Suffix Trees

- String Dictionaries
- Tries
- Suffix Trees
- Suffix Sorting

Philip Bille

Suffix Trees

- String Dictionaries
- Tries
- Suffix Trees
- Suffix Sorting

String Dictionaries

- String dictionary problem. Let S be a string of characters from alphabet Σ. Preprocess S into data structure to support:
- search(P): Return the starting positions of all occurrences of P in S.

- Example.
  - S = yabbadabbado
  - search(abba) = {1,6}

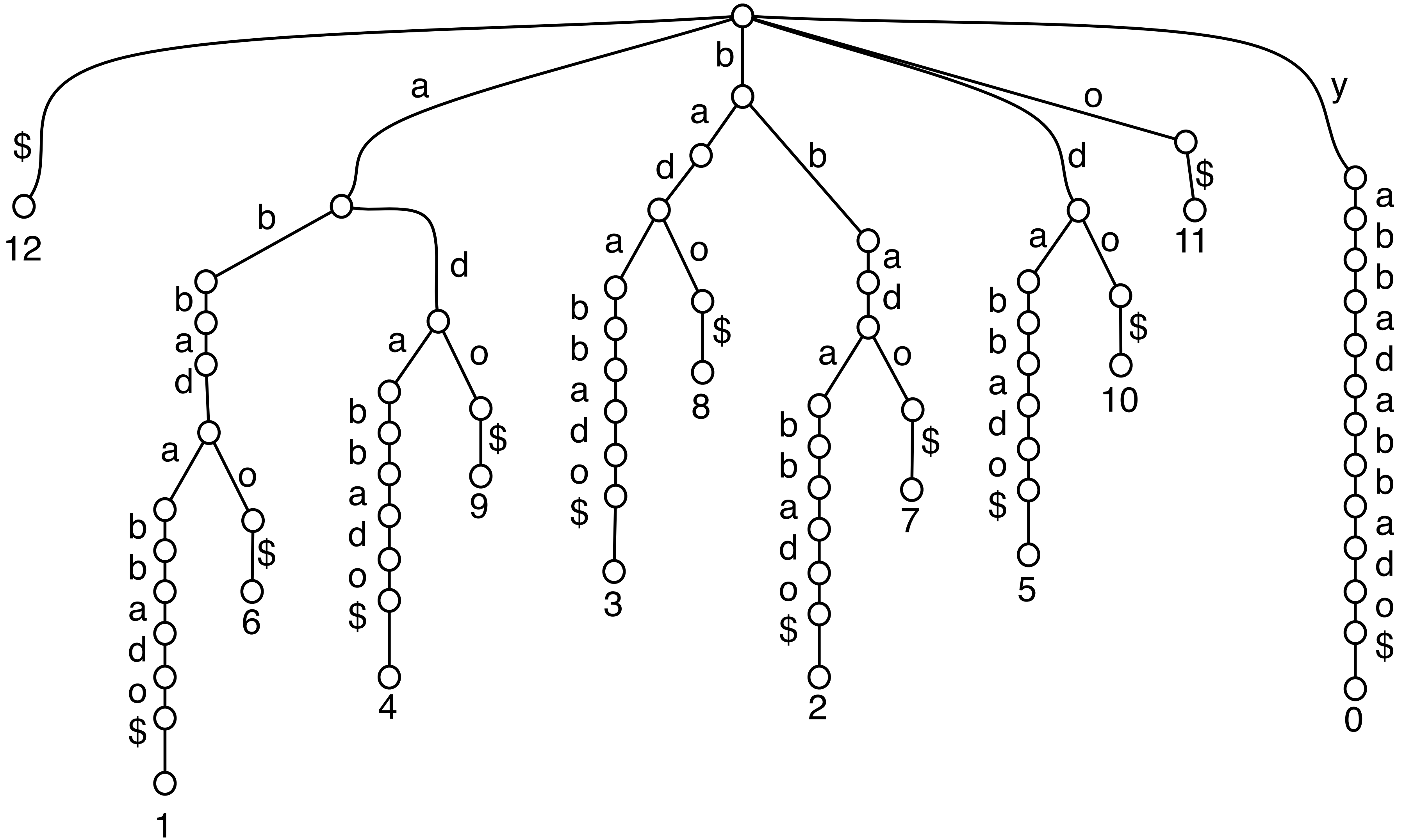
Suffix Trees

- String Dictionaries
- Tries
- Suffix Trees
- Suffix Sorting

Tries

- Tries [Fredkin 1960]. Retrieval. Store a set of strings in a rooted tree such that:
  - Each edge is labeled by a character. Edges to children of a node are sorted from left-to-right alphabetically.
  - Each root-to-leaf path represents a string in the set. (obtained by concatenating the labels of edges on the path).
  - Common prefixes share same path maximally.
- Prefix free.
  - Append special character $\$ <$ any character in Σ to each string.
  - $\Longrightarrow$ Each leaf correspond to a unique string.
- Suffix trie.
  - Trie of all suffixes of a string.

