Algorithm of Suffix Tree

by farseerfc@gmail.com

Jiachen Yang[1]

[1]Research Student in Osaka University

November 10, 2011

Part Outline

1. What is Suffix Tree
2. History & Naïve Algorithm
3. Optimization of Naïve Algorithm

What can we do with suffix tree?

Linear algorithms for exact string matching

- like KMP

Search for strings

1. Check if a string P of length m is a substring in $O(m)$ time.
2. Find all z occurrences of the patterns $P_1, \cdots, P_q$ of total length m as substrings in $O(m+z)$ time.
3. Search for a regular expression P in time expected sublinear in n .
4. Find for each suffix of a pattern P, the length of the longest match between a prefix of $P[i \dots m]$ and a substring in D in $\theta(m)$ time .
5. ...

Find properties of the strings

1. Find the longest common substrings of the string S_i and S_j in $\theta(n_i+n_j)$ time .
2. Find all maximal pairs, maximal repeats or supermaximal repeats in $\theta(n+z)$ time, if there are z such repeats .
3. Find the Lempel-Ziv decomposition in $\theta(n)$ time .
4. Find the longest repeated substrings in $\theta(n)$ time.
5. Find the most frequently occurring substrings of a minimum length in $\theta(n)$ time.
6. Find the shortest strings from $\sum$ that do not occur in D, in $O(n+z)$ time, if there are z such strings.
7. Find the shortest substrings occurring only once in $\theta(n)$ time.
8. Find, for each i, the shortest substrings of S_i not occurring elsewhere in D in $\theta(n)$ time.
9. ...

Trie, Radix Tree, Suffix Trie & Suffix Tree

trie[1] (AKA prefix tree) is a dictionary tree.

- stores a set of words.
- each node represents a character except that root is empty string.
- words with common prefix share same parent nodes.
- is a minimal deterministic finite automaton that accepts all words.

radix tree (AKA patricia trie or radix trie) is a trie with compressed chain of nodes.

- Each internal node has at least 2 children.

suffix trie is a trie which stores all suffix of a given string.

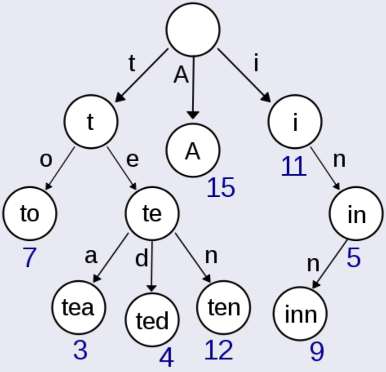
suffix tree is a suffix radix tree.

- that enables linear time construction and fast algorithms of other problems on a string.

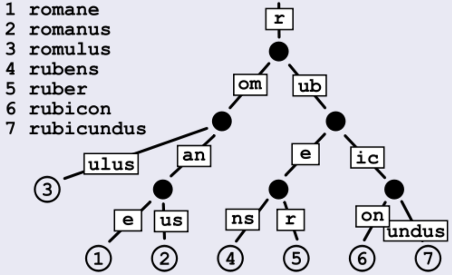
[1] pronounced as in word retrieval by its inventor, /triː/ “tree”, but pronounced /traɪ/ “try” by other authors

Trie & Radix Tree

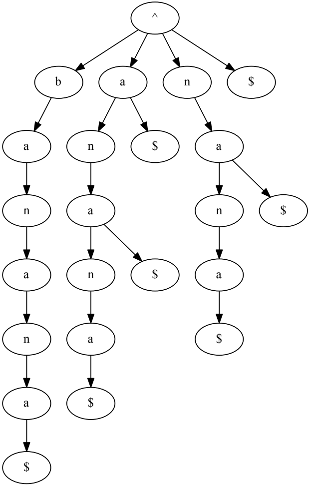
Trie of “A to tea ted ten i in inn”

