Multiple Pattern Matching

Chapters 5 and 7 of Dan Gusfield: *Algorithms on strings, trees, and sequences*

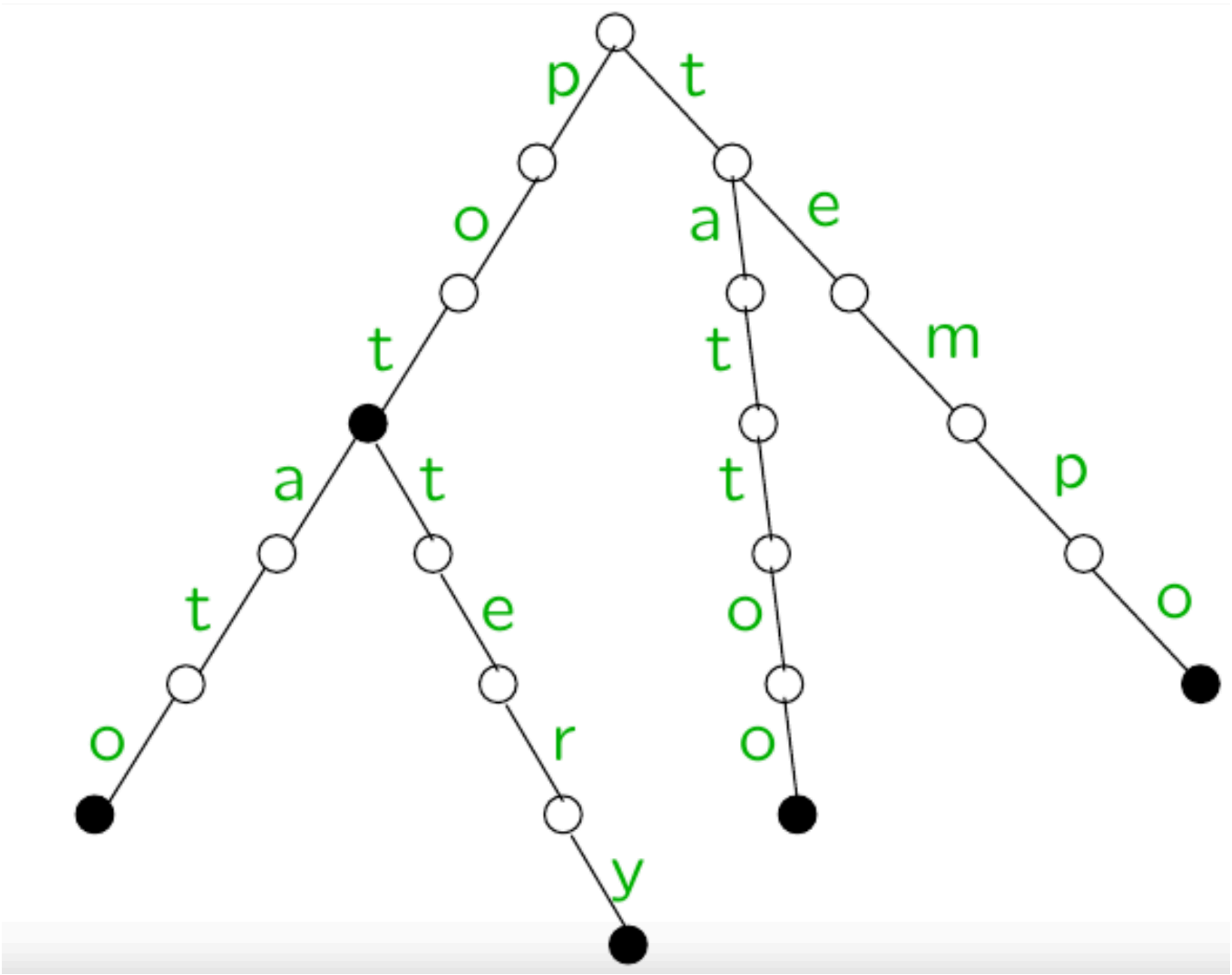
Giulia Bernardini
giulia.bernardini@units.it

Algorithmic Design
a.y. 2022/2023

Tries: an example

Let R={pot, potato, pottery, tattoo, tempo}

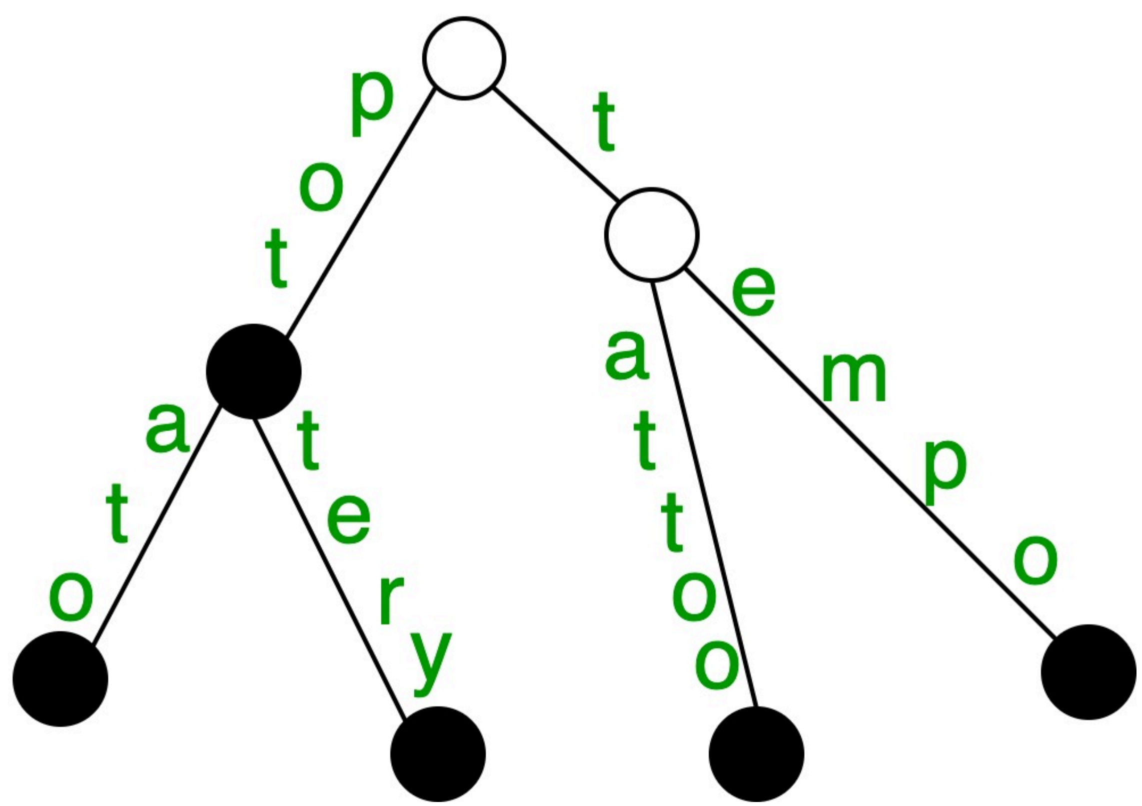
Trie(R) is represented below. Black nodes mark the end of the strings in R.