Tries

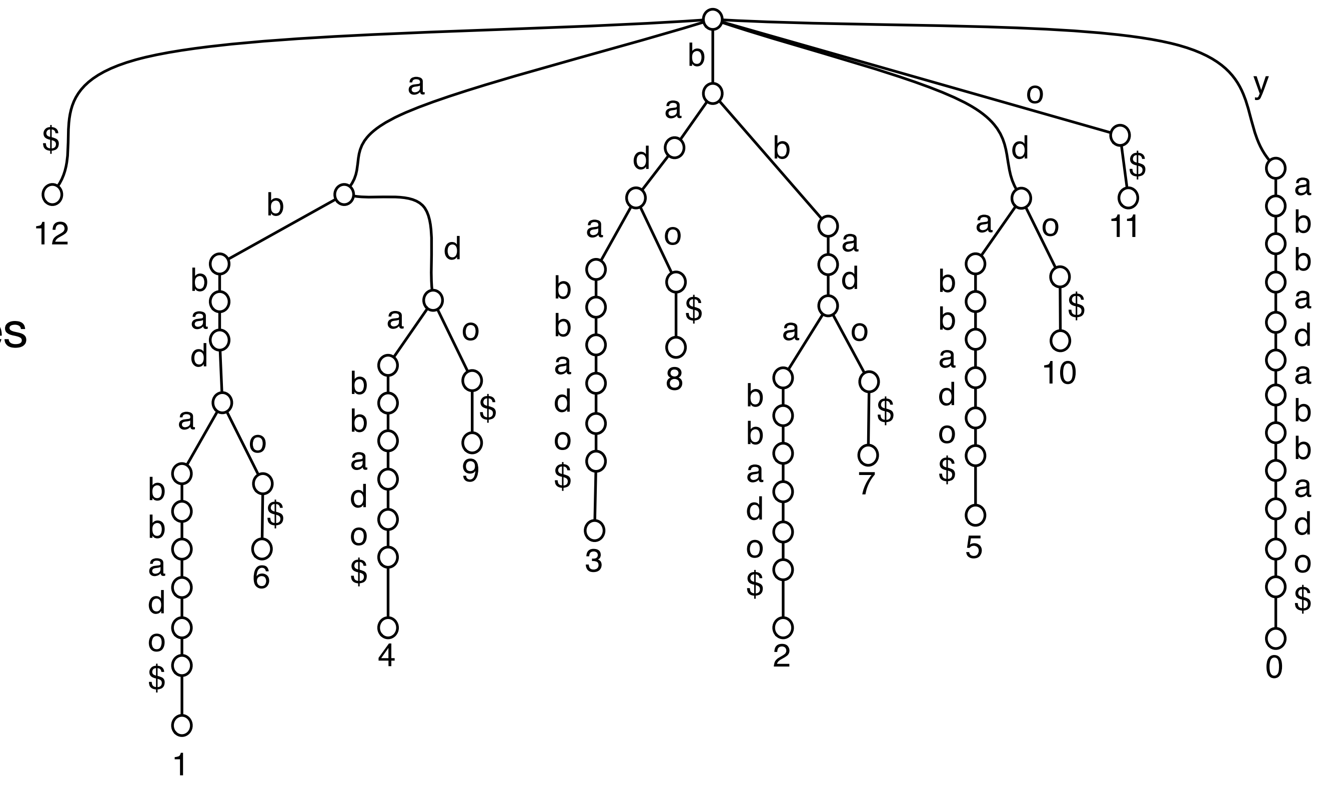


Trie of all suffixes for yabbadabbado$

- Space. $O(n^2)$
- Preprocessing. $O(n^2)$

Tries

- Search(P):
  - Process P from left-to-right while doing top-down search of trie:
    - At each node identify (unique) edge matching next character in P.
    - If no such edge, P is not a substring of S.
  - Report labels of all leaves below final node.
- Example.
  - search(abba) = {1,6}
- Time.
  - Top-down search +
  - Time for reporting leaves
  - $\Longrightarrow$ O(m + occ)



Tries

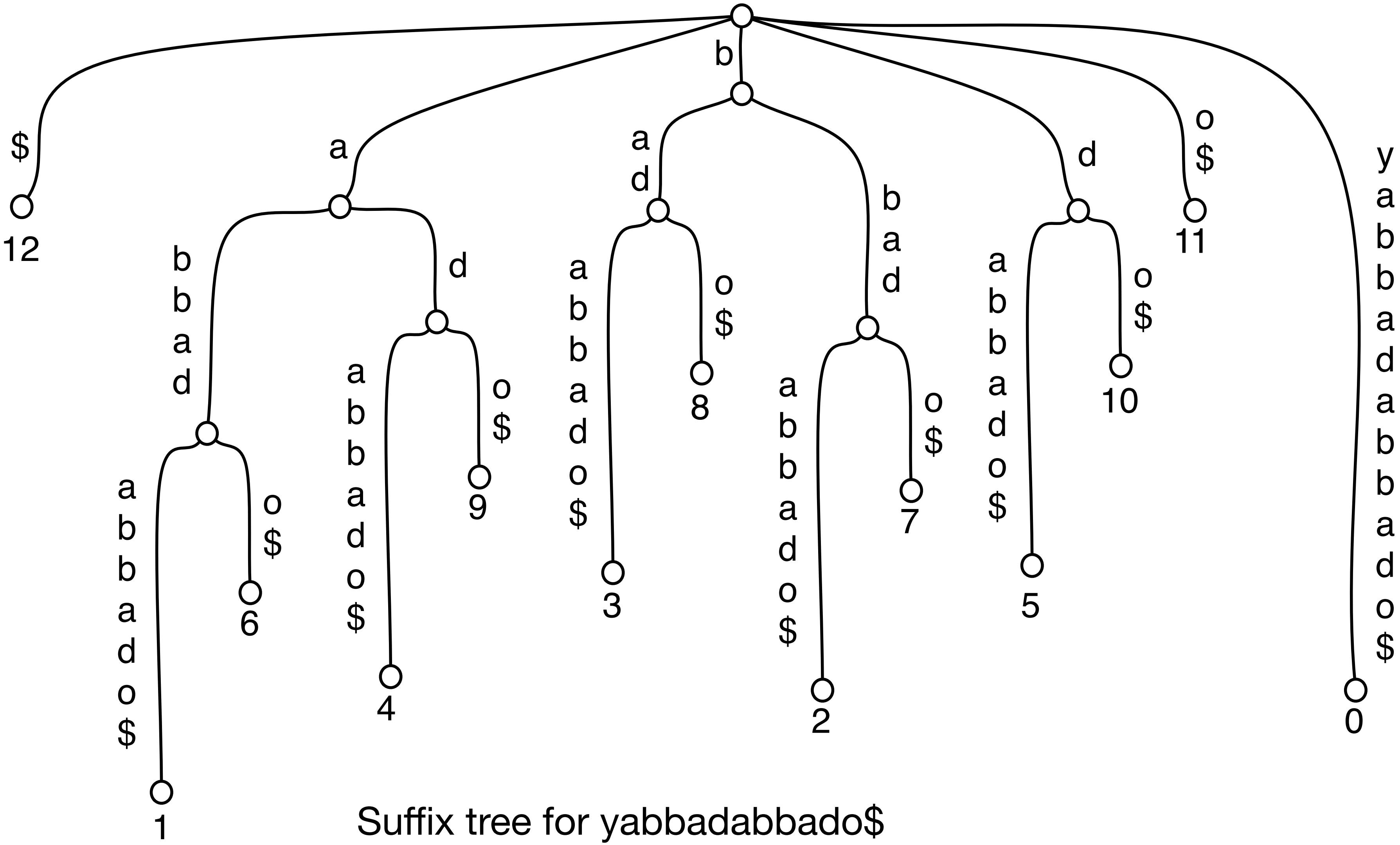
- Theorem. We can solve the string dictionary problem in
  - $O(n^2)$ space and preprocessing time.
  - $O(m + occ)$ time for queries.