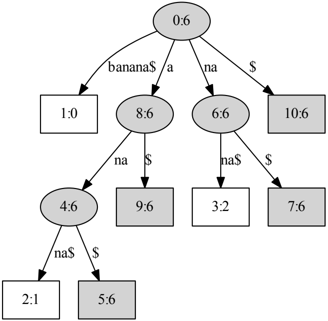


Radix tree example

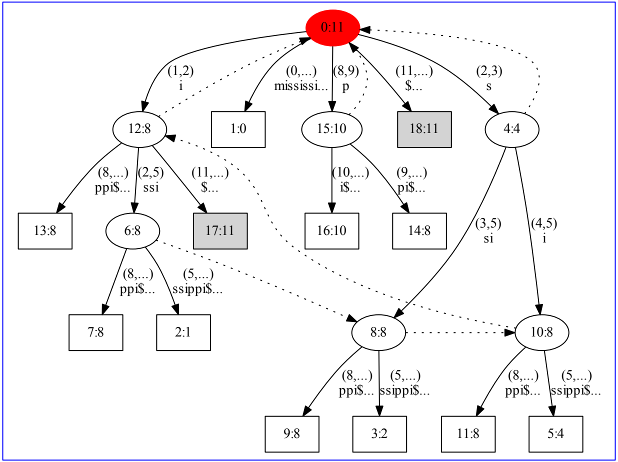


Suffix Trie & Suffix Tree of "banana"





Suffix Tree of “mississippi”



Part Outline

History of Suffix Tree Algorithms

- First linear algorithm was introduced by Weiner 1973 as position tree. Awarded by Donald Knuth as "Algorithm of the year 1973".
- Greatly simplified by McCreight 1976.

Above two algorithms are processing string backward.

- First online construction by Ukkonen 1995, which is easier to understand.

Above algorithms assume size of alphabet as fixed constant .

- Limitation was break by Farach 1997, optimal for all alphabets.

Further study are continued to scale to scenarios when the whole suffix tree or even input string cannot fit into memory.

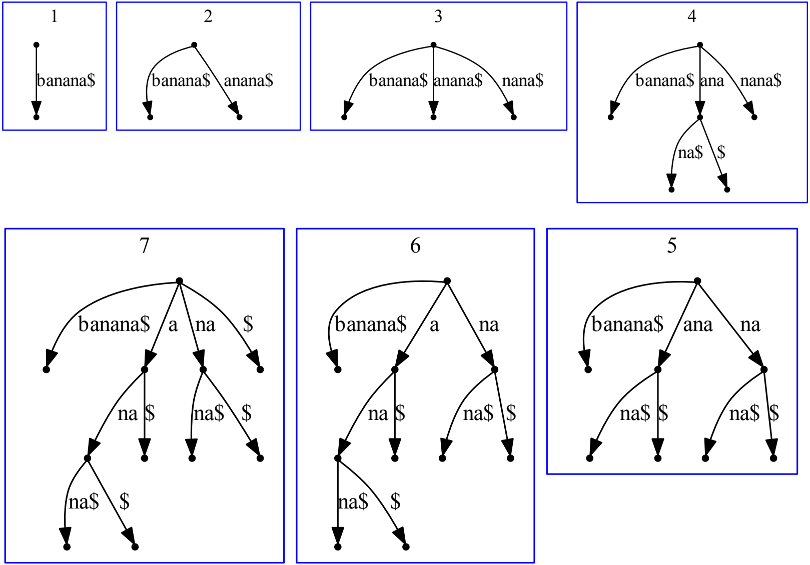
M. Farach. Optimal suffix tree construction with large alphabets. In *focs*, page 137. Published by the IEEE Computer Society, 1997.

E. M. McCreight. A space-economical suffix tree construction algorithm. *J. ACM*, 23:262–272, April 1976. ISSN 0004-5411. doi: http://doi.acm.org/10.1145/321941.321946. URL `http://doi.acm.org/10.1145/321941.321946`.

E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14:249–260, 1995. ISSN 0178-4617. URL `http://dx.doi.org/10.1007/BF01206331`. 10.1007/BF01206331.

