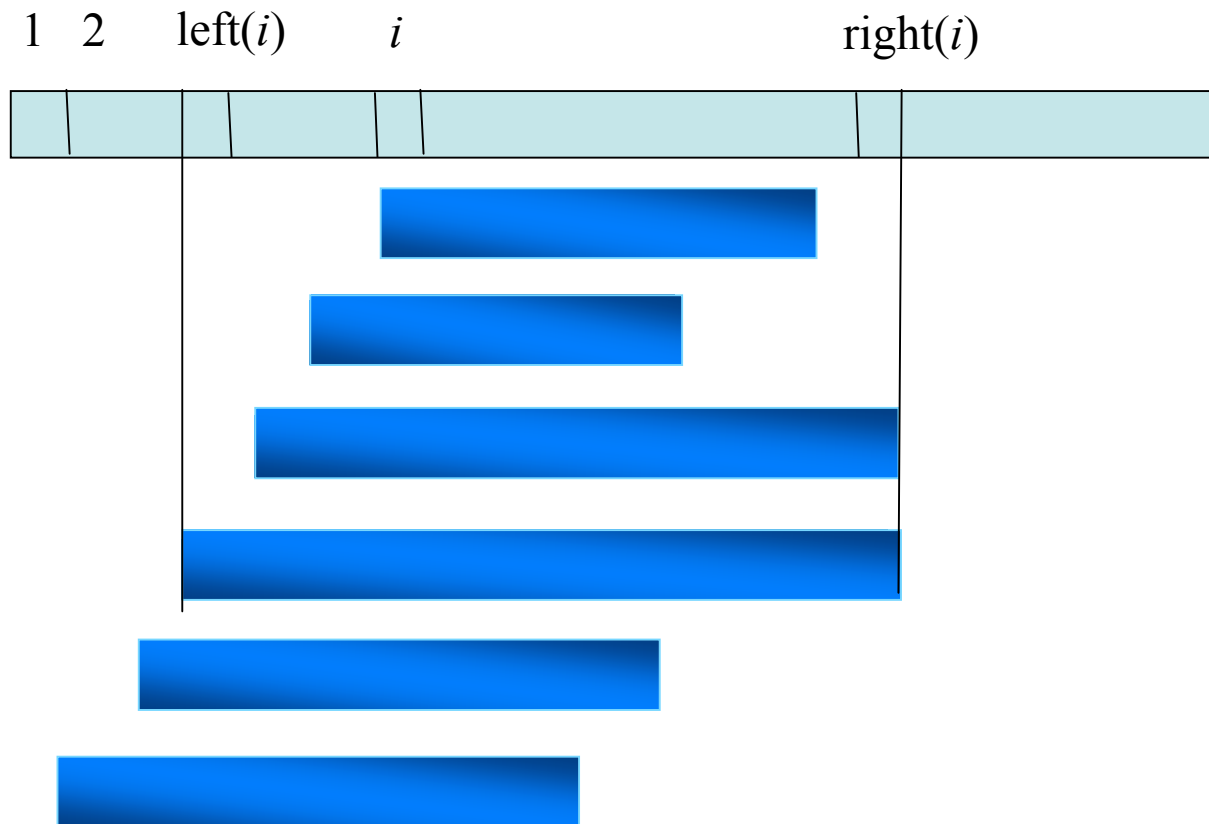


Calculate Z values in linear
time

Notation

- $\text{right}(i) = \max \{j + Z(j) - 1 \mid 1 < j \leq i\}$.
- $\text{left}(i) = \min \{j \mid \text{right}(j) = \text{right}(i)\}$.
- Observation: $\text{left}(i)$ and $\text{right}(i)$ nondecreasing.

Illustration



Strategy

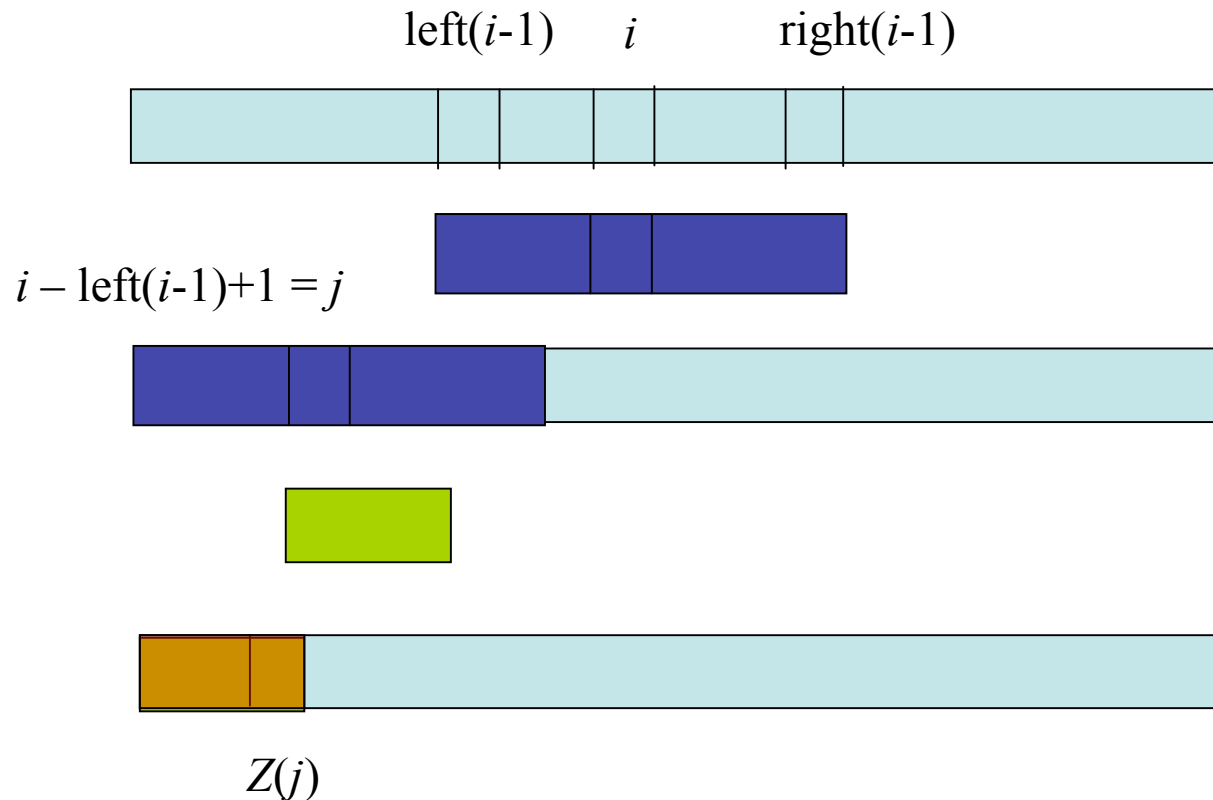
- Computing $Z(i)$, $\text{right}(i)$, $\text{left}(i)$ from
 - $Z(1), Z(2), \dots, Z(i - 1)$;
 - $\text{right}(i - 1)$;
 - $\text{left}(i - 1)$.

Case 1: $\text{right}(i-1) \leq i-1$.