Tries: an example

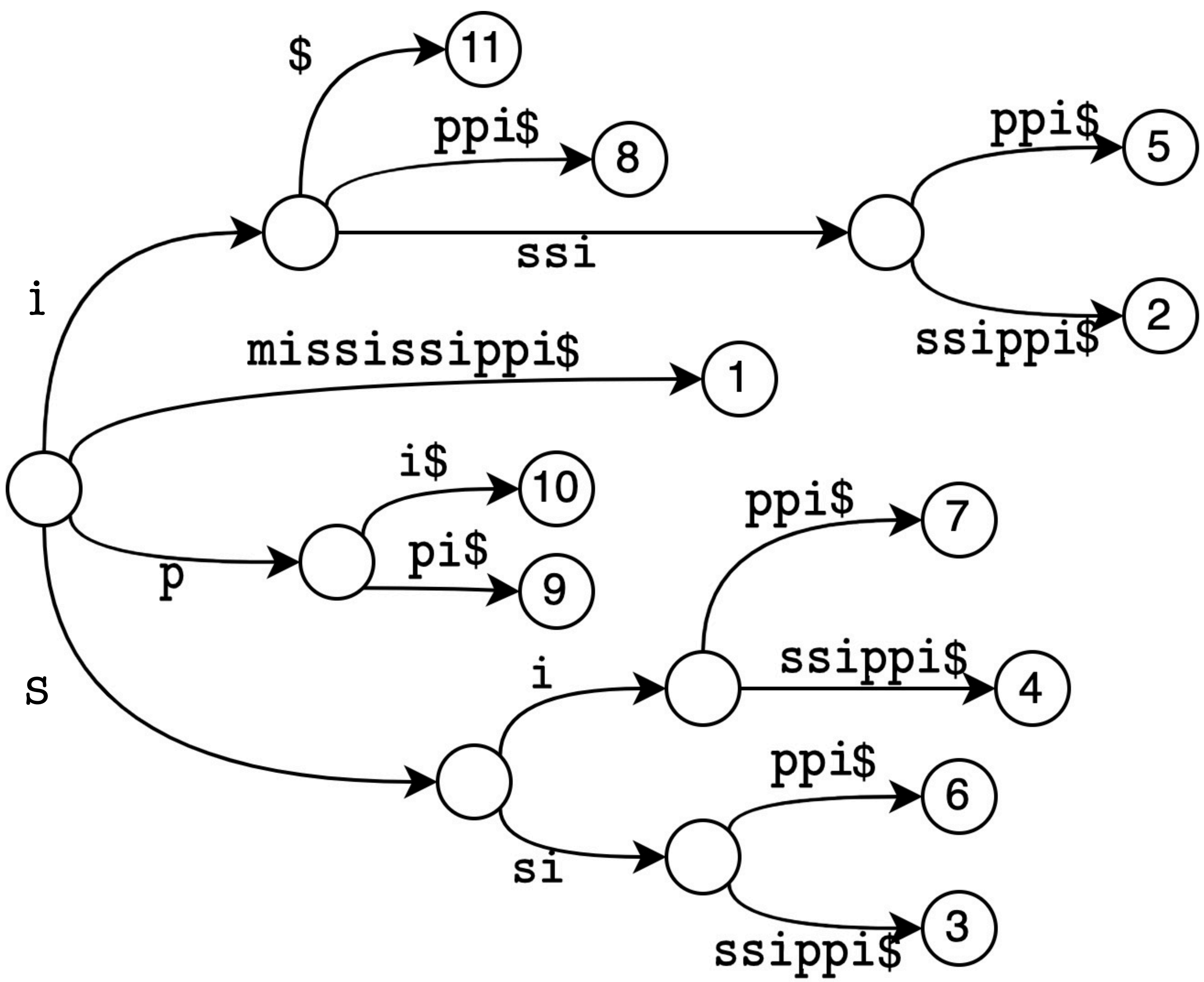
Let R={pot, potato, pottery, tattoo, tempo}

Trie(R) is represented below. Black nodes mark the end of the strings in R. A compacted trie has edges labelled by strings instead of letters, and no nodes with just one child.