Suffix Trees

- String Dictionaries
- Tries
- Suffix Trees
- Suffix Sorting

Suffix Trees

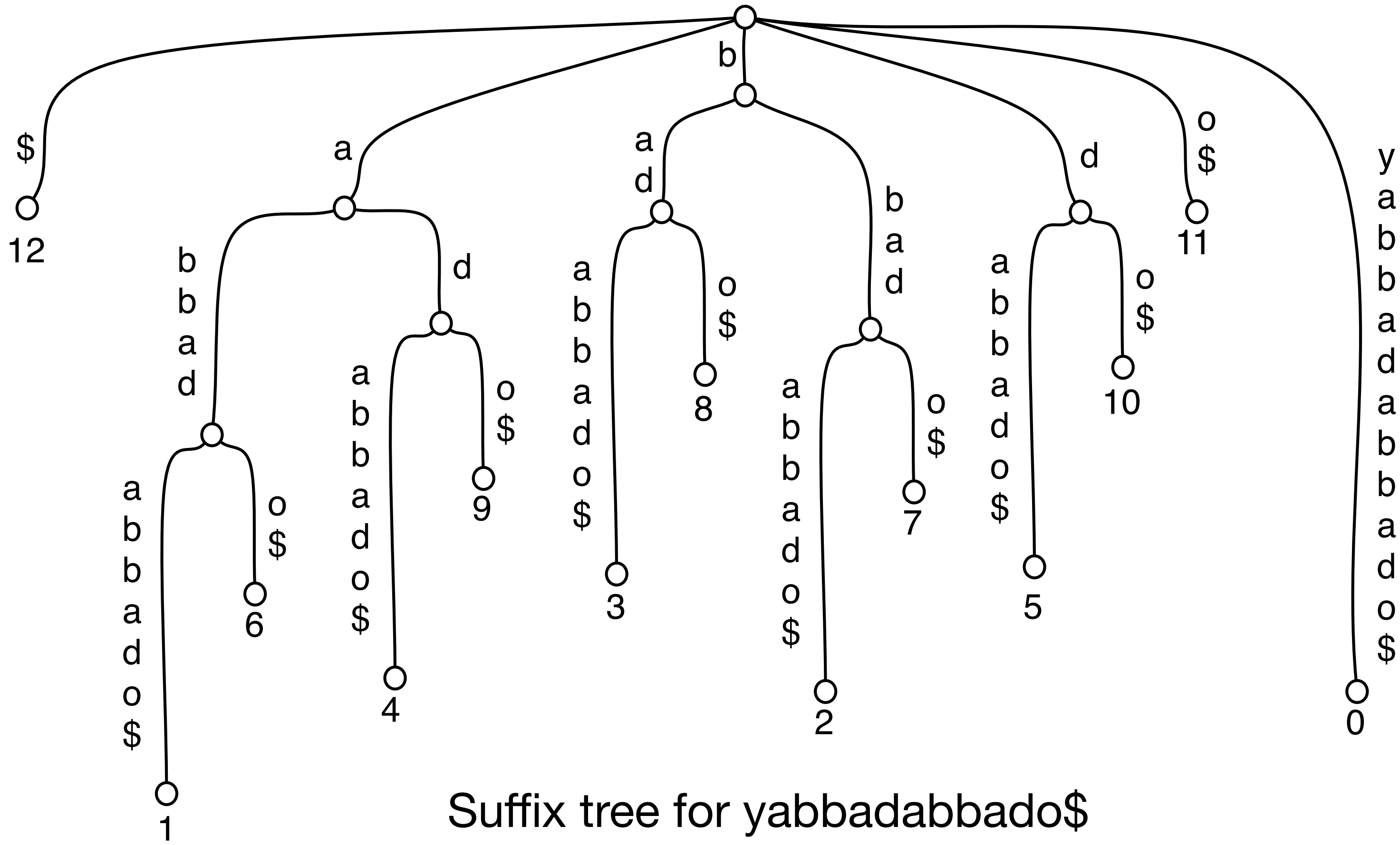
- Suffix trees. The compact trie of all suffixes of S.
  - Chains of nodes with single child are compacted into a single edge.



Suffix tree for yabbadabbado$

Suffix Trees

- Space.
  - Number of edges + space for edge labels
  - $\Longrightarrow$ O(n) space
- Preprocessing. O(sort(n,|Σ|))
- sort(n,|Σ|) = time to sort n characters from an alphabet Σ.
- Search(P): as before.



Suffix tree for yabbadabbado$

Suffix Trees

- Theorem. We can solve the string dictionary problem in
  - O(n) space and sort(n,|Σ|) preprocessing time.
  - O(m + occ) time for queries.

Suffix Trees

- Applications.
  - Approximate string matching problems
  - Compression schemes (Lempel-Ziv family, ...)
  - Repetitive string problems (palindromes, tandem repeats, ...)
  - Information retrieval problems (document retrieval, top-k retrieval, ...)
  - ...

Longest Common Extension

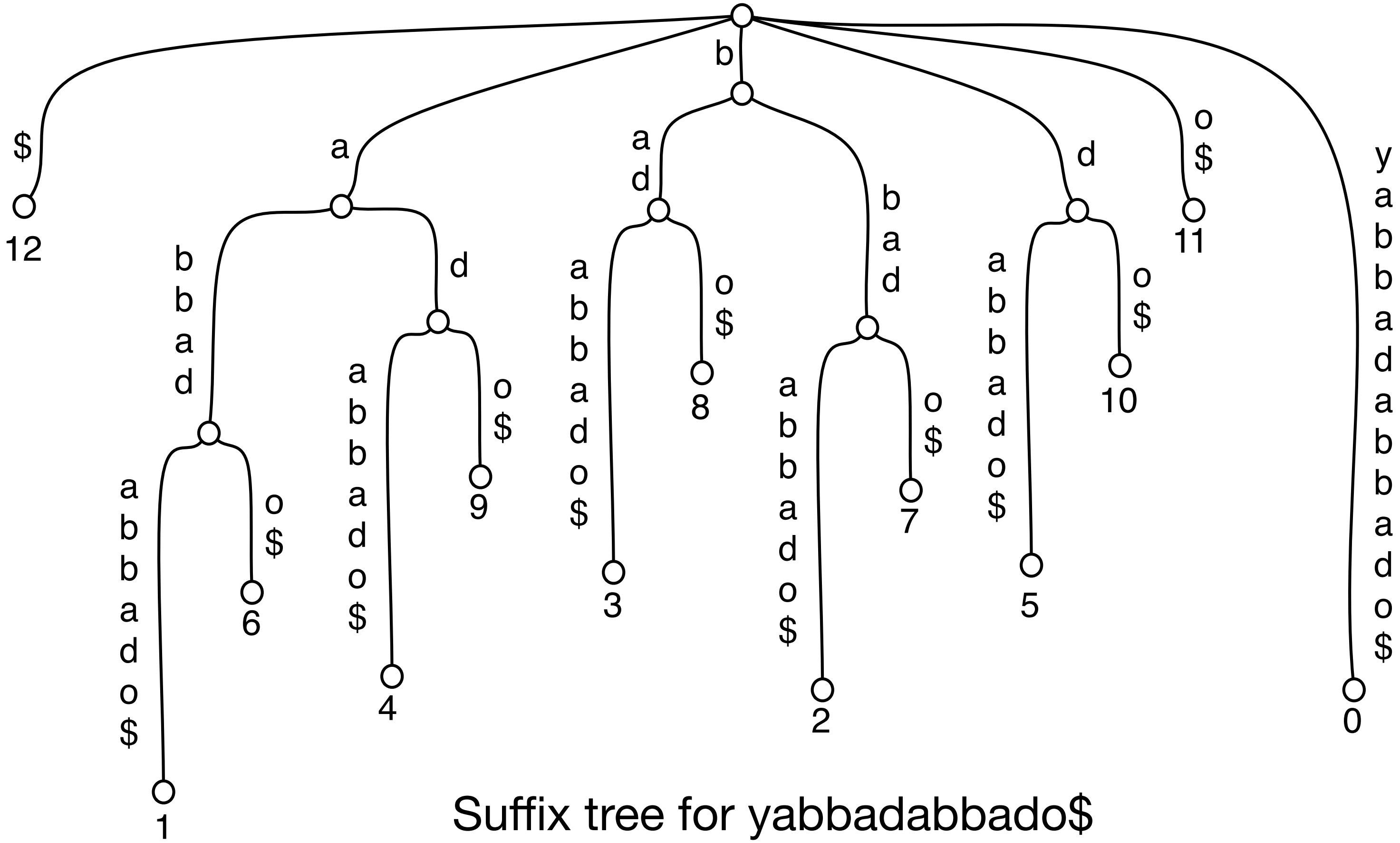
- Longest common extension problem. Let S be a string of characters from alphabet Σ. Preprocess S into data structure to support
  - LCP(i,j): Return the length of the longest common prefix of S[i,n] and S[j,n].

yabbadabbado

LCP(1,6) = 5

Longest Common Extension

- LCP and suffix trees?
- Solution. Suffix tree + string depth of each node + nearest common ancestor data structure.
- ⟹ O(n) space and O(1) query time.