P. Weiner. Linear pattern matching algorithms. In *Switching and Automata Theory, 1973. SWAT '08. IEEE Conference Record of 14th Annual Symposium on*, pages 1 –11, oct. 1973. doi: 10.1109/SWAT.1973.13.

Backward Construction of Suffix Tree



Construct Suffix Tree by Sorting Suffix

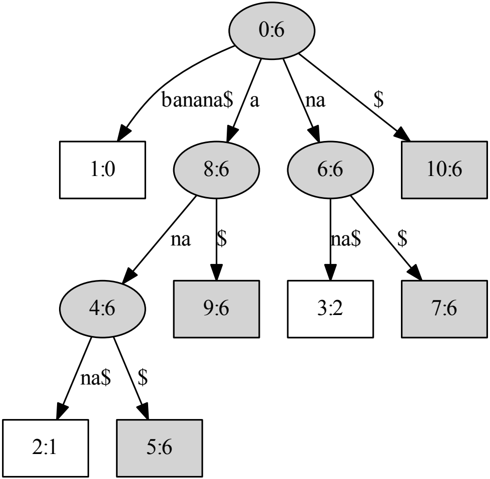
Suffix : | Sorted suffix : | Tree of sorted suffix :

```
mississippi   i             |-i->|-
ississippi    ippi          |    |-ppi
ssissippi     issippi       |    |-ssi->|-ppi
sissippi      ississippi    |           |-ssippi
issippi       mississippi   |-mississippi
ssippi        pi            |-p->|-i
sippi         ppi           |    |-pi
ippi          sippi         |-s->|-i-->|-ppi
ppi           sissippi           |     |-ssippi
pi            ssippi             |-si->|-ppi
i             ssissippi                |-ssippi
```

Time complexity will be $O(N^2 \log N)$.
Space complexity will be $O(N^2)$.

Properties of Suffix Tree

1. Each update will add exactly one leaf node to the tree.
   - $nr\_leaf = N$
2. Suffix tree is full tree. Each internal node has at least 2 children.
   - $nr\_internal < N$
   - $nr\_node < 2N$



Naïve Algorithm

```
SUFFIXTREE(string)
    for i ← 1 to len(string)
        do UPDATE(tree_i)
```

```
UPDATE(tree_i)
    for j ← 1 to i
        do node ← tree_i.FIND(suffix[j to i])
           EXTEND(node, char[i])
```

Time complexity will be $O(N^2 \log N)$.
Space complexity will be $O(N^2)$.
The challenge is to make sure $tree_i$ is updated to $tree_{i+1}$ efficiently.

Suffix Extend Rules of Naïve Algorithm

Rule I If path $S_{j \ldots i}$ ends at leaf, append a char S_{i+1} to end of edge into leaf.

$S_{j \ldots i} = \ldots na$ $\qquad$ $S_{j \ldots i+1} = \ldots nan$

Rule II If path $S_{j \ldots i}$ ends in the middle of an edge , and next char S_{i+1} is not equal to the next char in the edge, split that edge, create a internal node, add a new edge to a new leaf.

$S_{j \ldots i} = \ldots na$ $\qquad$ $S_{j \ldots i+1} = \ldots nay$

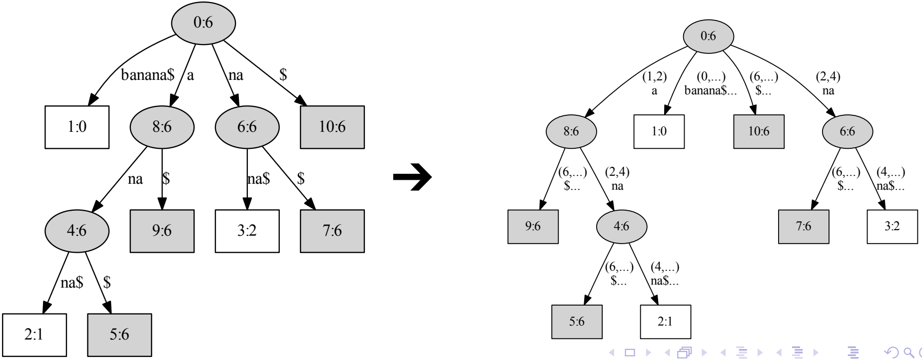
Rule III If path $S_{j \ldots i}$ ends in the middle of an edge , and next char S_{i+1} is equal to the next char in the edge, do nothing, extend has done.