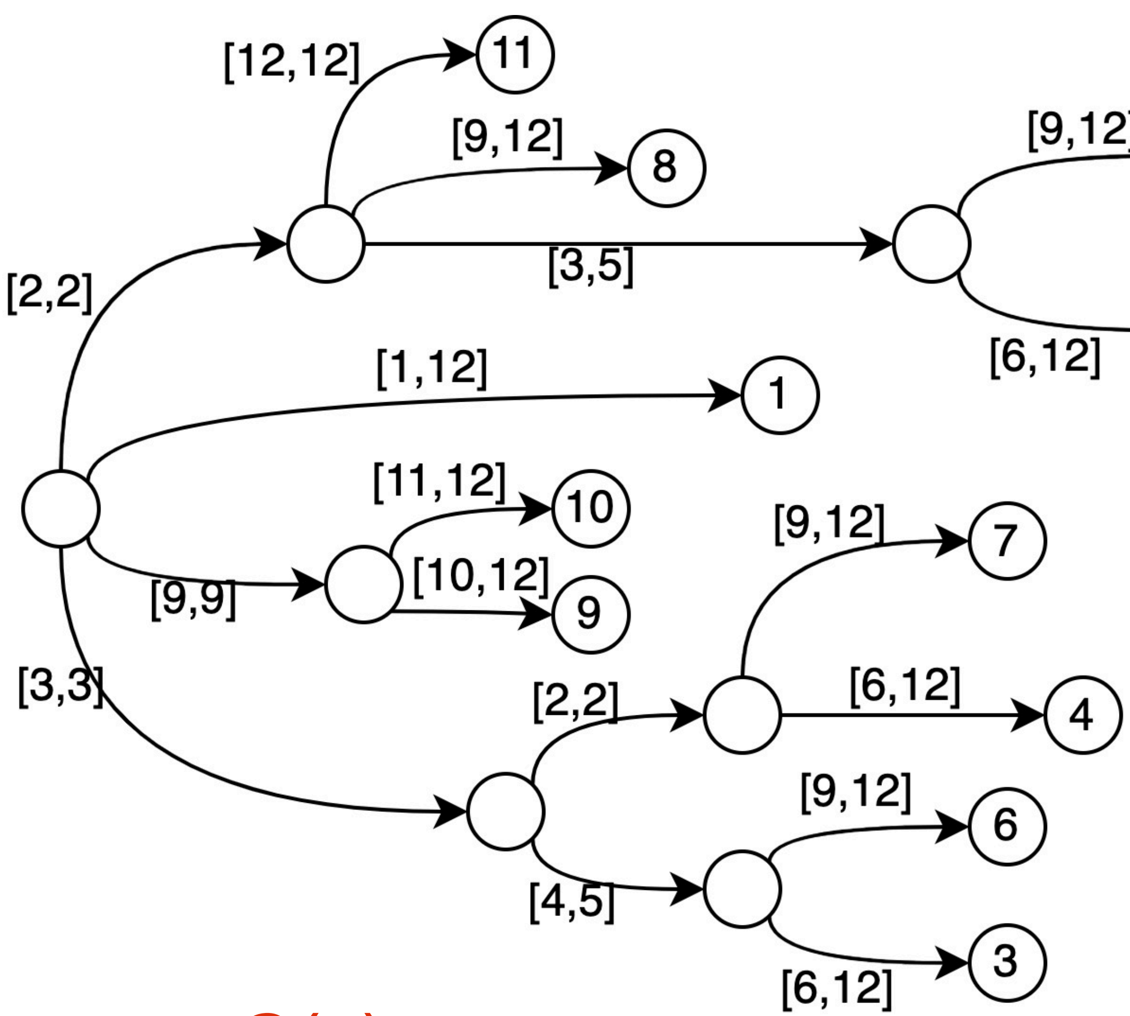
Definition of Suffix Tree

For constructing the suffix tree, it is desirable that all the terminal nodes are leaves. That's why it is standard to add an extra letter $\$ \notin \Sigma$ at the end of the string, and to construct the suffix tree of this extended string. The suffix tree of T=mississippi$ is

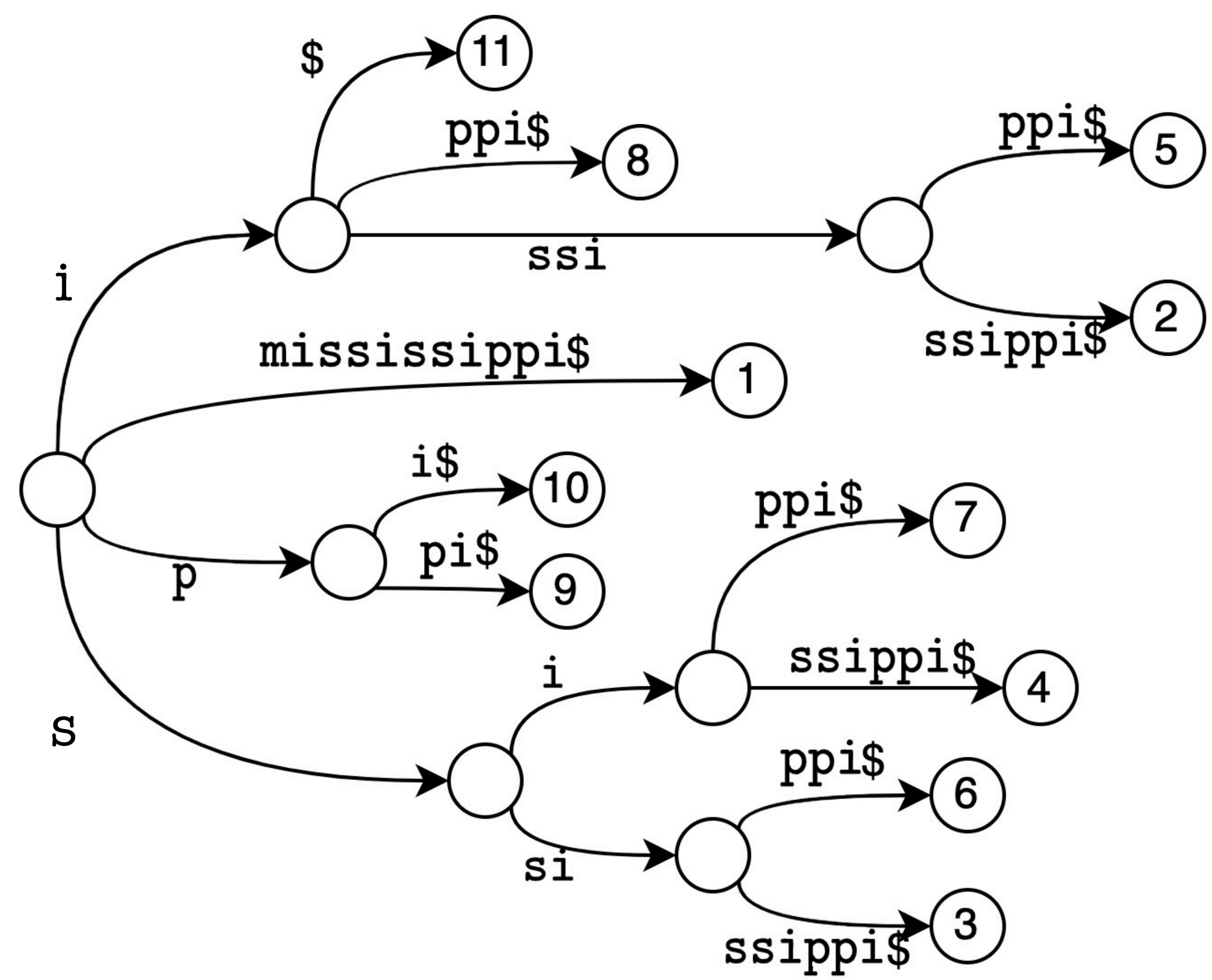


Properties of the Suffix Tree

…but all the strings labelling the edges of the suffix tree of T are substrings of T. Thus each of them can be represented by an interval of positions over T. Representing one such interval requires *O*(1) space, and since the suffix tree has *O*(n) edges (because there are *O*(n) nodes) the whole representation requires *O*(n) space!