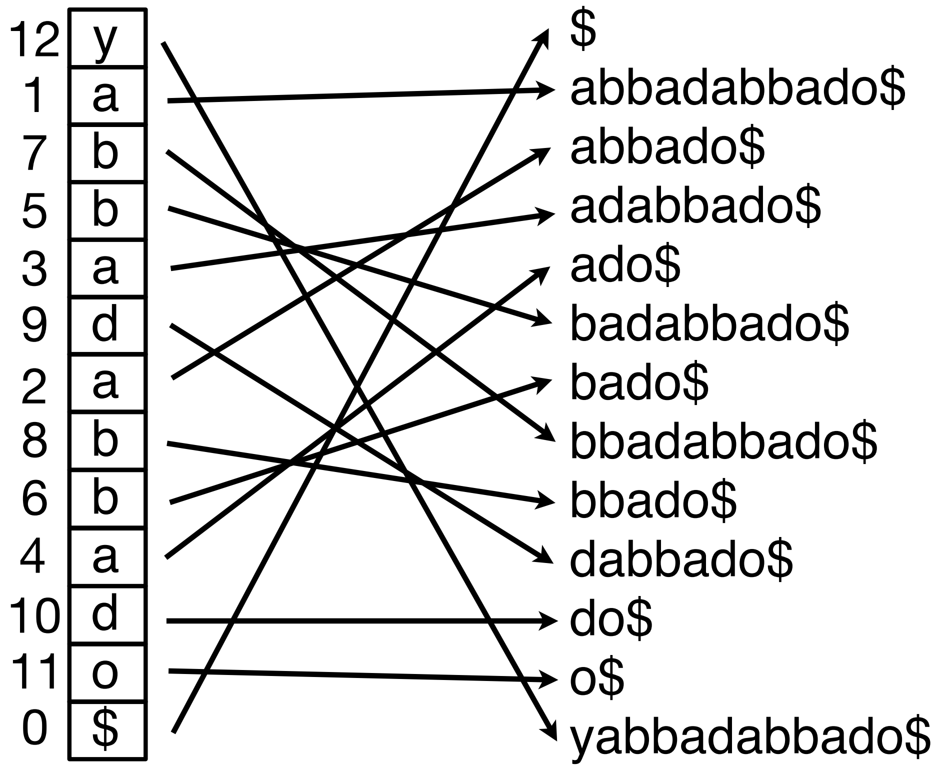
Suffix tree for yabbadabbado$
LCP(1,6) = 5

Suffix Trees

- String Dictionaries
- Tries
- Suffix Trees
- Suffix Sorting

Suffix Sorting

- Suffix sorting. Given string S of length n over alphabet Σ, compute the sorted lexicographic order of all suffixes of S.
- Theorem [Kasai et al. 2001]. Given the sorted lexicographic order of suffixes of S, we can construct the suffix tree for S in linear time.
- How do we sort suffixes?



Suffix Sorting

- Goal. Compute the lexicographic order of all suffixes of S fast.
- Warm up. Sorting small universes.
- Solution in 3 steps.
  - Solution 1: Radix sorting
  - Solution 2: Prefix doubling
  - Solution 3: Difference cover sampling

Sorting Small Universes

- Let X be a sequence of n integers from a universe U = {0, 1, ..., u-1}.
- How fast can we sort if the size of the universe is not too big?
  - U = {0,1}?
  - U = {0, ... , n-1}?
  - U = {0, ..., n^3 - 1}?

Sorting Small Universes

- Positional number systems. The base-n representation of x is x written in base n.
- Example.
  - $(10)_{10} = (1010)_2$ $(1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0)$
  - $(107)_{10} = (212)_7$ $(2 \cdot 7^2 + 1 \cdot 7^1 + 2 \cdot 7^0)$

- Radix Sort [Hollerith 1887]. Sort sequence X of n integers from $U = \{0, ..., n^3-1\}$.
  - Write each $x \in X$ as a base n integer (x_1, x_2, x_3): $x = x_1 \cdot n^2 + x_2 \cdot n + x_3$
  - Sort X according to rightmost (least significant) digit
  - Sort X according to middle digit
  - Sort X according to leftmost (most significant) digit
- Each sort should be stable.
- Final result is the sorted sequence of X.

Sorting Small Universes

$n = 10, U = \{0, ..., n^3 - 1 = 999\}$

170	170	200	002
045	090	802	024
075	200	002	045
090	802	905	066
802	002	024	075
024	024	045	090
002	045	066	170
066	075	170	200
200	905	075	802
905	066	090	905

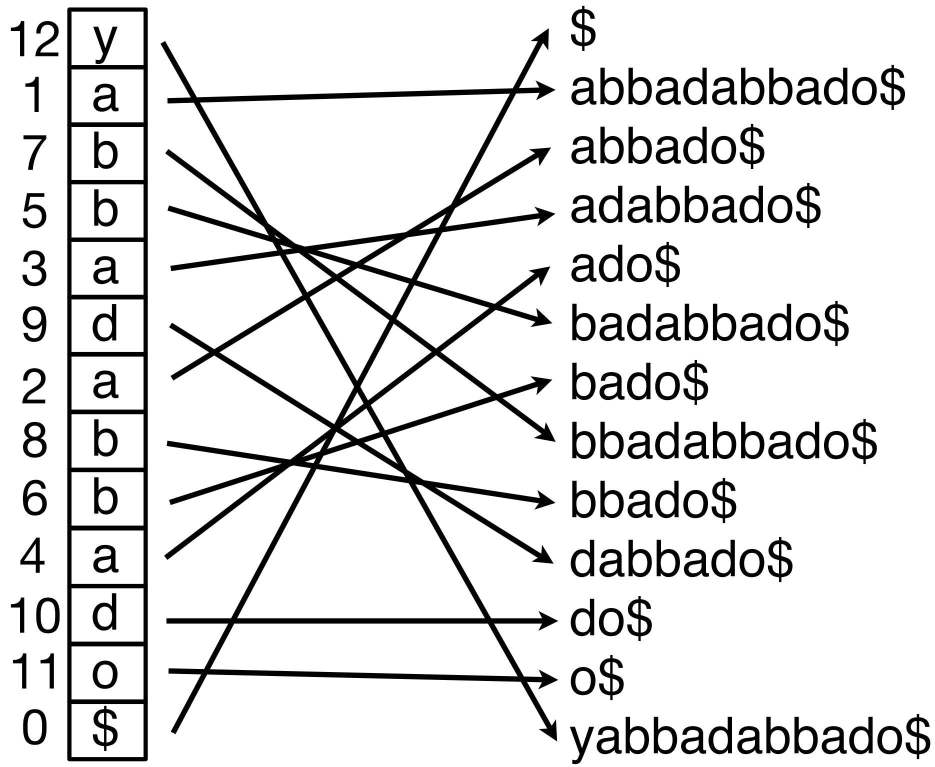
Sorting Small Universes

- Theorem. We can sort n integers from a universe $U = \{0, ..., n^3 - 1\}$ in $O(n)$ time.
- Theorem. We can sort n integers from a universe $U = \{0, ..., n^k - 1\}$ in $O(kn)$ time.

- Larger universes?
- Theorem [Han and Thorup 2002]. We can sort n integers in $O(n \log \log n)$ time or $O(n (\log \log n)^{1/2})$ expected time.