$S_{j \ldots i} = \ldots na$ $\qquad$ $S_{j \ldots i+1} = \ldots nan$

Part Outline

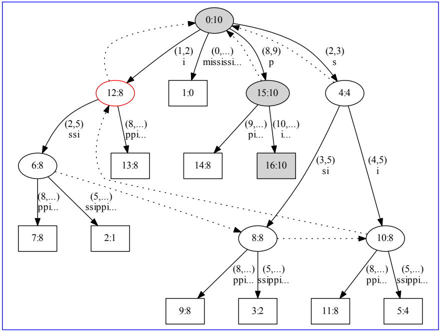
Optimization of Naïve Algorithm

1. Substring can be represented as (start, end) pair
   - Reduce space complexity to $O(N)$ if size of alphabet is fixed constant.
2. Once a leaf, Always a leaf
   - Represent edge that links to a leaf as (start, $\cdots$).
   - Extend leaf nodes for free. We do not need Extend Rule I.

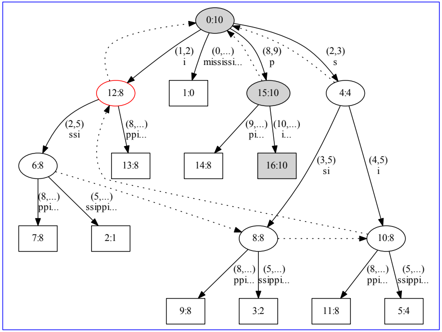


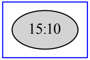
Active Point

Algorithm of Constructing Suffix Tree



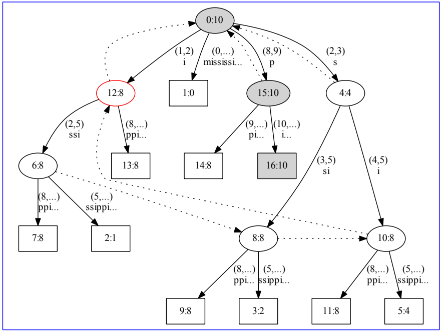
Algorithm of Constructing Suffix Tree

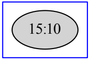


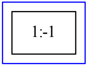
- 

Node (id:generation)

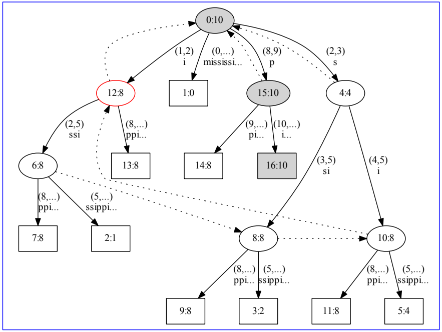
Algorithm of Constructing Suffix Tree

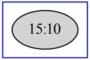


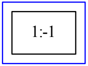
-  Node (id:generation)

