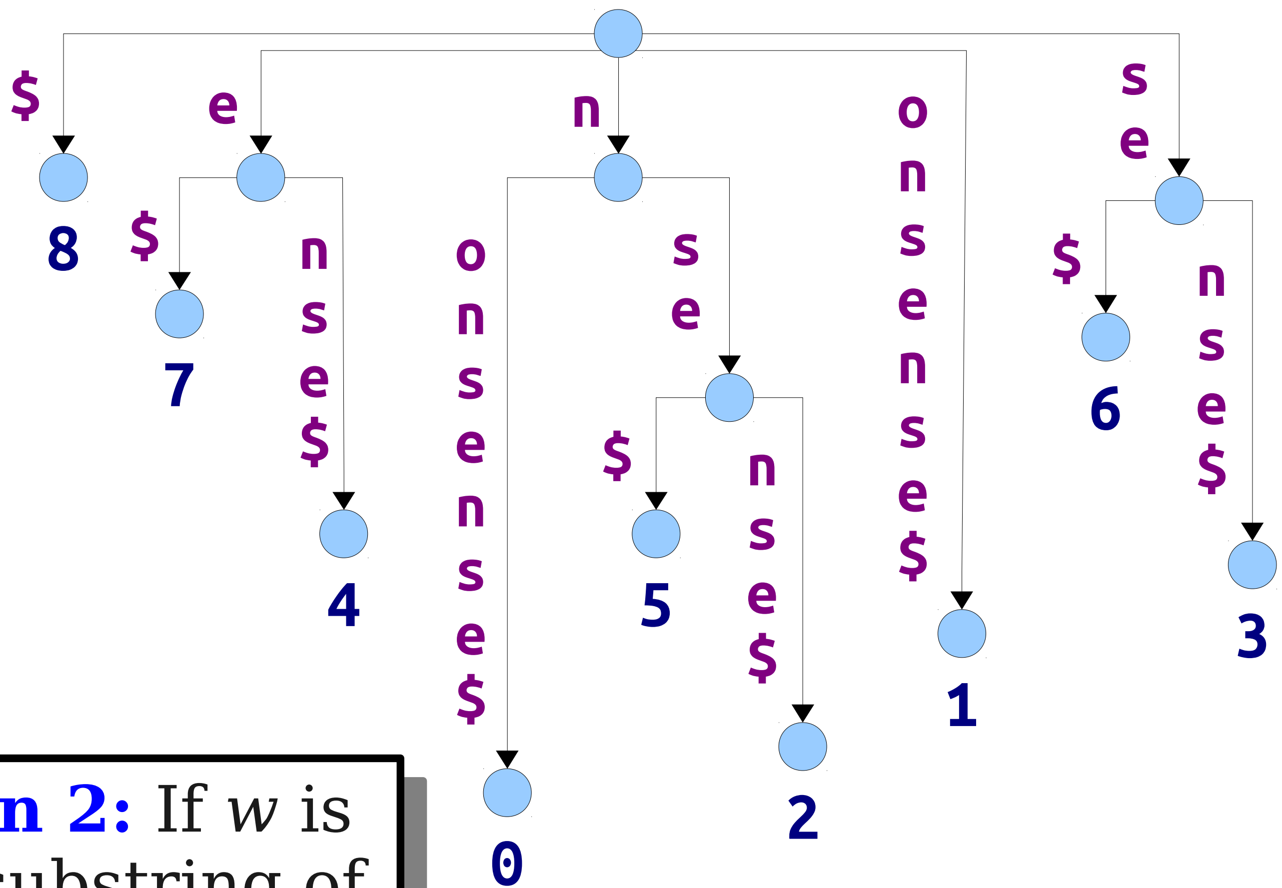
Longest Repeated Substring



Observation 1: If w is a repeated substring of T, it must be a prefix of at least two different suffixes.