Suffix Sorting

- Suffix sorting. Given string S of length n over alphabet Σ, compute the sorted lexicographic order of all suffixes of S.
- For simplicity assume $|\Sigma| = O(n)$



Solution 1: Radix Sort

- Radix Sort.
  - Generate all suffixes (pad with $).
  - Radix sort.

```
yabbadabbado$
abbadabbado$$
bbadabbado$$$
badabbado$$$$
adabbado$$$$$
dabbado$$$$$$
abbado$$$$$$$
bbado$$$$$$$$
bado$$$$$$$$$
ado$$$$$$$$$$
do$$$$$$$$$$$
o$$$$$$$$$$$$
$$$$$$$$$$$$$
```

- Theorem. We can suffix sort a string of length n in time $O(n^2)$.

Solution 2: Prefix Doubling

- Prefix doubling [Manber and Myers 1990]. Sort substrings (padded with $) of lengths 1, 2, 4, 8, ..., n. Each step uses radix sort on pair from previous step.

5	y	8	51	ya	10	84	yabb
1	a	1	12	ab	1	13	abba
2	b	4	22	bb	6	42	bbad
2	b	3	21	ba	4	35	bada
1	a	2	13	ad	2	21	adab
3	d	5	31	da	7	54	dabb
1	a	1	12	ab	1	13	abba
2	b	4	22	bb	6	42	bbad
2	b	3	21	ba	5	36	bado
1	a	2	13	ad	3	27	ado$
3	d	6	34	do	8	60	do$$
4	o	7	40	o$	9	70	o$$$
0	$	0	00	$$	0	00	$$$$

. . . .

- Theorem. We can suffix sort a string of length n in time O(n log n).

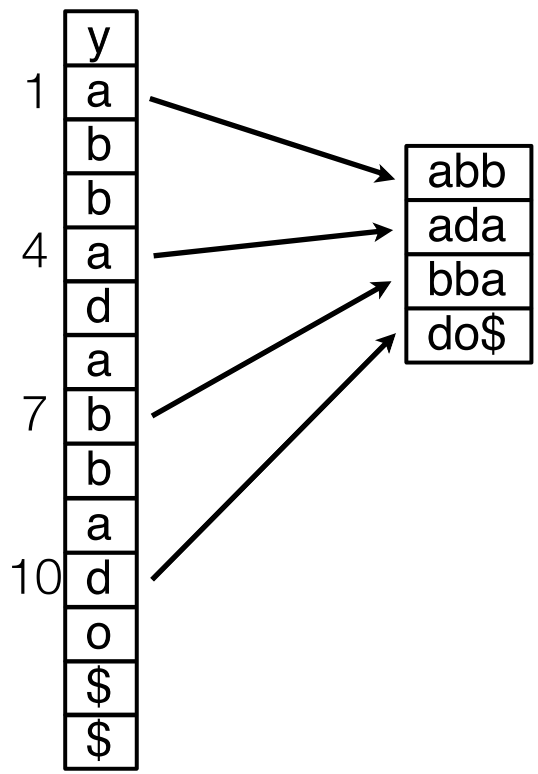
Solution 3: Difference Cover Sampling

- DC3 Algorithm [Karkkainen et al. 2003]. Sort suffixes in three steps:
  - Step 1. Sort sample suffixes.
    - Sample all suffixes starting at positions i = 1 mod 3 and i = 2 mod 3.
    - Recursively sort sample suffixes.
  - Step 2. Sort non-sample suffixes.
    - Sort the remaining suffixes (starting at positions i = 0 mod 3).
  - Step 3. Merge.
    - Merge sample and non-sample suffixes.