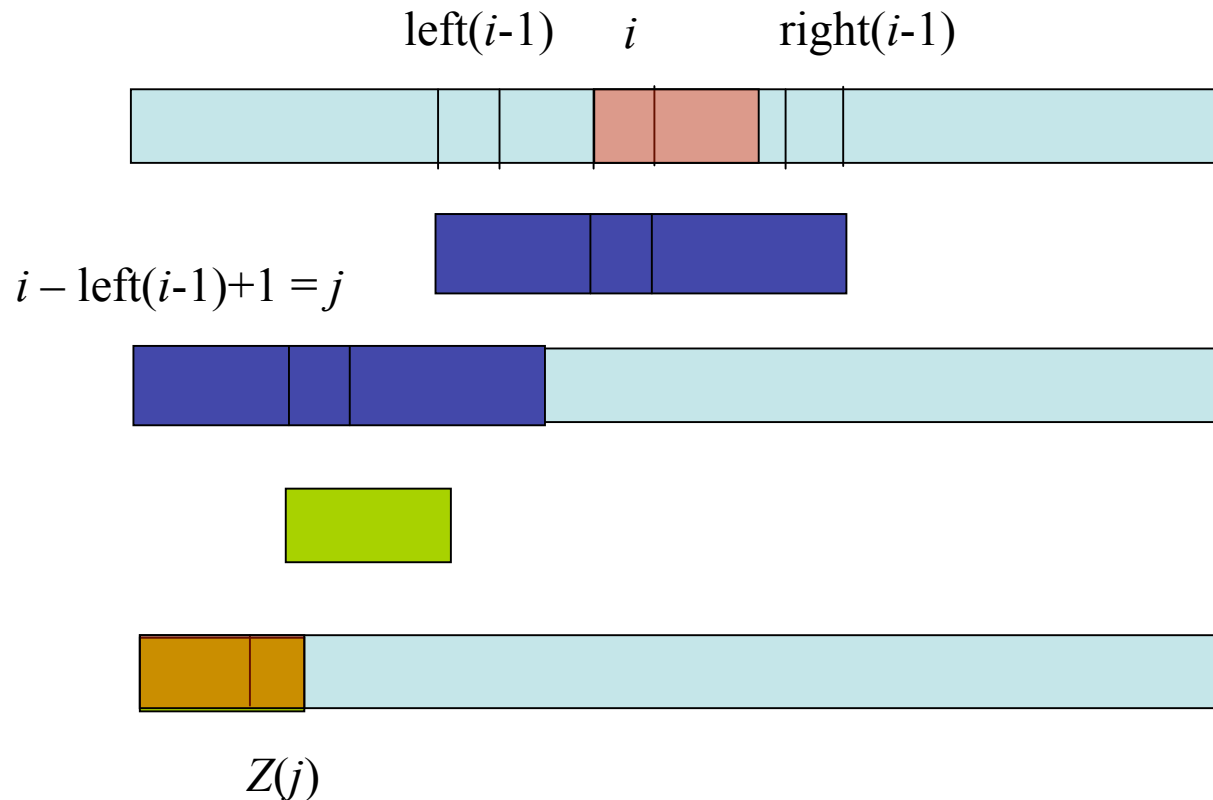
- $\text{right}(i-1)$ does not cover i .
- Computing $Z(i)$ naively in $O(1+Z(i))$ time.
- $\text{left}(i) = i$.
- $\text{right}(i) = i + Z(i) - 1$.
- Observation

$$1+Z(i) = \text{right}(i) - i + 2 \leq \text{right}(i) - \text{right}(i-1) + 1.$$

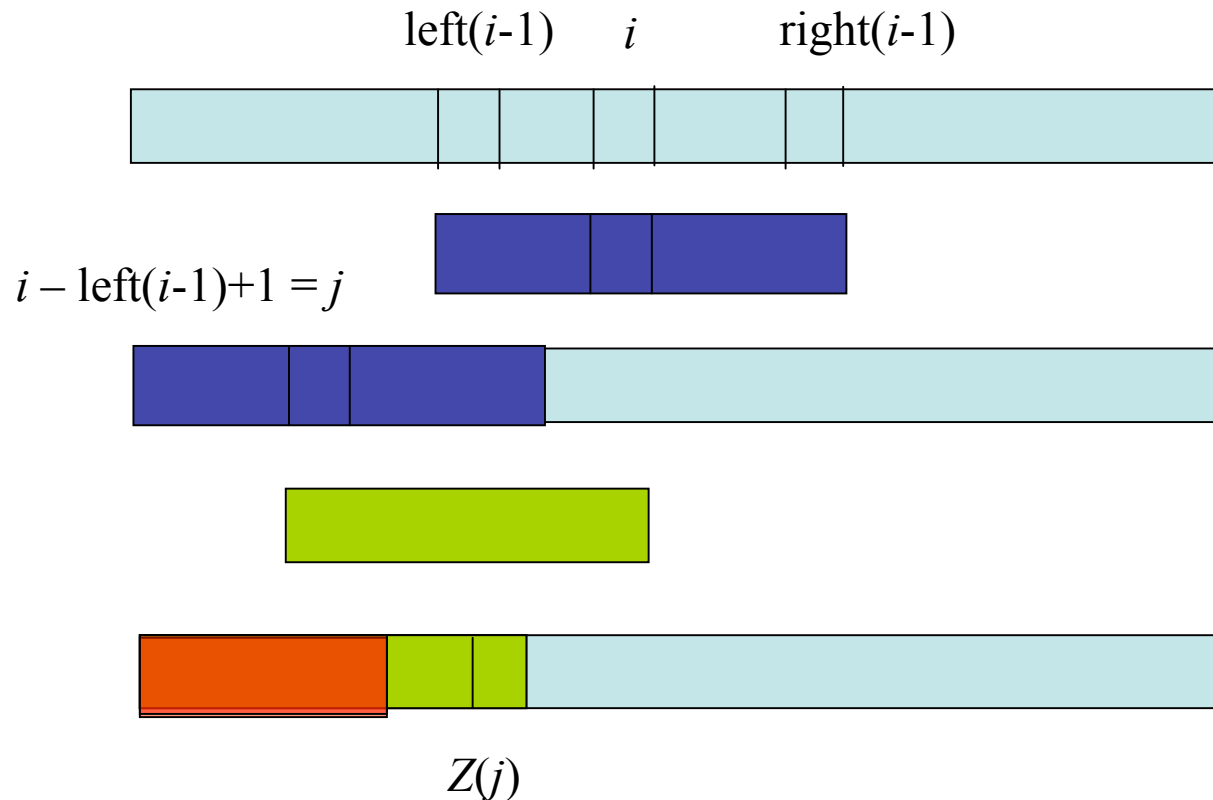
Case 2: $\text{right}(i-1) \geq i$
 and $Z(j) < \text{right}(i-1) - i + 1$.