O(n) space



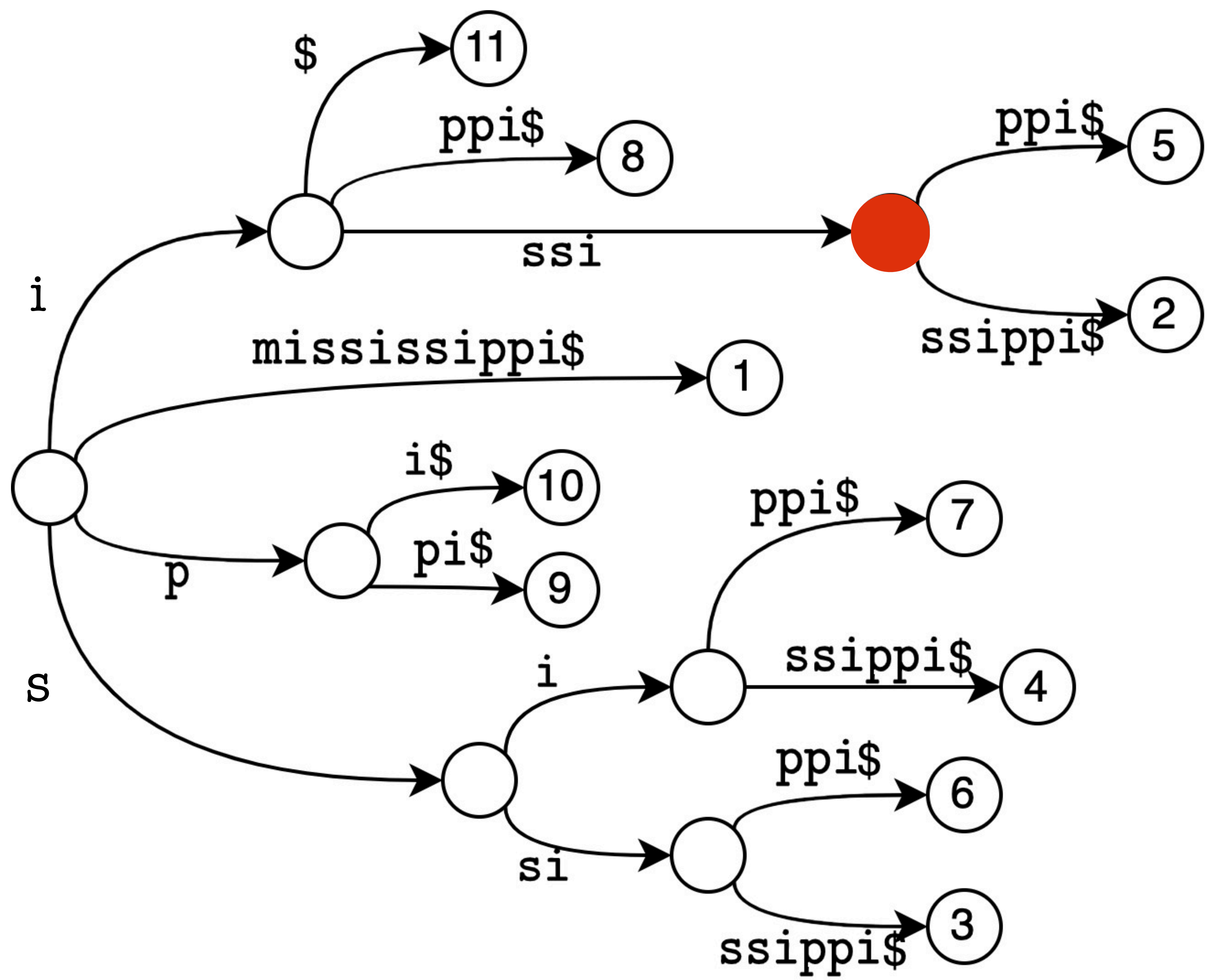
O(n^2) space

Using the Suffix Tree: Longest Repeating Factor

The longest repeating factor of a text T is the longest substring that occurs at least twice in T. It is represented by the deepest branching node in the suffix tree.

The longest repeating factor of T=mississippi$ is “issi”.

Exercise. Write pseudocode for a solution to this problem, and analyse its time complexity.

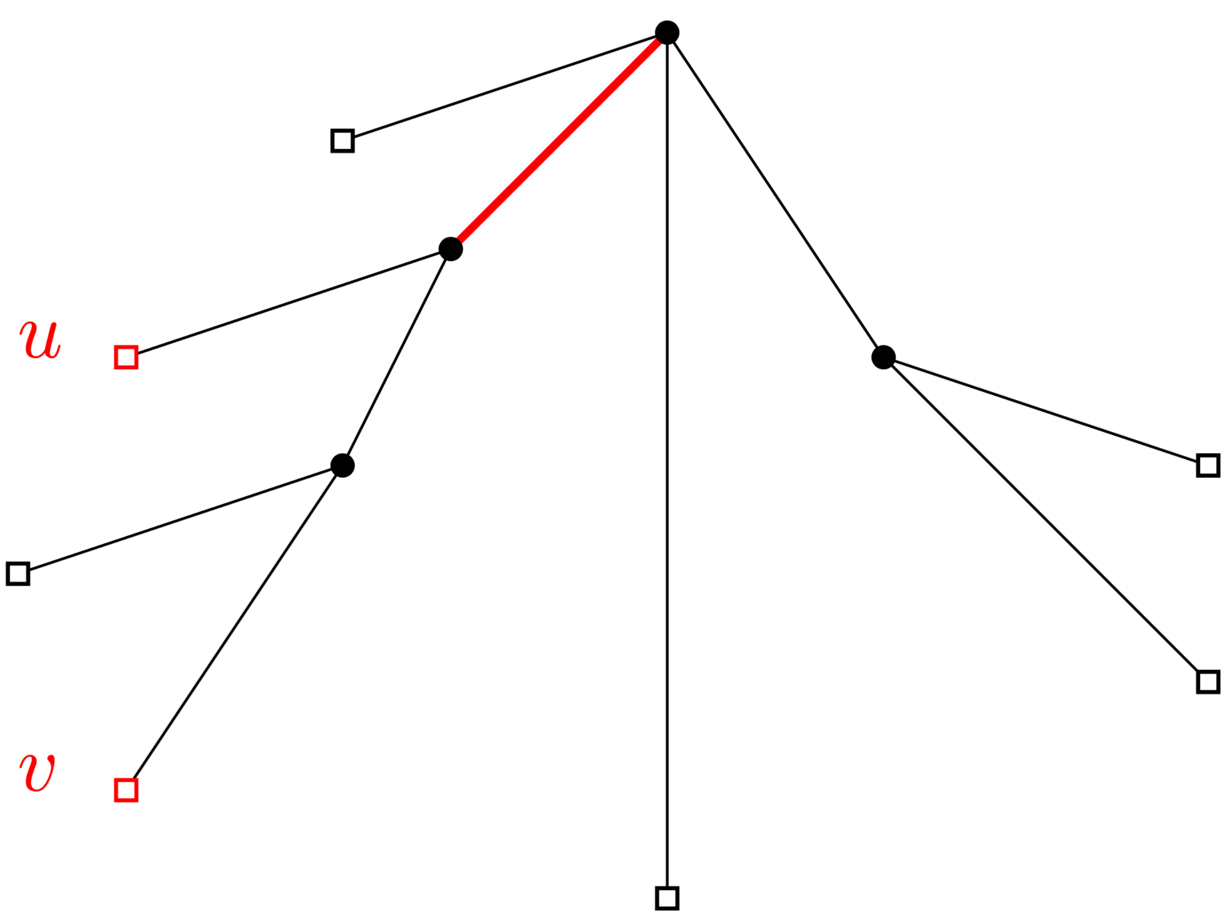


Using the Suffix Tree: Longest Common Prefix

Problem: preprocess a text T of length n so that the following queries can be answered efficiently.