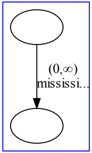
-  Leaf

Algorithm of Constructing Suffix Tree



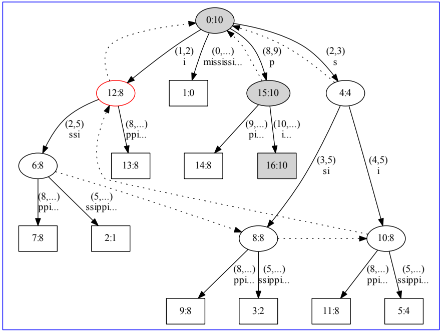
- 

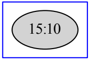
Node (id:generation)
- 

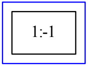
Leaf
- 

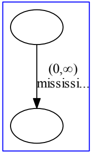
Child-relation

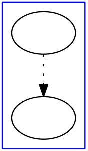
Algorithm of Constructing Suffix Tree



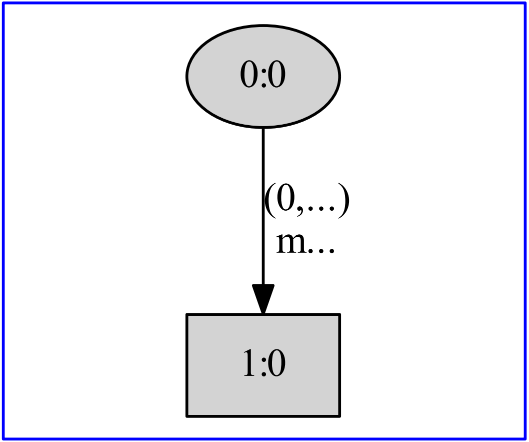
- 