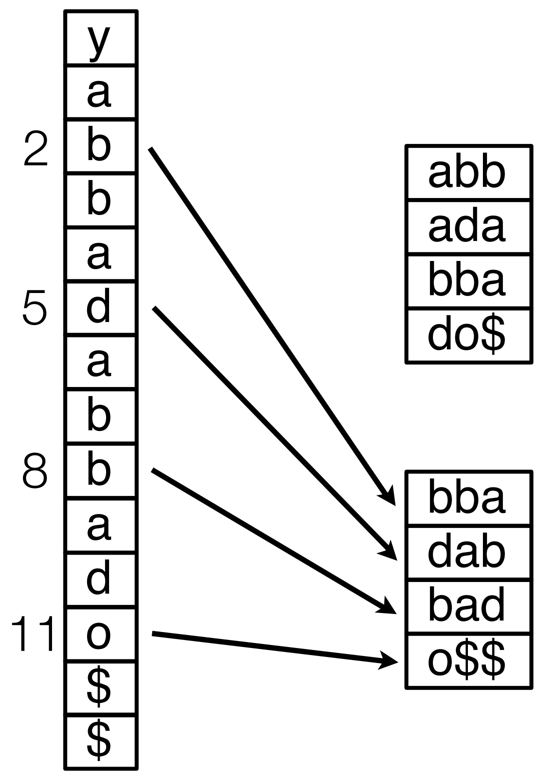
Step 1: Sort Sample Suffixes

y
a
b
b
a
d
a
b
b
a
d
o
$
$

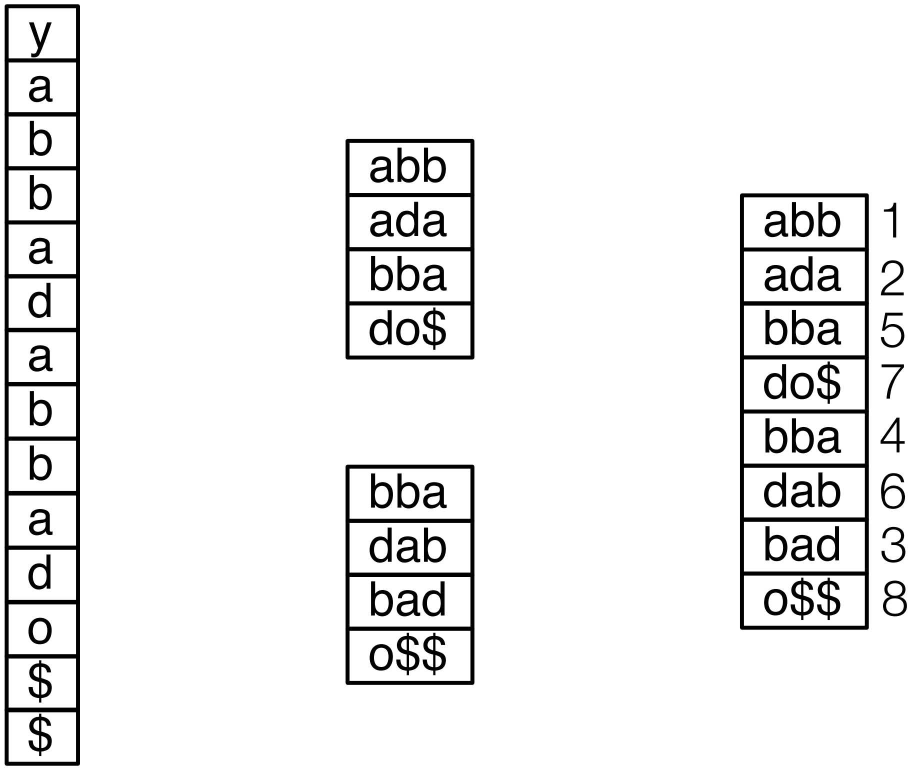
Step 1: Sort Sample Suffixes



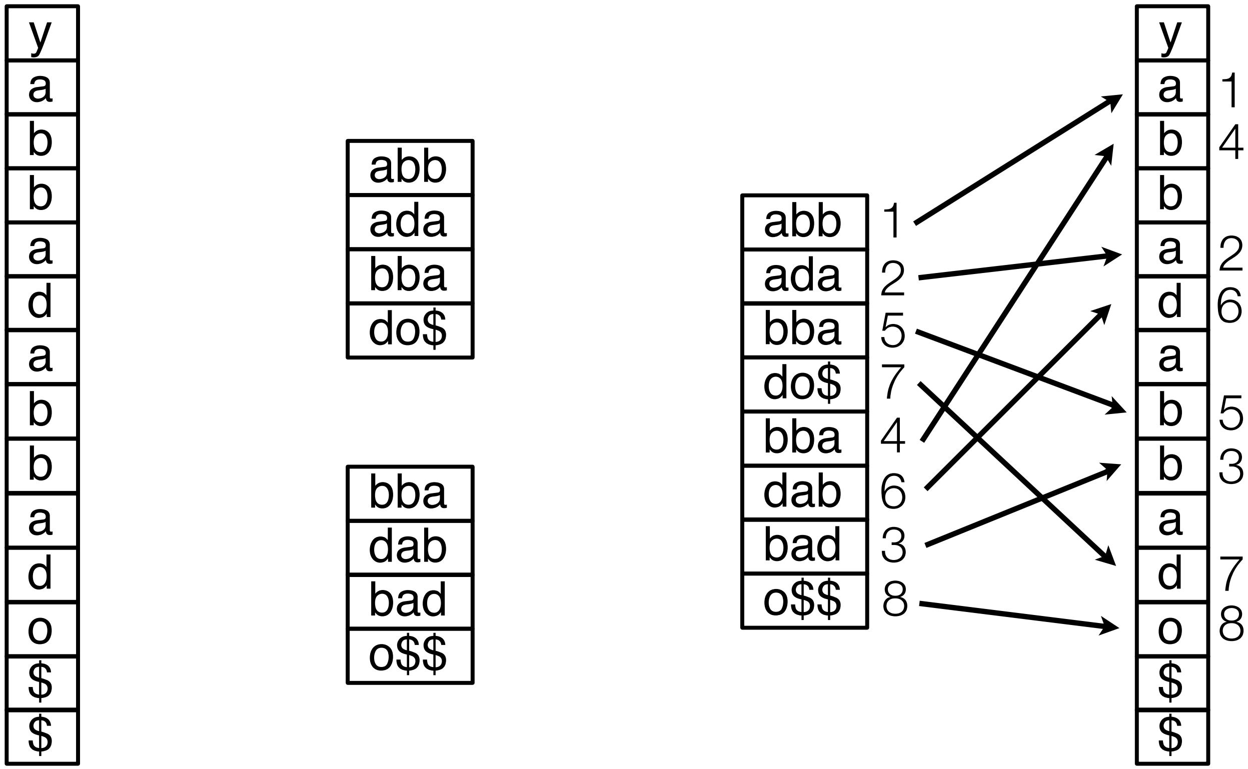
Step 1: Sort Sample Suffixes



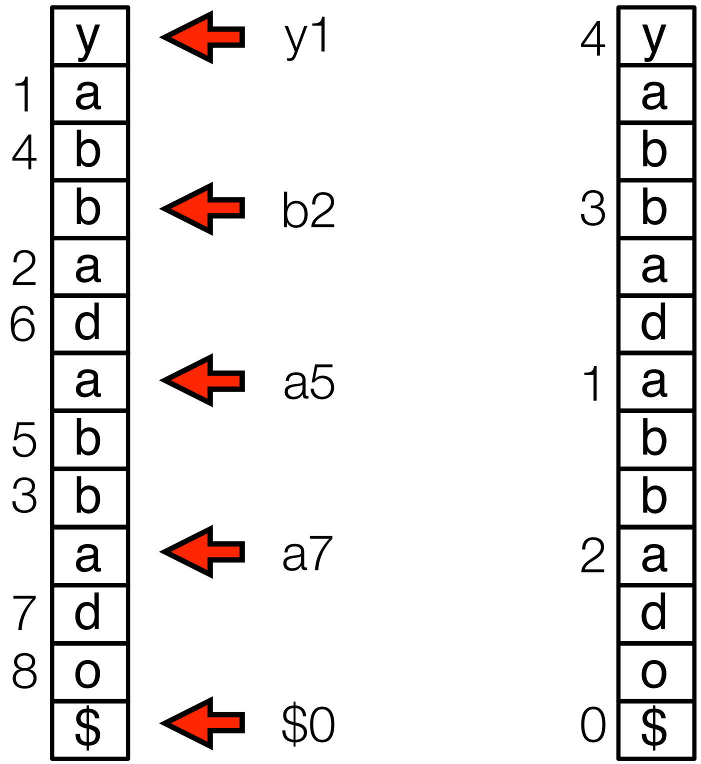
Step 1: Sort Sample Suffixes



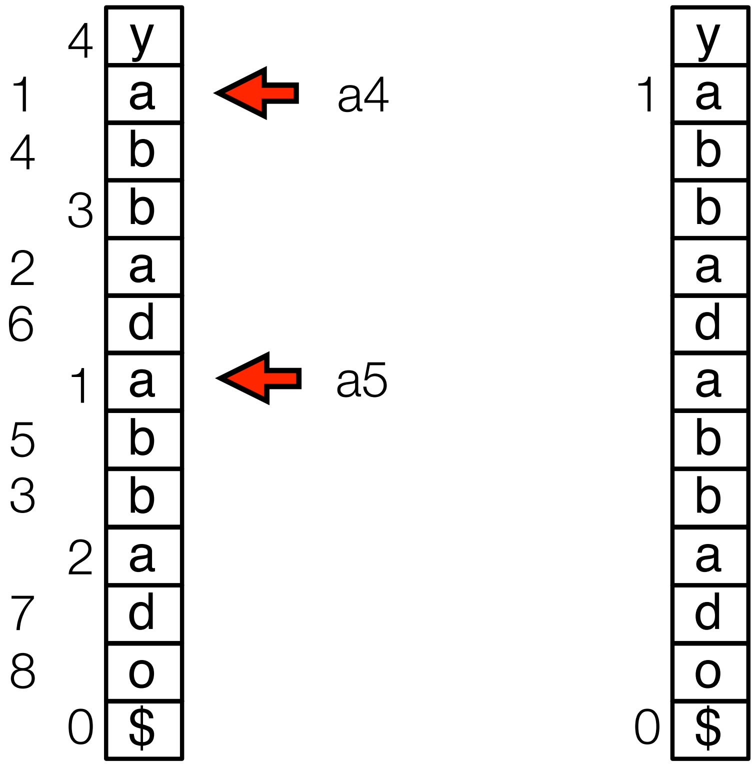