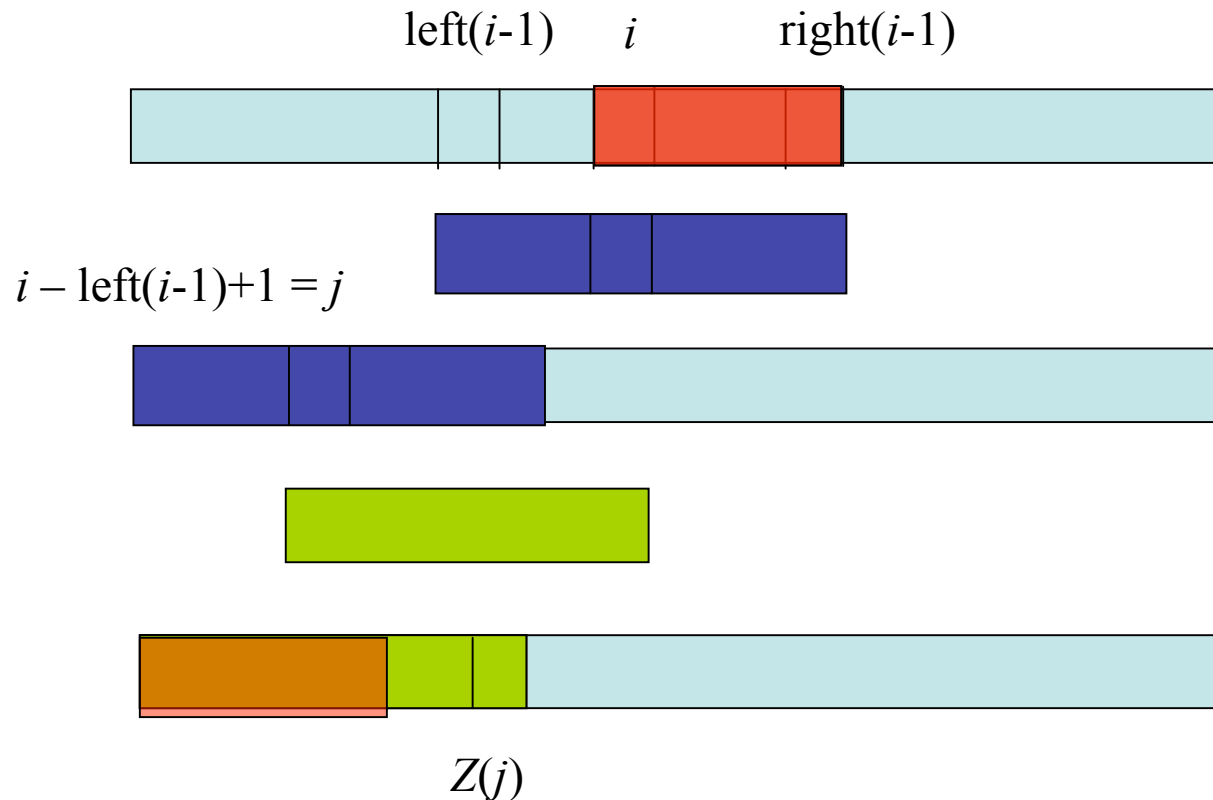
$$Z(i) = Z(j), \text{ left}(i) = \text{left}(i-1), \\ \text{right}(i) = \text{right}(i-1).$$



Case 3: $\text{right}(i-1) \geq i$
 and $Z(j) \geq \text{right}(i-1) - i + 1$.



Finding $Z(i)$ by comparisons starting from $\text{right}(i-1)+1$.



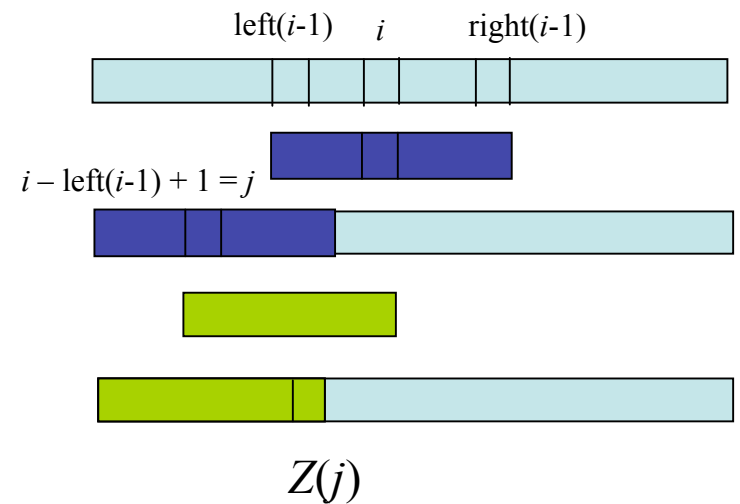
Computing left(i) and right(i).

- $\text{right}(i) = i + Z(i) - 1.$

- $\text{left}(i) = i.$

- How many comparisons?

 - $\text{right}(i) - \text{right}(i-1) + 1.$



Time complexity is linear

- Case 1:
 - $O(|Z(i)|+1) = O(\text{right}(i)-\text{right}(i-1)+1)$.
- Case 2:
 - $O(1) = O(\text{right}(i)-\text{right}(i-1)+1)$.
- Case 3:
 - $O(\text{right}(i)-\text{right}(i-1)+1)$.

Overall time complexity: $O(|S|)$

Suffix trees

Data structures for string pattern matching: Suffix trees

- Linear algorithms for exact string matching
 - KMP
 - Z-value algorithm
- What is suffix tree?
 - A tree-like **data structure** for solving problems involving strings.
 - Related data structures: Trie (**retrieval**) & PATRICIA (radix tree)
 - Allow the storage of **all substrings** of a given string in linear space
 - Simple algorithm to solve string pattern matching problem in linear time

Better than hash tables?

- Hash tables are certainly easier to understand. And, one can produce a hash table of all length k strings in $O(m)$ time and look up a k -length string x in $O(k)$ time, finding all p places where string x is found in $O(p)$ time. This is the same as the bound for suffix trees.
- What if you don't know how long the string x is going to be?
- And most other string matching tricks don't work for it either.

Suffix Tree: definition

- A suffix tree ST for an m -character string S is a rooted directed tree with exactly m leaves numbered 1 to m .
- Each internal node, other than the root, has at least two children and each edge is labeled with a nonempty substring of S .

Suffix tree: definition

- No two edges out of a node can have edge-labels beginning with the same character.
- The key feature of the suffix tree is that for any leaf i , the concatenation of the edge-labels on the path from the root to the leaf i exactly spells out the suffix of S that starts at position i .

Suffix Trees: Example

- Suffixes of 'papua'
 - 'papua'
 - 'apua'
 - 'pua'
 - 'ua'
 - 'a'
 - ''

Suffix Trees: Example

- Suffixes of 'papua'

- 'papua'

- 'apua'

- 'pua'

- 'ua'

- 'a'

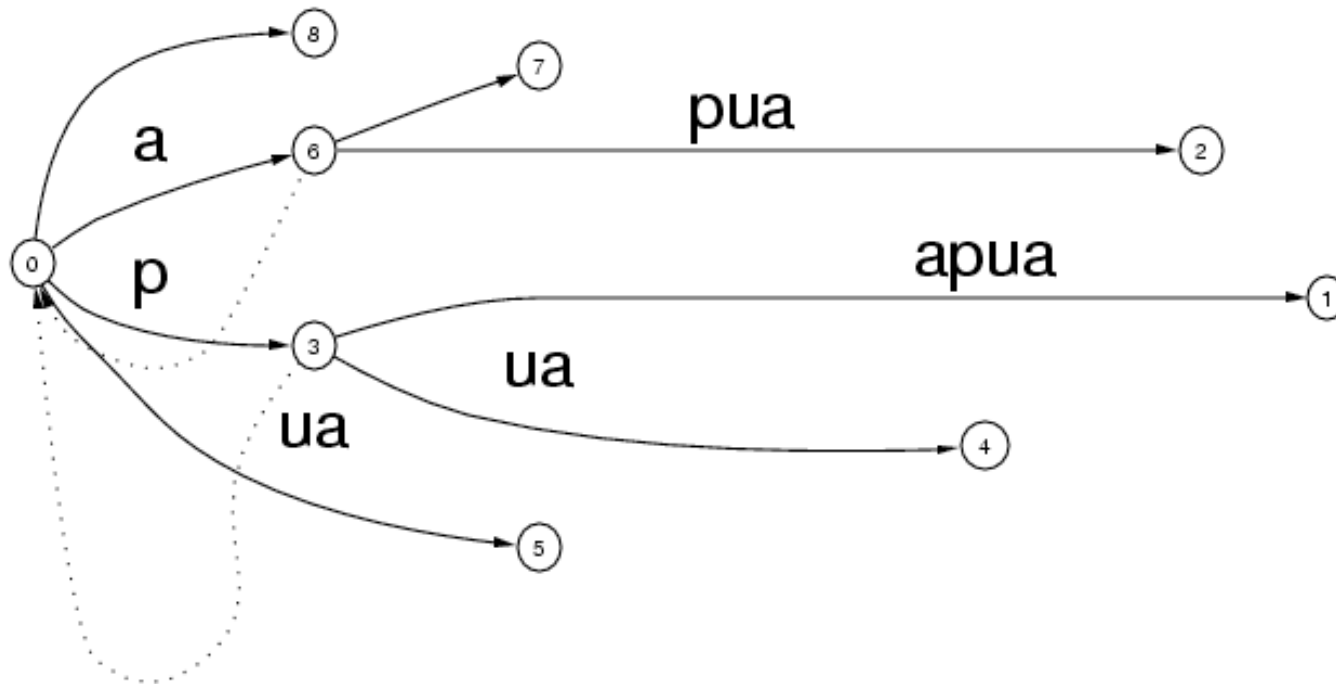
- ''