Node (id:generation)
- 

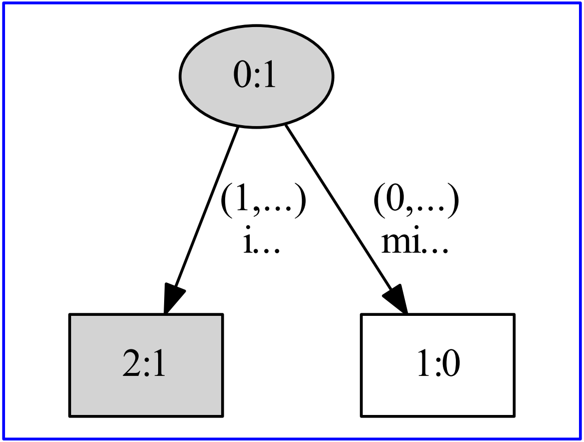
Leaf
- 

Child-relation
- 
suffix-link relation

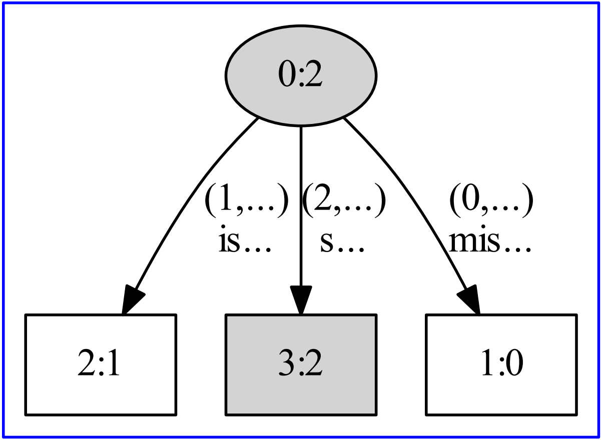
Experiment – mississippi



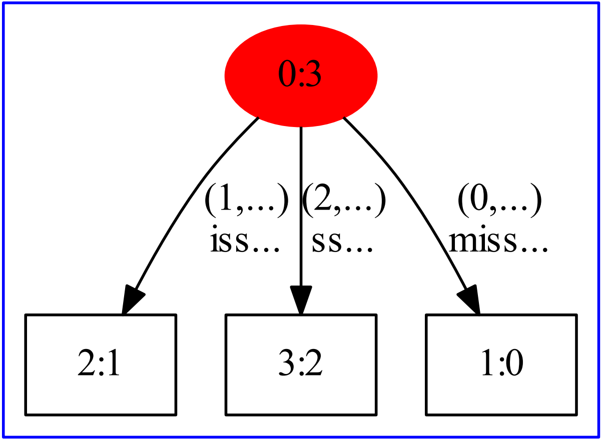
Experiment – mississippi



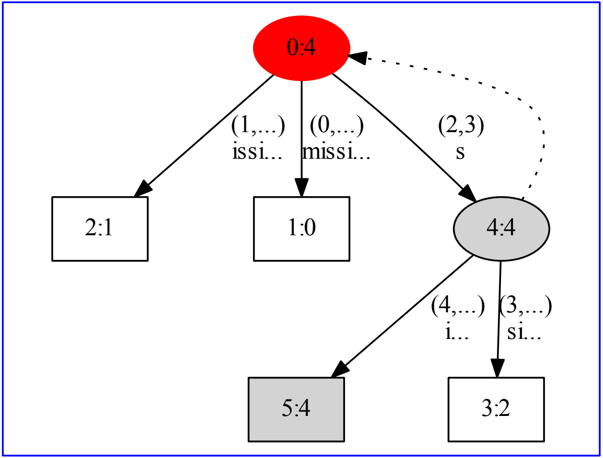
Experiment – mississippi



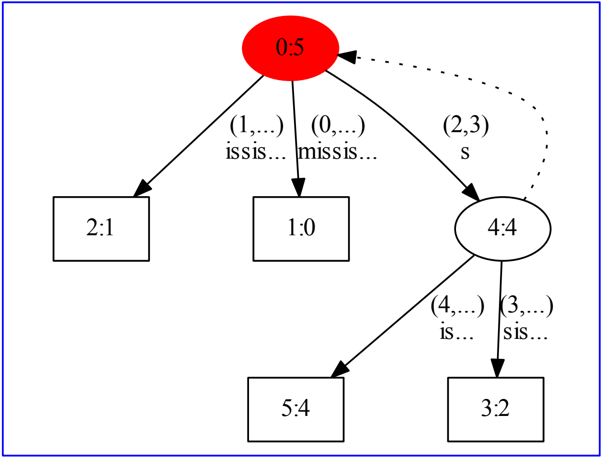
Experiment – mississippi



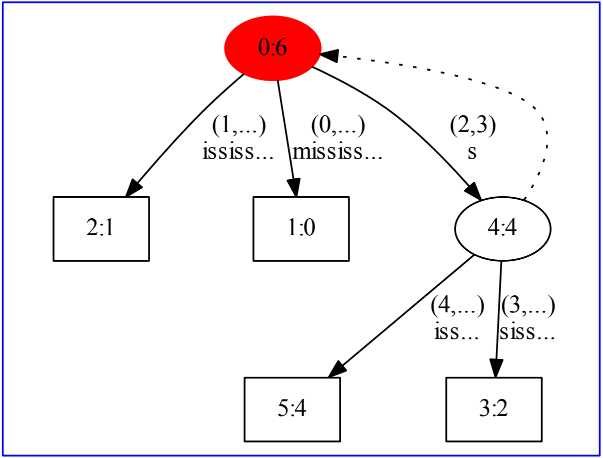