Step 1: Sort Sample Suffixes



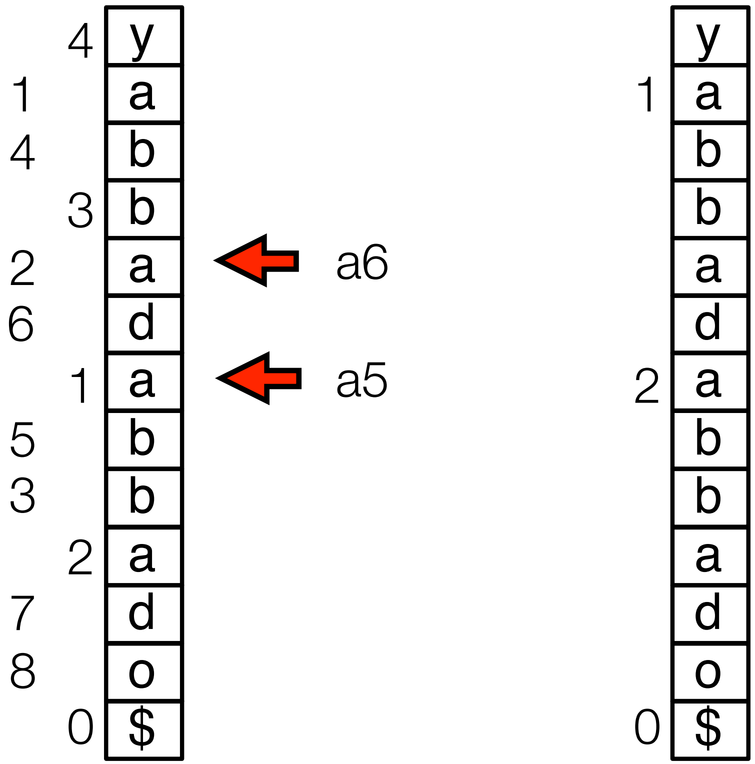
Step 2: Sort Non-Sample Suffixes



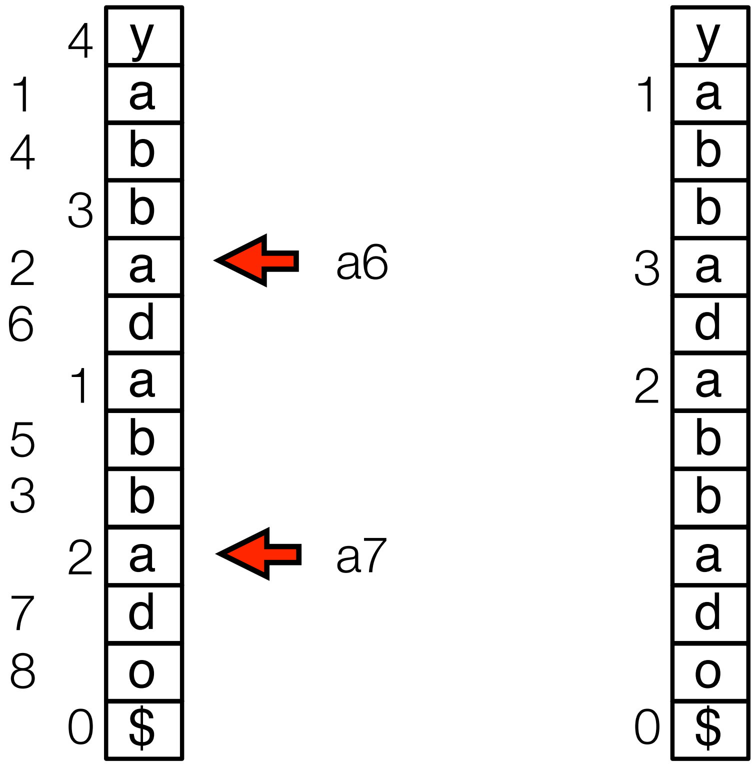
Step 3: Merge



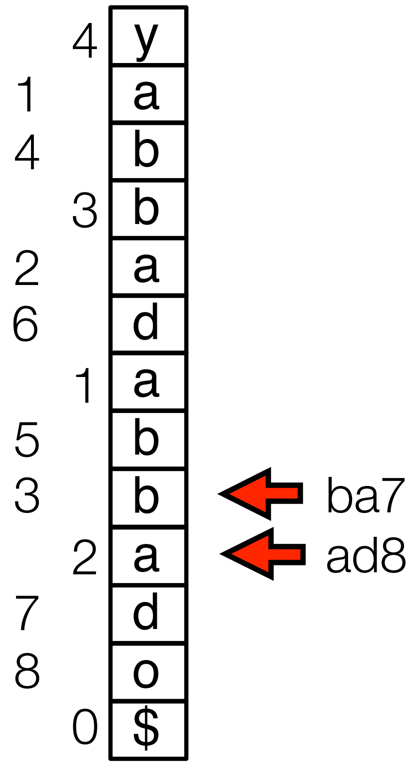
Step 3: Merge



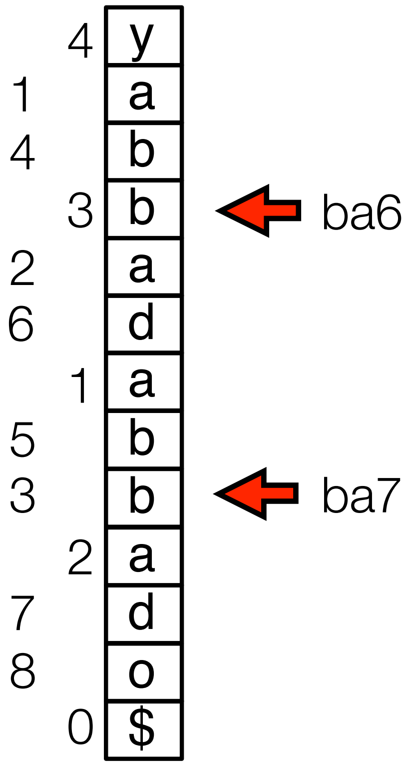
Step 3: Merge



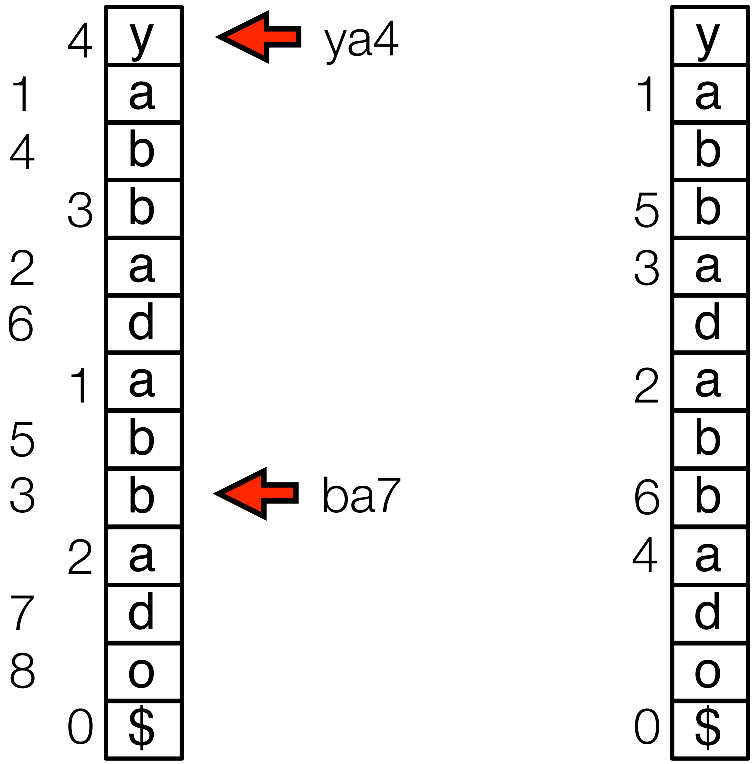
Step 3: Merge