Query: given a pair (i,j), return the longest common prefix of T[i..n] and T[j..n]

The lowest common ancestor (LCA) of two nodes u and v is the deepest node that is an ancestor of both u and v.



Using the Suffix Tree: Longest Common Prefix

Problem: preprocess a text T of length n so that the following queries can be answered efficiently.

Query: given a pair (i,j), return the longest common prefix of T[i..n] and T[j..n]

The lowest common ancestor (LCA) of two nodes u and v is the deepest node that is an ancestor of both u and v.

Theorem (Bender and Farach-Colton). Any tree of size O(N) can be preprocessed in O(N) time so that the LCA of any two nodes can be computed in O(1) time.

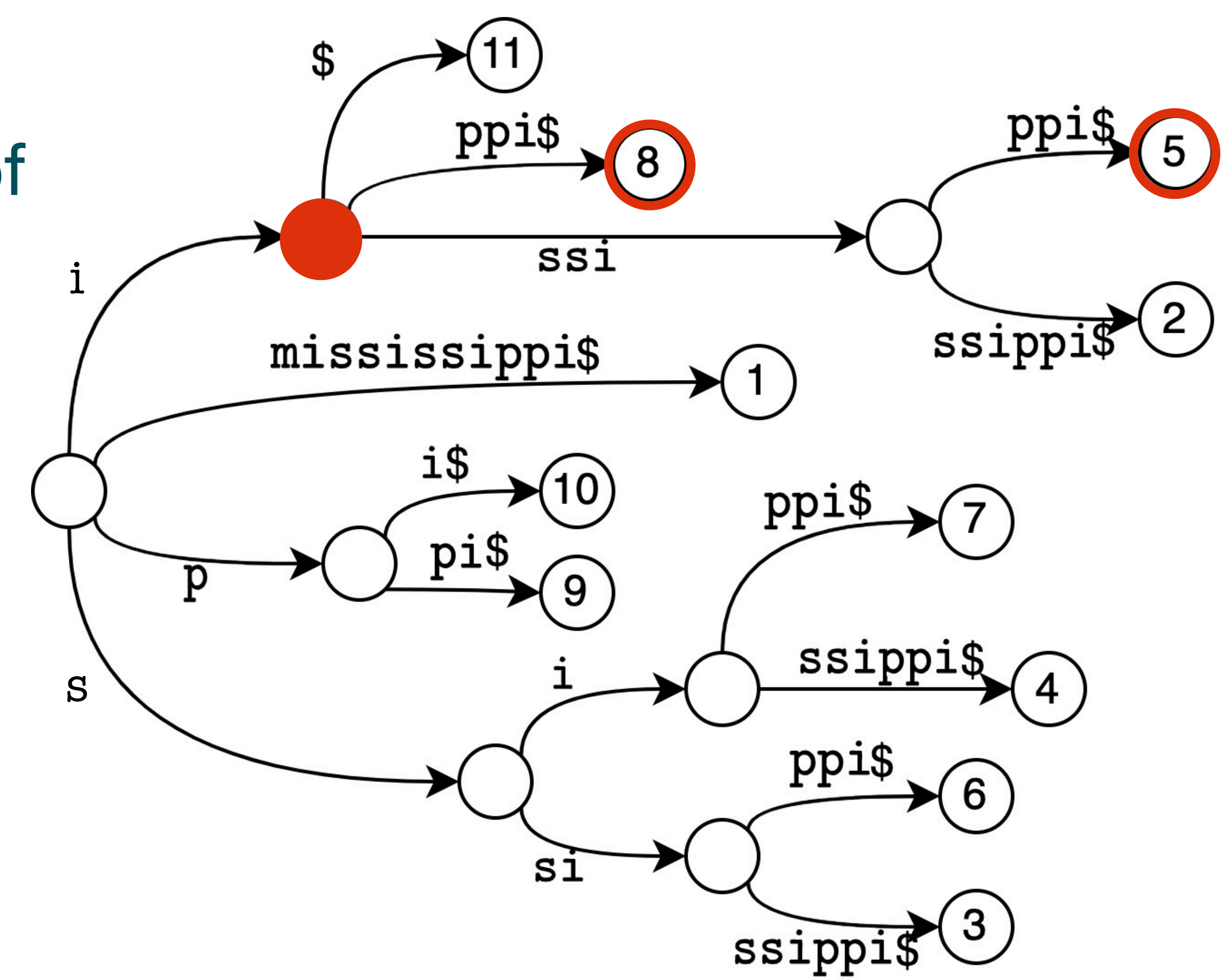
Theorem. Longest Common Prefix queries in T can be answered in O(1) time after O(n) time preprocessing of the suffix tree of T.

Using the Suffix Tree: Longest Common Prefix

Problem: preprocess a text T of length n so that the following queries can be answered efficiently.

Query: given a pair (i,j), return the longest common prefix of T[i..n] and T[j..n]

For T=mississippi$, let (5,8) be the query. The answer is "i", which is the path label of the LCA of leaves 5 and 8.