NOTE: Assume the string terminates with some character found nowhere else in the string. (eg. '\0')

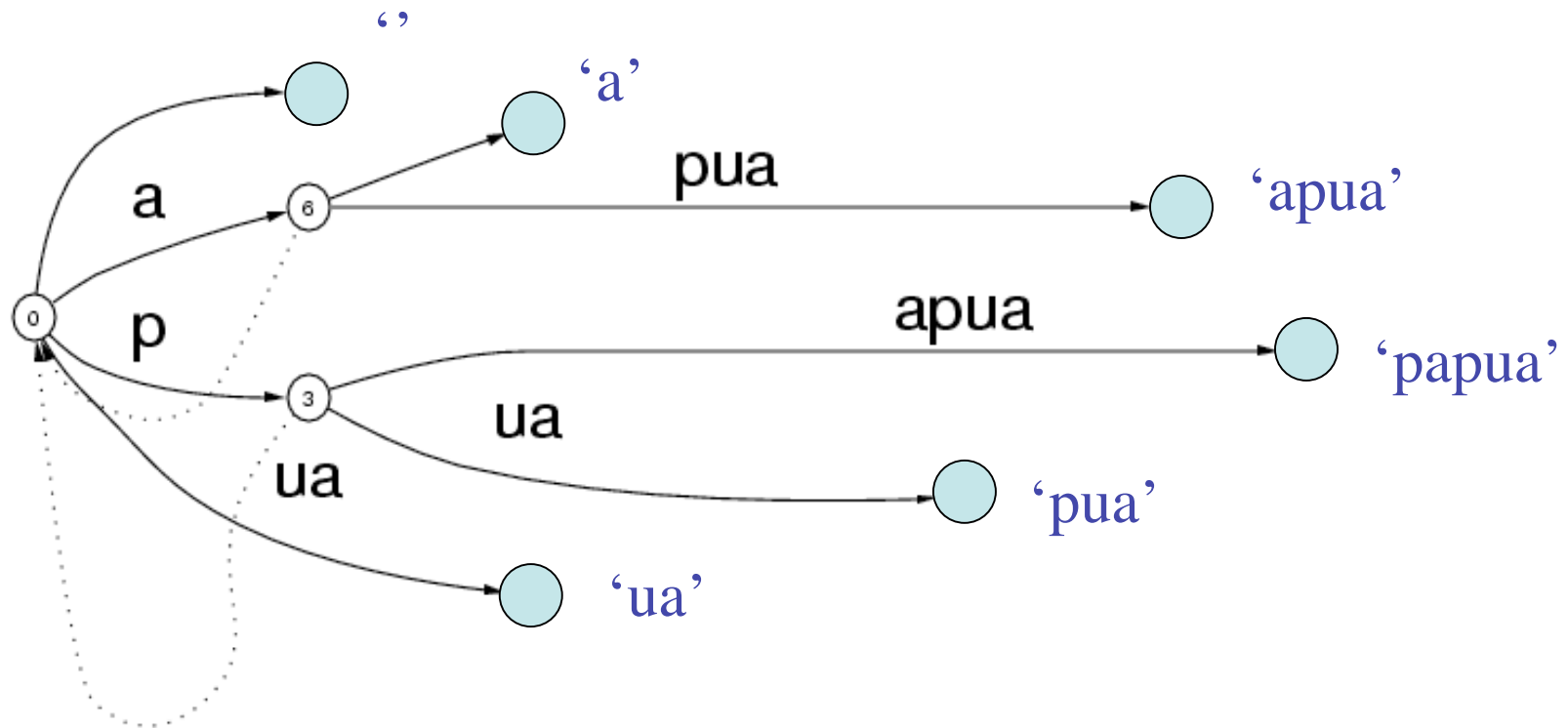
Suffix Trees: Example

- Suffix tree for 'papua'