Experiment – mississippi



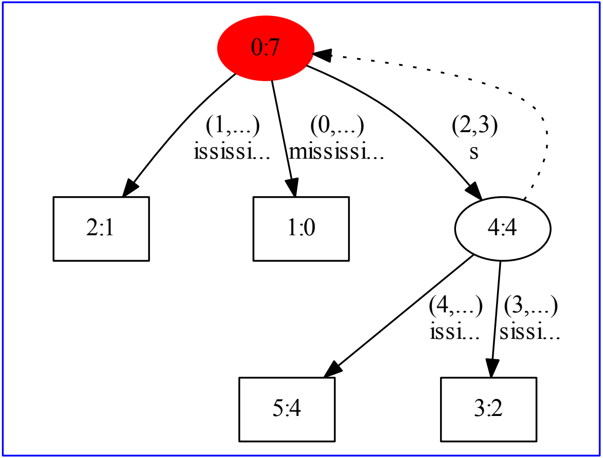
Experiment – mississippi



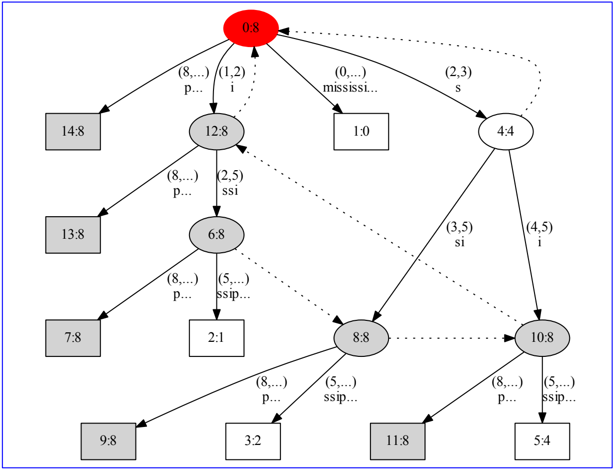
Experiment – mississippi



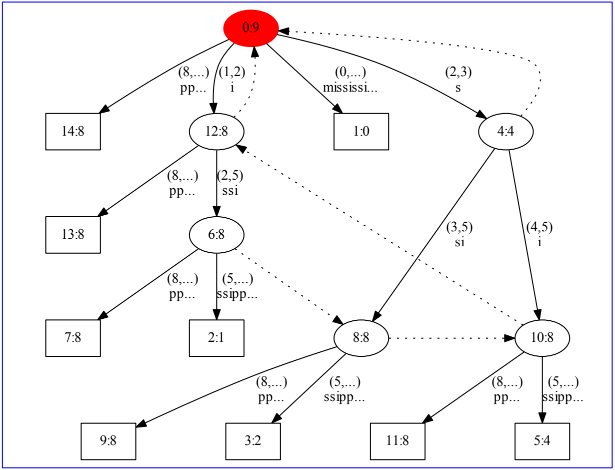
Experiment – mississippi



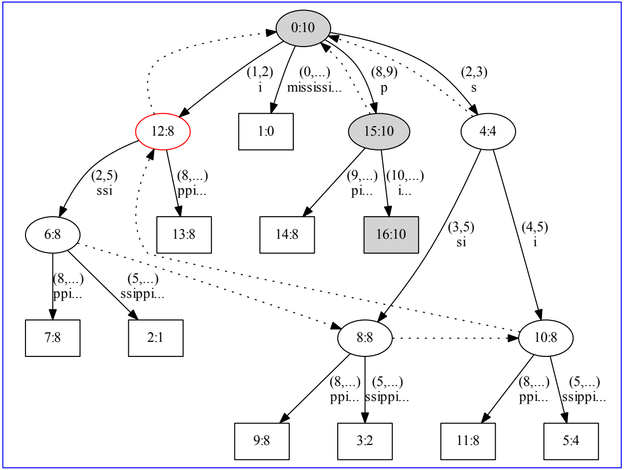
Experiment – mississippi



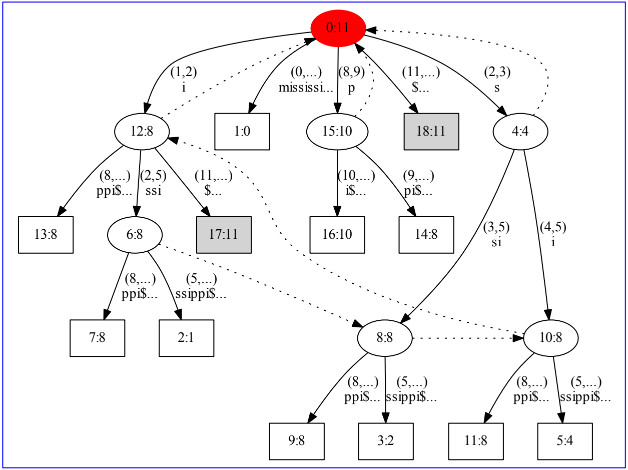
Experiment – mississippi



Experiment – mississippi



Experiment – mississippi



Experiment – English text