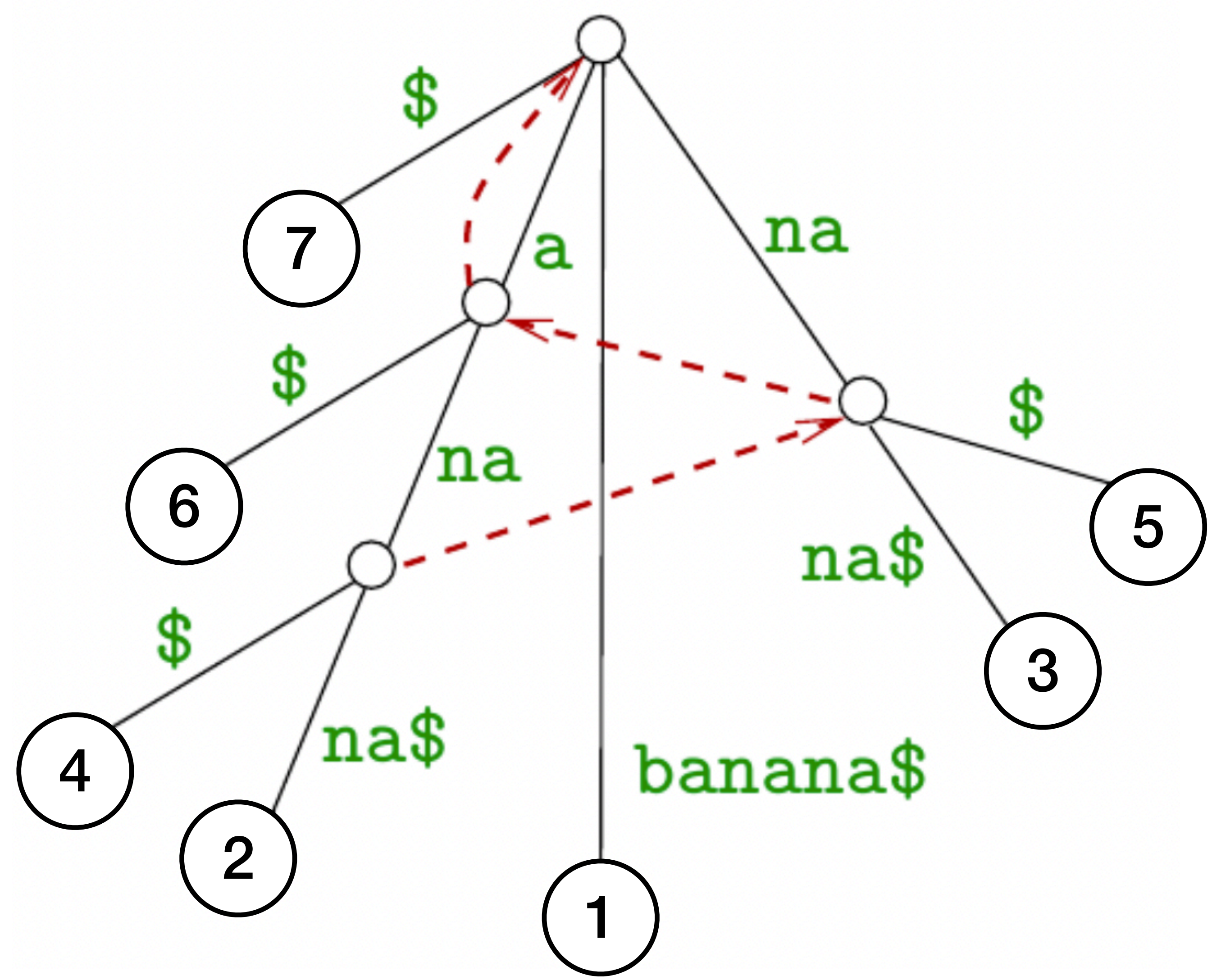
Suffix links

The key to efficient suffix tree construction are suffix links:

For an explicit node u, slink(u) is the node v such that S_v is the longest proper suffix of S_u, i.e., if $S_u = T[i..j]$ then $S_v = T[i+1..j]$.

For example, let T = banana$.

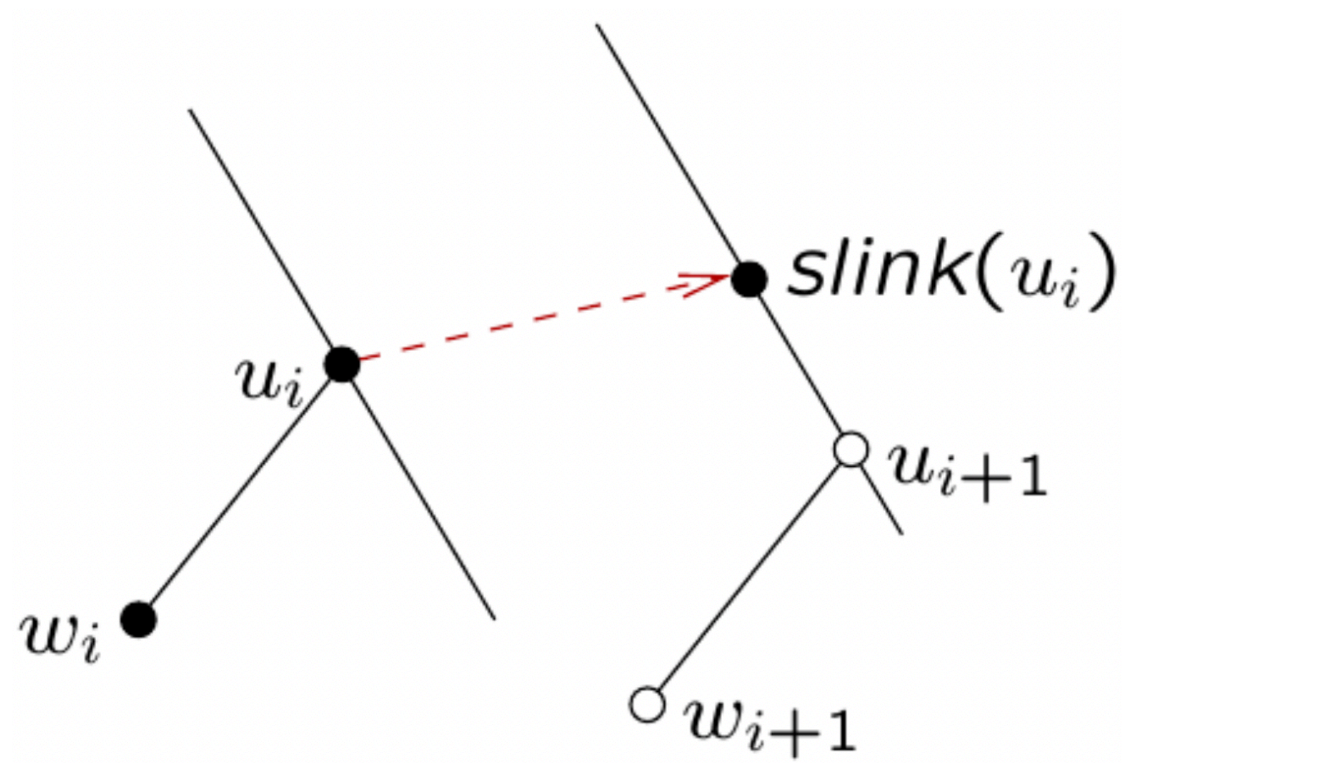
The suffix links are represented by the red arrows.



McCreight's Construction Algorithm

McCreight's suffix tree construction is a simple modification of the brute force algorithm that computes the suffix links during the construction and uses them as shortcuts.

Say we have just added a leaf w_i representing the suffix T_i as a child to a node u_i. The next step is to add w_{i+1} as a child to a node u_{i+1}. The brute force algorithm finds u_{i+1} by traversing the partially constructed suffix tree from the root; McCreight's algorithm takes a shortcut to slink(u_i). This is safe because slink(u_i) represents a prefix of T_{i+1}!