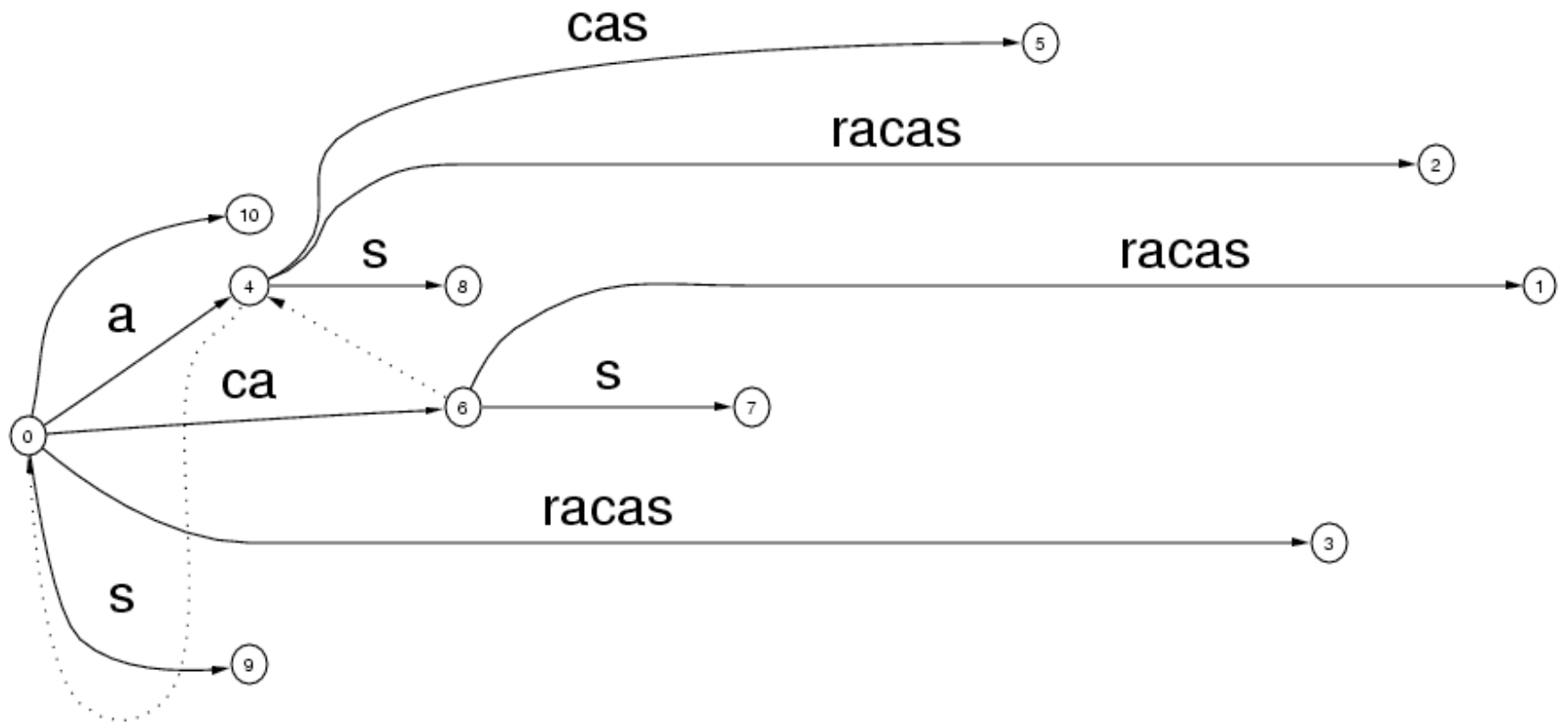


Suffix Trees: Example

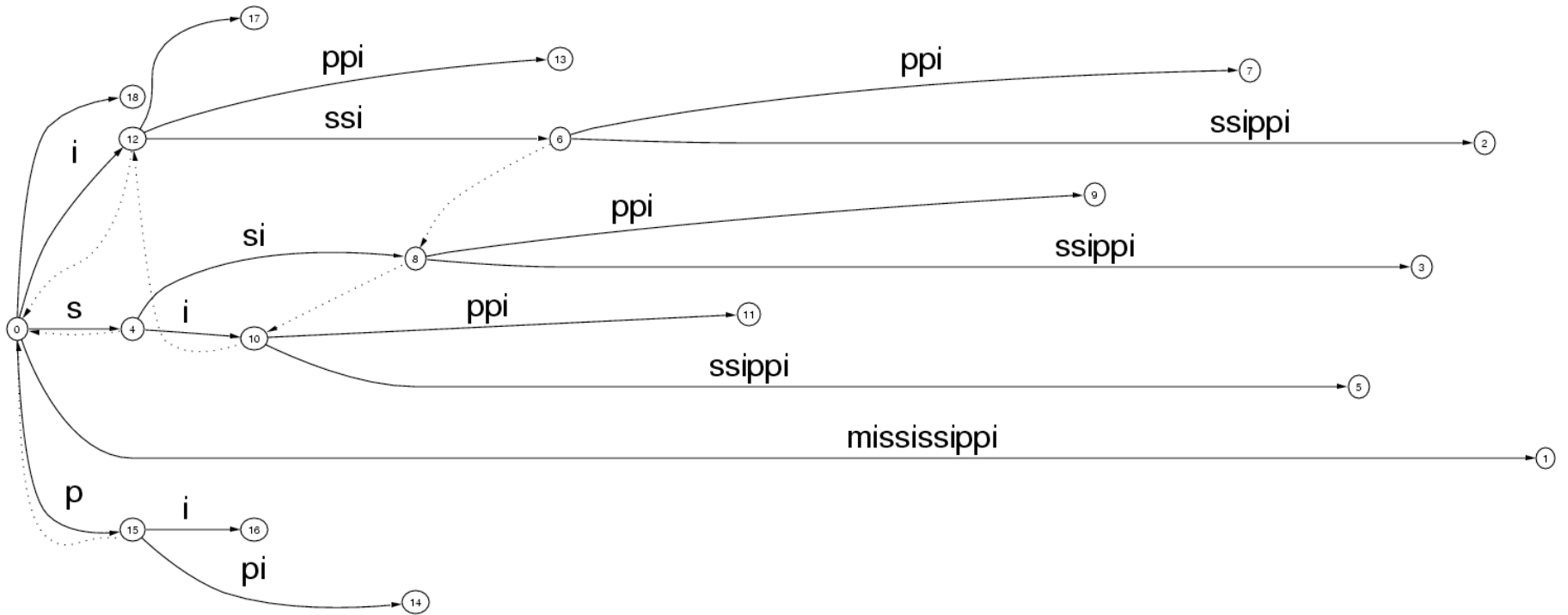
- Suffix tree for 'papua'



caracas



mississippi



Suffix Trees

- Exact matching in linear time
- Many others
- “We know of no other single data structure that allows efficient solutions to such a wide range of complex string problems.” - Dan Gusfield

Exact string matching problem

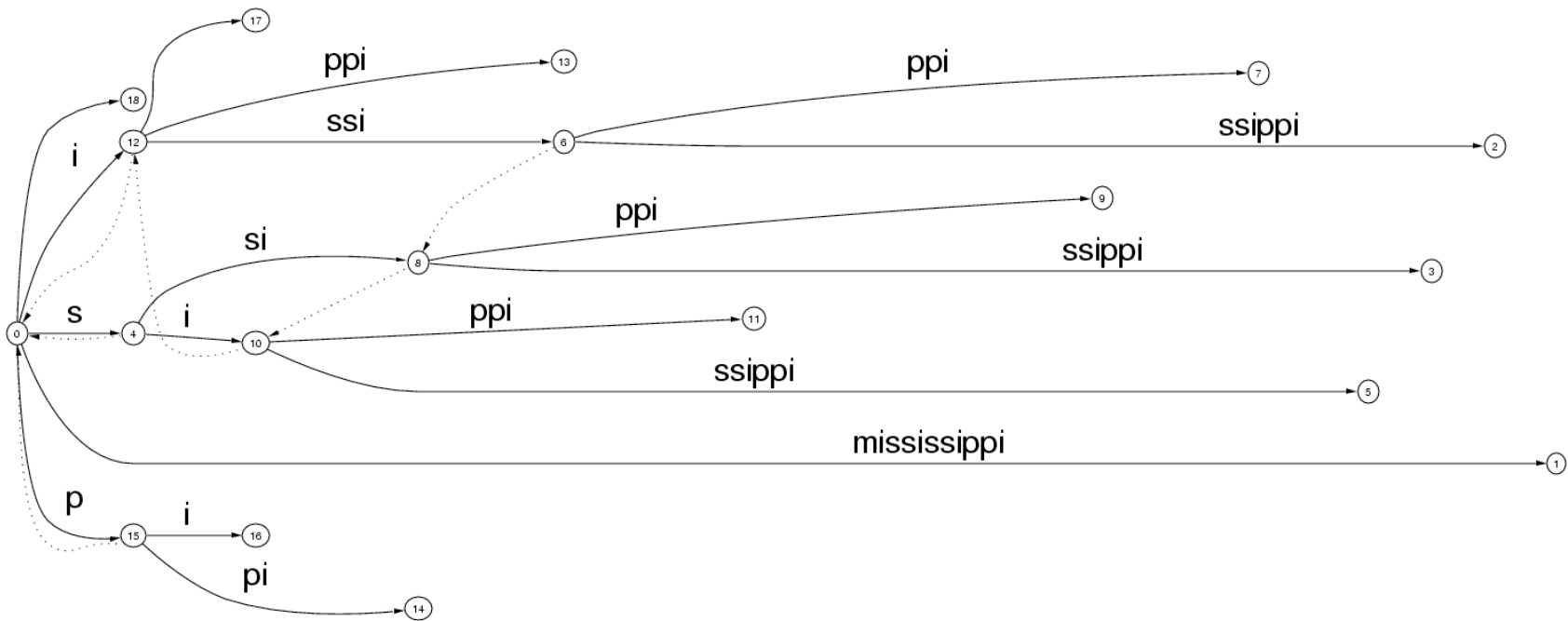
- Given a pattern P of length m , find all occurrences of P in text T
 - $O(n+m)$ algorithm
- Solution: Build a suffix tree ST for text T in $O(m)$ time. Then, match the characters of P along the unique path in ST until either P is exhausted or no more matches are possible.