nonsense$
012345678

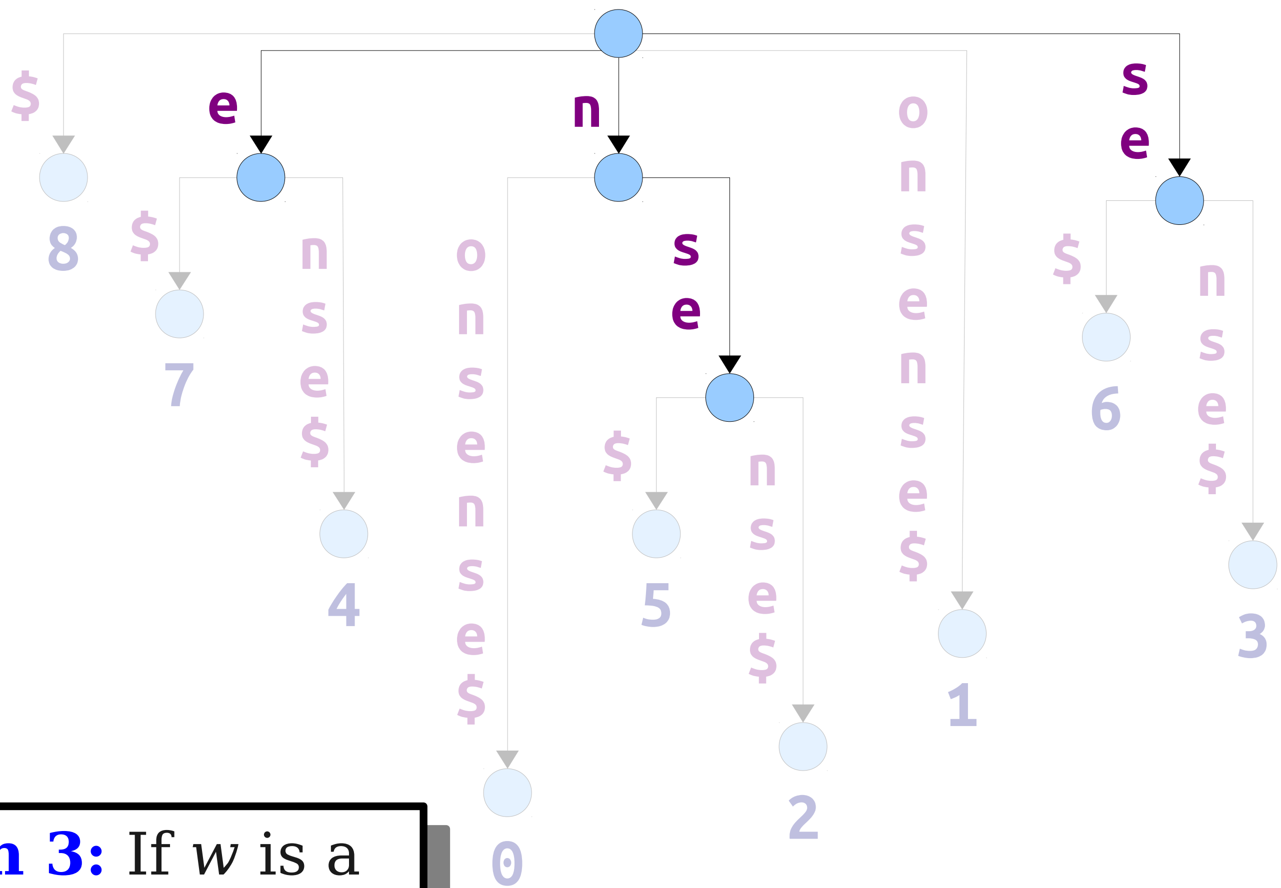
Longest Repeated Substring



Observation 2: If *w* is a repeated substring of *T*, it must correspond to a prefix of a path to an internal node.

nonsense$
012345678

Longest Repeated Substring



Observation 3: If *w* is a longest repeated substring, it corresponds to a full path to an internal node.

nonsense$
012345678

Longest Repeated Substring

- For each node v in a suffix tree, let $s(v)$ be the string that it corresponds to.
- The *string depth* of a node v is defined as $|s(v)|$, the length of the string v corresponds to.
- The longest repeated substring in T can be found by finding the internal node in T with the maximum string depth.

Longest Repeated Substring

- Here's an O(m)-time algorithm for solving the longest repeated substring problem:
  - Build the suffix tree for T in time O(m).
  - Run a DFS over T, tracking the string depth as you go, to find the internal node of maximum string depth.
  - Recover the string T corresponds to.
- **Good exercise:** How might you find the longest substring of T that repeats at least k times?