Generalised Suffix Tree for a Set of Strings

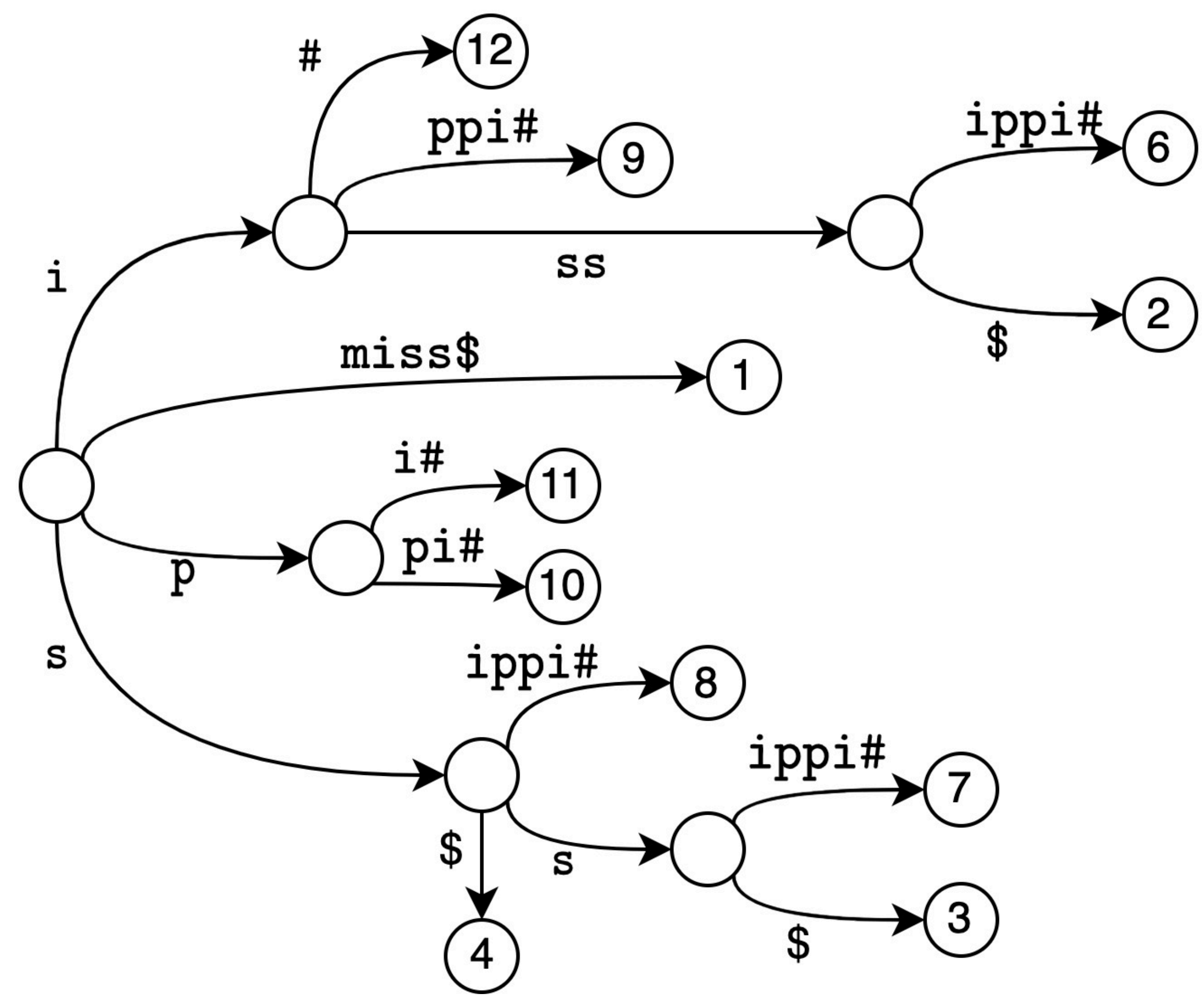
The concept of suffix tree of a string can be easily extended to a set of strings.

The generalised suffix tree of a set of strings $S_1, S_2, \ldots, S_k$ is the compacted trie of all the suffixes of all the strings in the set.

To build it, it suffices to build the suffix tree of their concatenation $S_1\$_1S_2\$_2\ldots S_k\$_k$, where $\$_1, \$_2, \ldots, \$_k$ are distinct terminal symbols.