Exact string matching problem

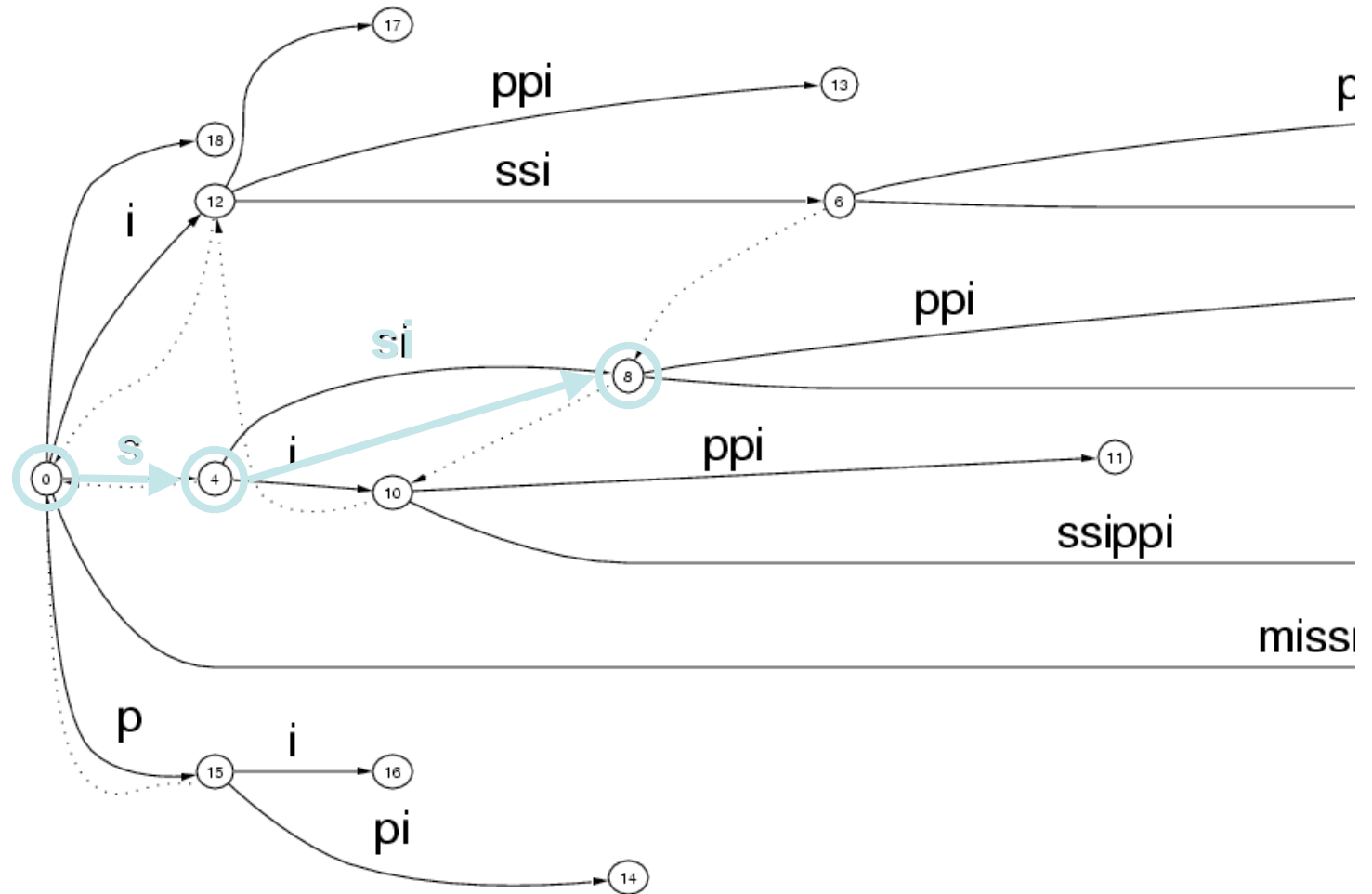
- Find 'ssi' in 'mississippi'

Exact string matching problem

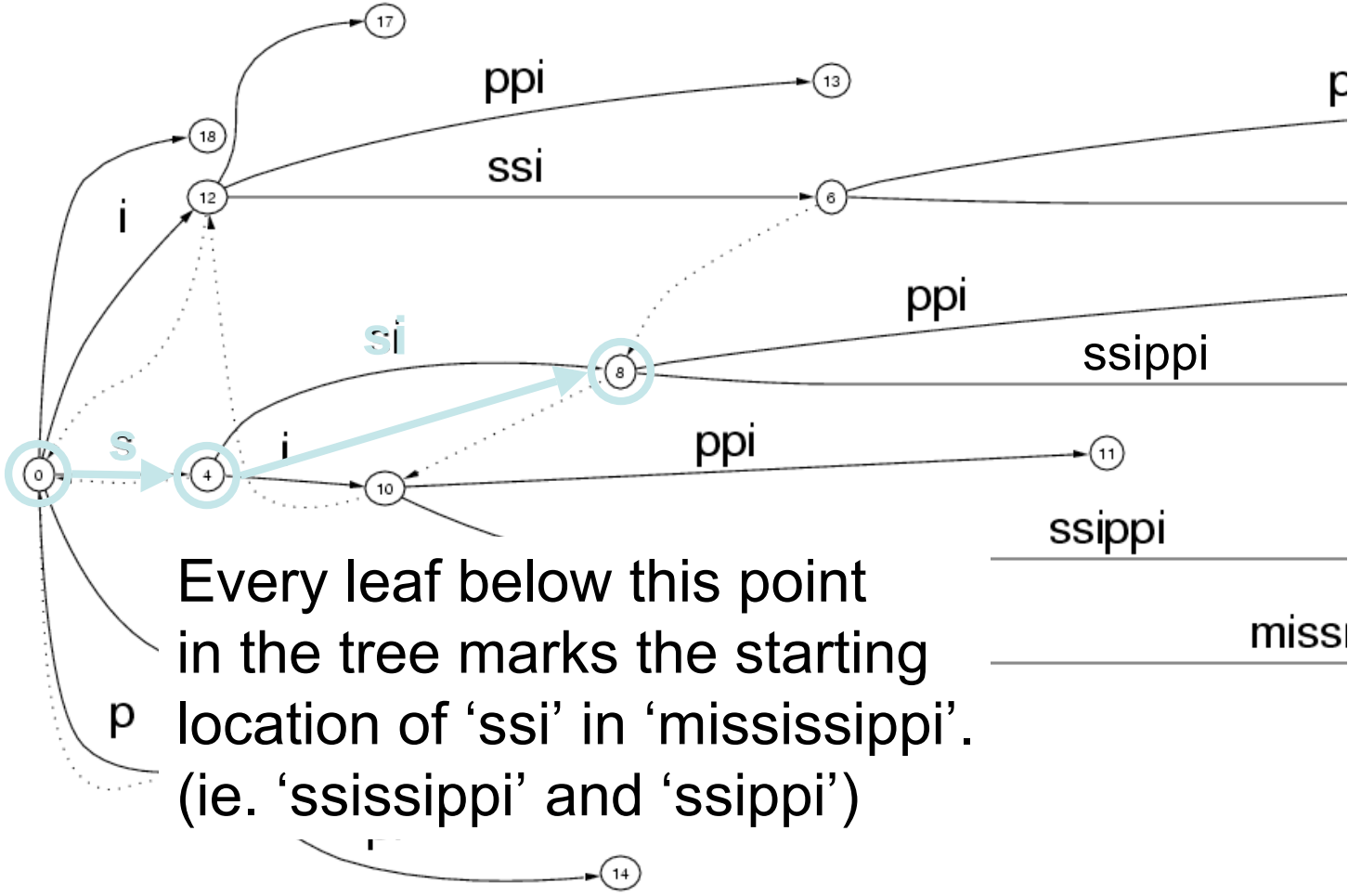
- Find 'ssi' in 'mississippi'



Exact string matching problem



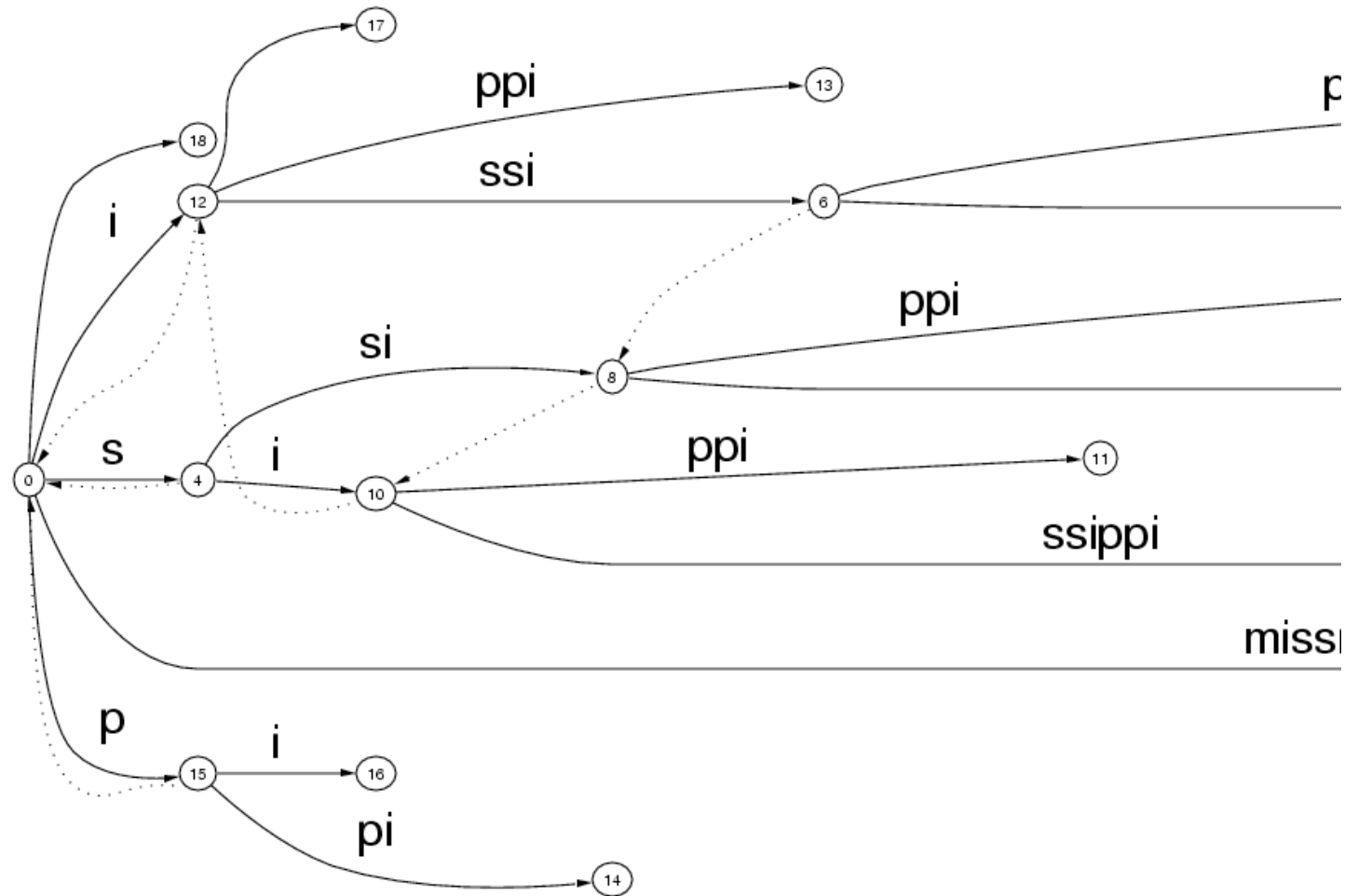
Exact string matching problem



Exact string matching problem

- Find 'sissy' in 'mississippi'

Exact string matching problem



Comparing to the other algorithms

- KMP and Boyer-Moore both achieve this worst case bound.
 - $O(m+n)$ when the text and pattern are presented together.
- Suffix trees are much faster when the text is fixed and known first while the patterns vary.
 - $O(m)$ for single time processing the text, then only $O(n)$ for each new pattern.
- Based on suffix trees, is faster for searching a number of patterns at one time against a single text (**exact set matching problem**)
 - Aho-Corasick algorithm: preprocessing P instead of T.

Building the Suffix Tree

- How do we build a suffix tree?