S_1=miss$

S_2=issippi#

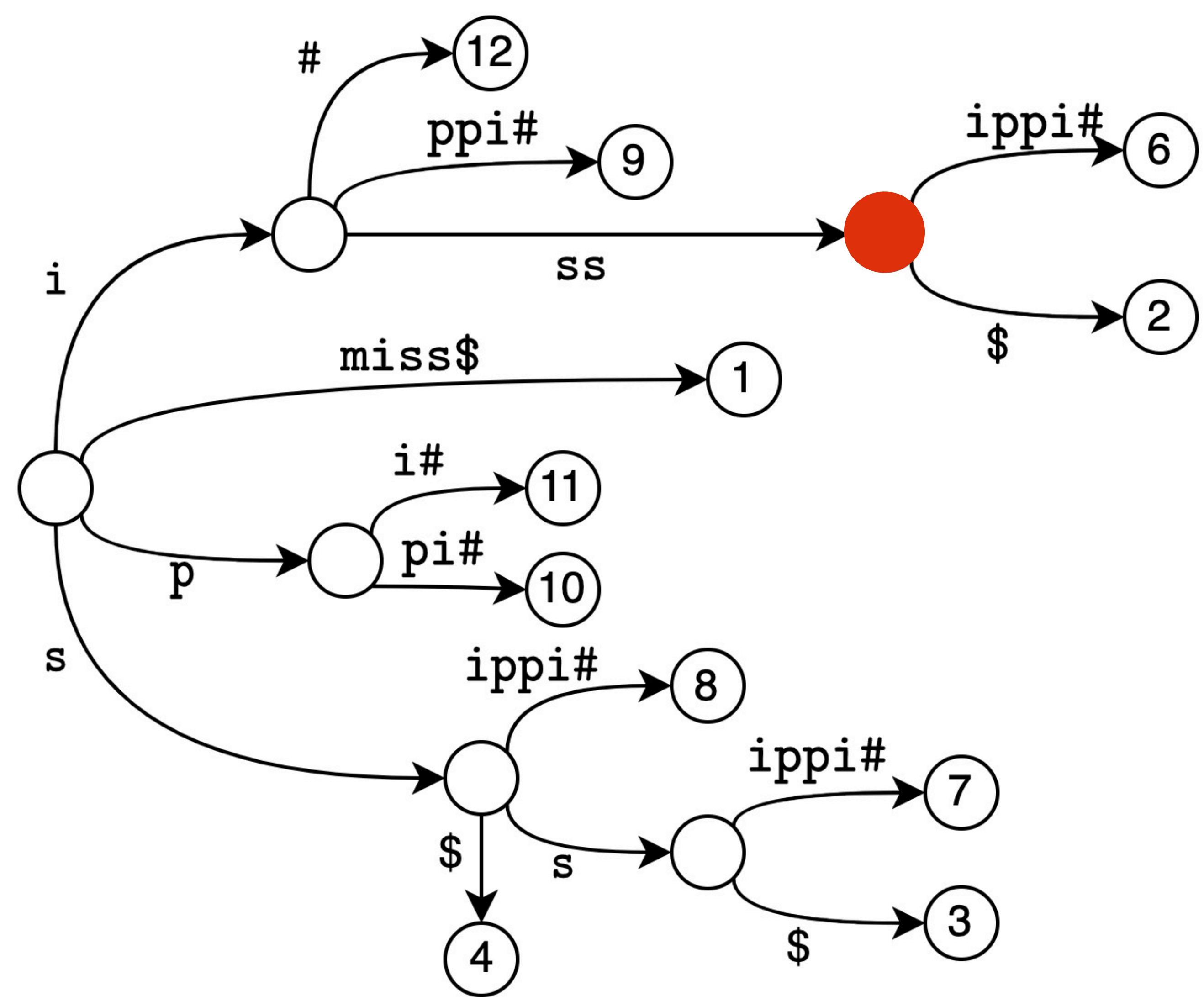


Use of the GST: Longest Common Substring

The Longest Common Substring (LCS) of two strings S and T is the longest substring that occurs both in S and in T.

It is represented by the deepest branching node in the suffix tree that have at least a descending leaf corresponding to S and at least a descending leaf corresponding to T.

The LCS of “miss” and “issipi” is “iss”



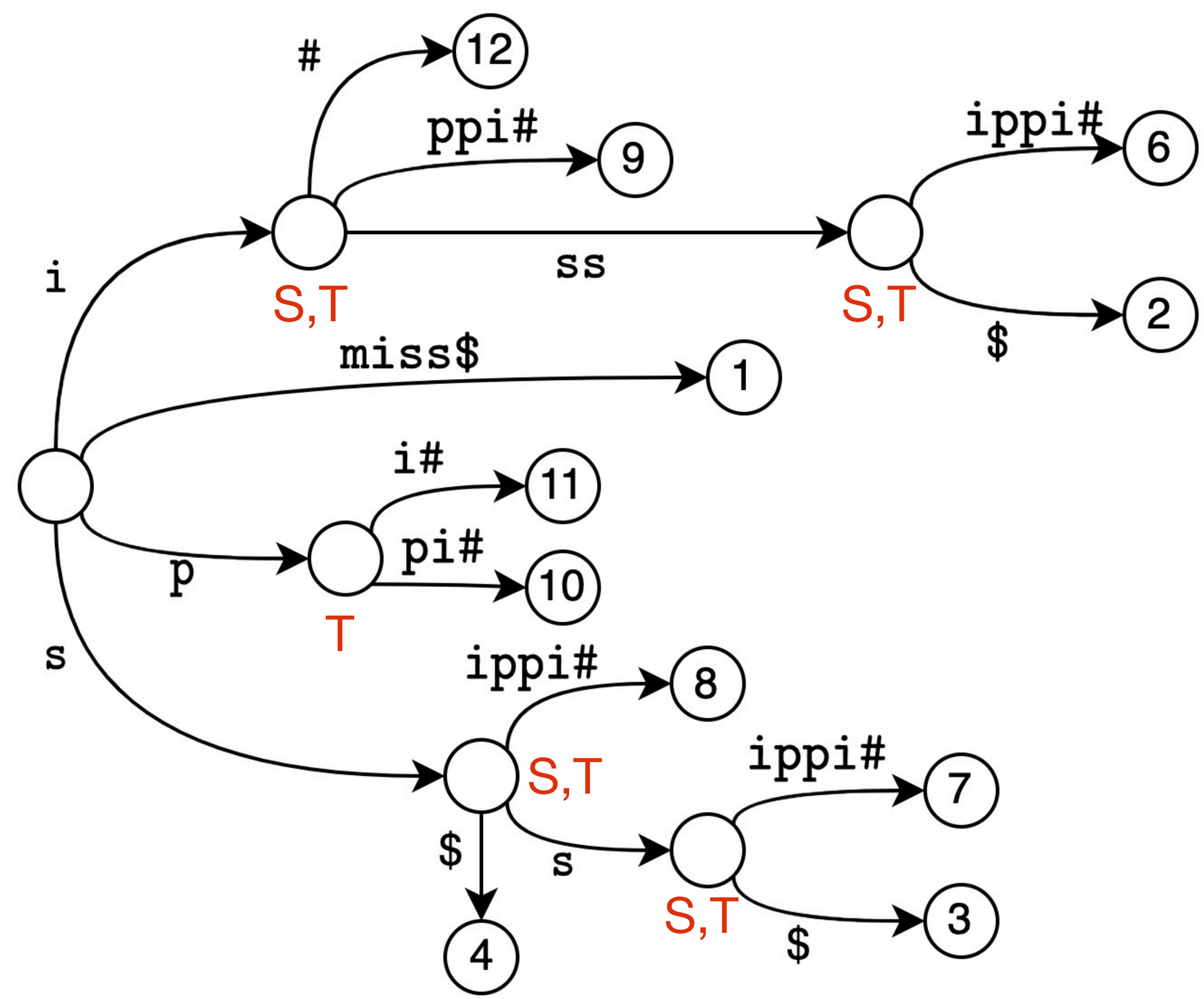
Use of the GST: Longest Common Substring

The LCS of S and T can be found in O(|S|+|T|) time by:

- preprocessing the GST of S and T to mark each branching node with the strings corresponding to the leaves descending from there. This can be done traversing the GST bottom-up.
- Picking the deepest node marked with both S and T. This can be done with a DFS.

S=miss$

T=issipi#



Generalised Suffix Tree for a Set of Strings

Building the suffix tree of $S_1\$_1S_2\$_2 \ldots S_k\$_k$, requires time linear in the sum of the lengths of the strings in the set.

The suffix tree built in this way, though, contains also spurious substrings that span more than one input string.

For example, the concatenation miss$issippi# contains the substring ss$issippi#.

However, because each terminal symbol is distinct and is not in any of the original strings, the label on any path from the root to a branching node must be a substring of one of the original strings.

To remove these spurious substrings it suffices to truncate the labels of the branches ending at the leaves to the first terminal symbol.