`while suffixes remain:`

`add next shortest suffix to the tree`

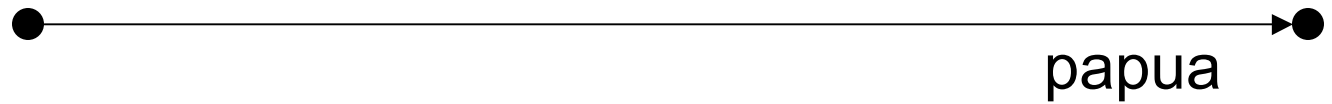
Building the Suffix Tree

- papua



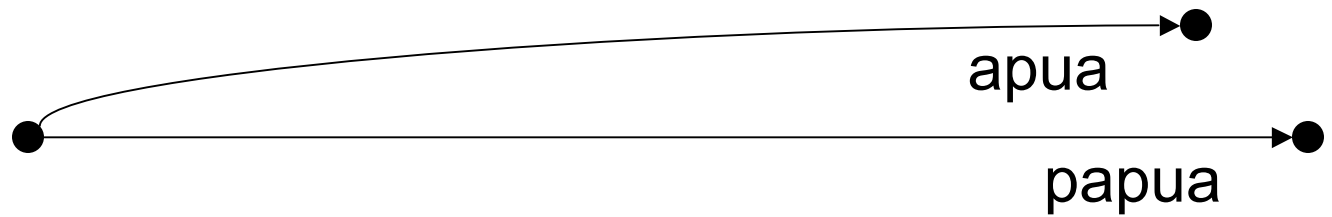
Building the Suffix Tree

- papua



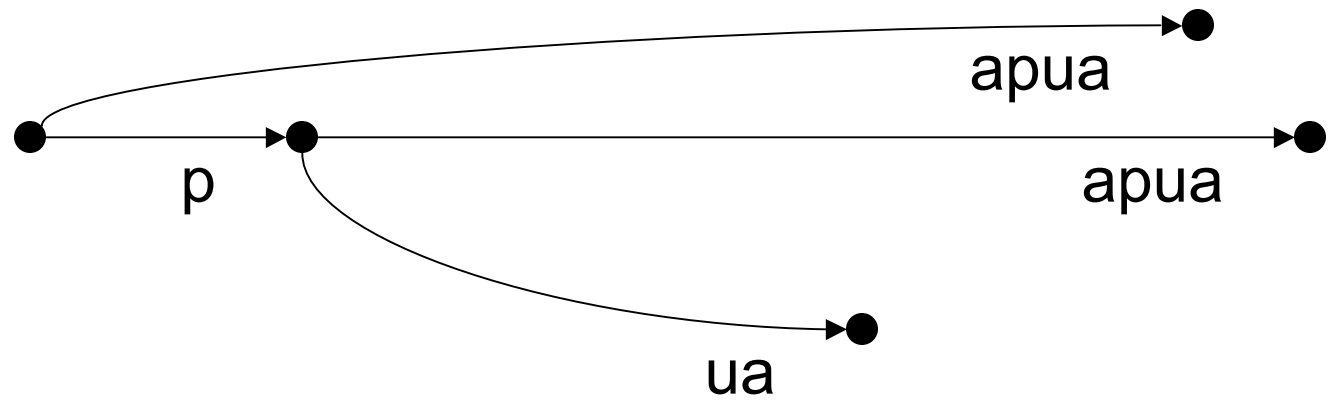
Building the Suffix Tree

- papua



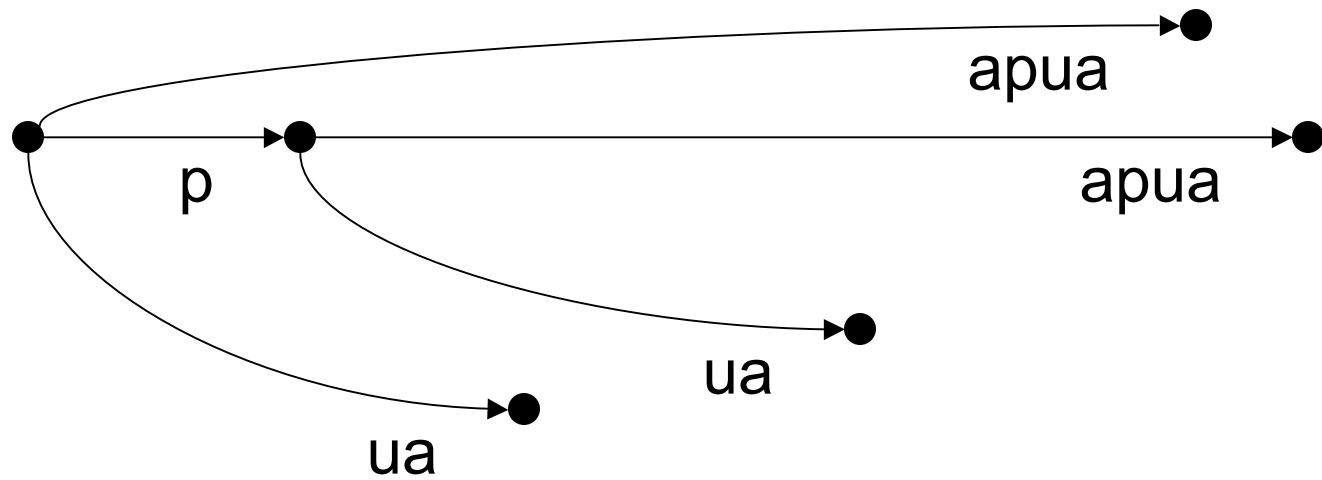
Building the Suffix Tree

- papua



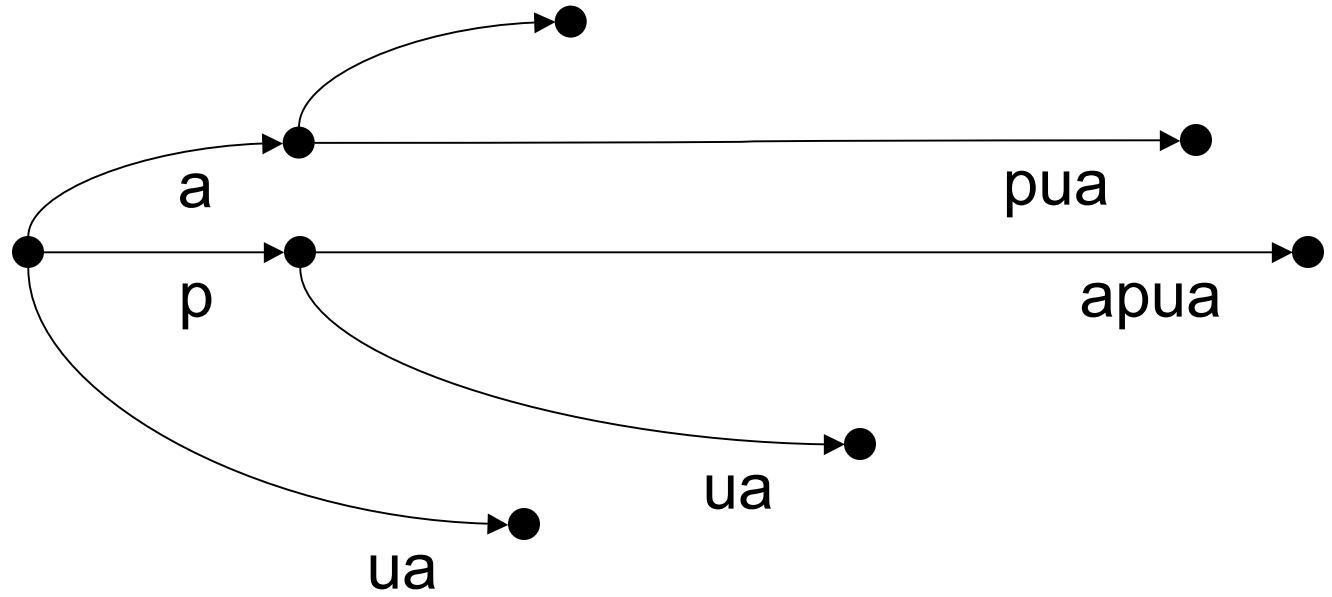
Building the Suffix Tree

- papua



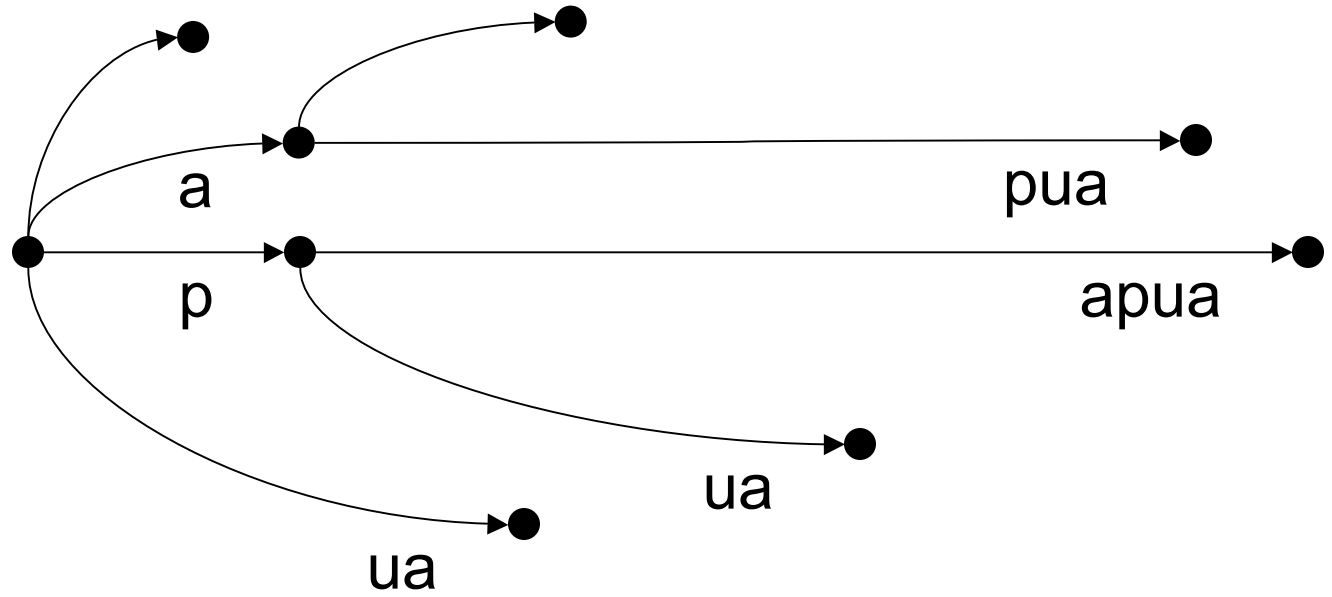
Building the Suffix Tree

- papua



Building the Suffix Tree

- papua



Building the Suffix Tree

- How do we build a suffix tree?

`while suffices remain:`

`add next shortest suffix to the tree`

Naïve method - $O(m^2)$ (m = text size)