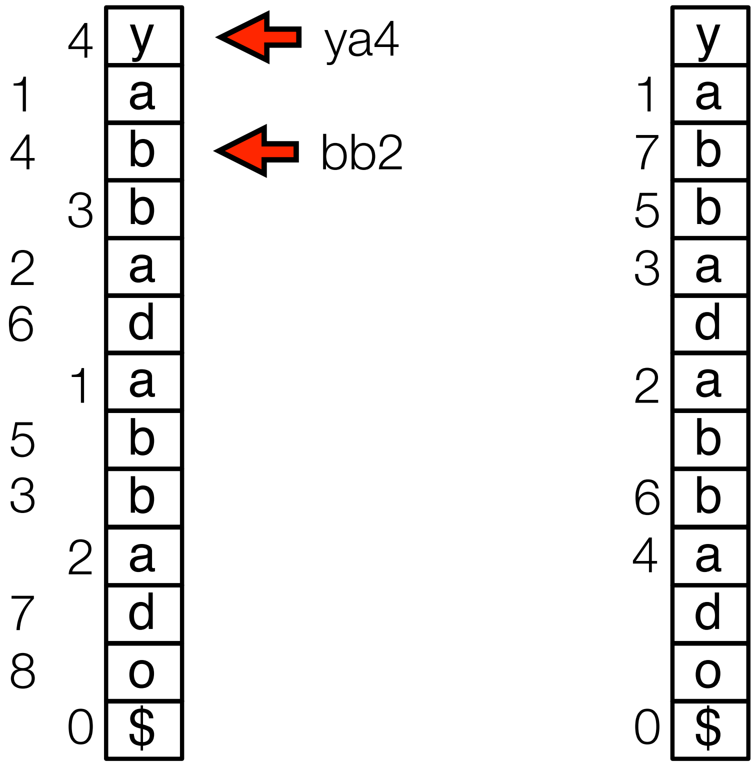
Step 3: Merge



Step 3: Merge

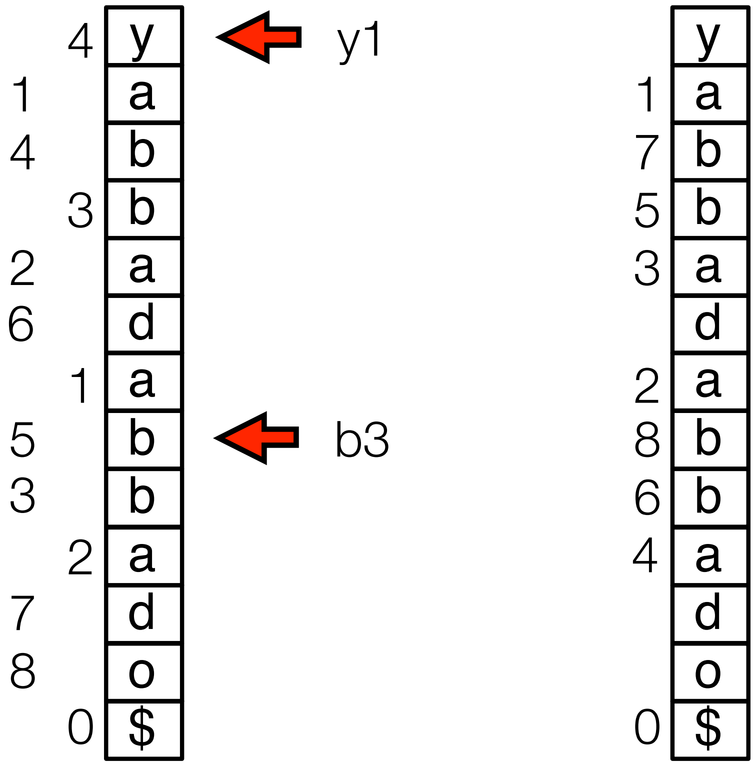


Step 3: Merge

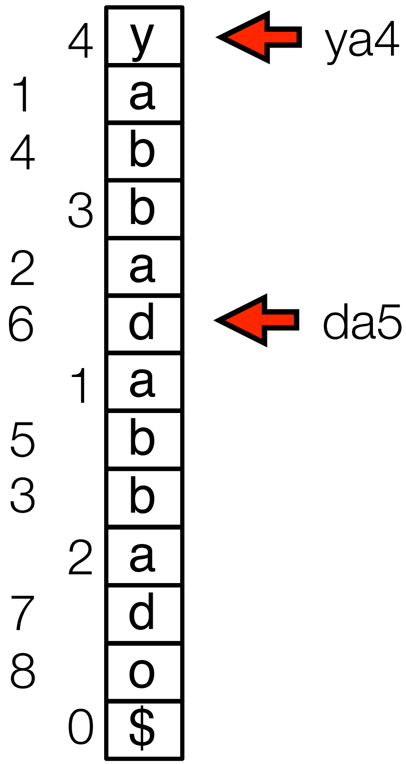


4 y
1 a
4 b
3 b
2 a
6 d
1 a
5 b
3 b
2 a
7 d
8 o
0 $
ya4
bb2
y
1 a
7 b
5 b
3 a
d
2 a
b
6 b
4 a
d
o
0 $

Step 3: Merge

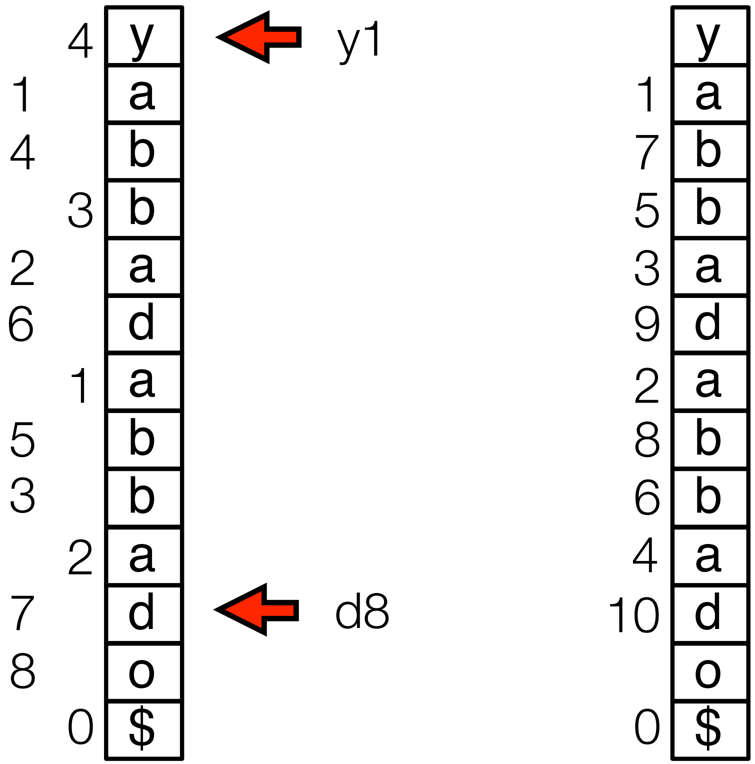


Step 3: Merge



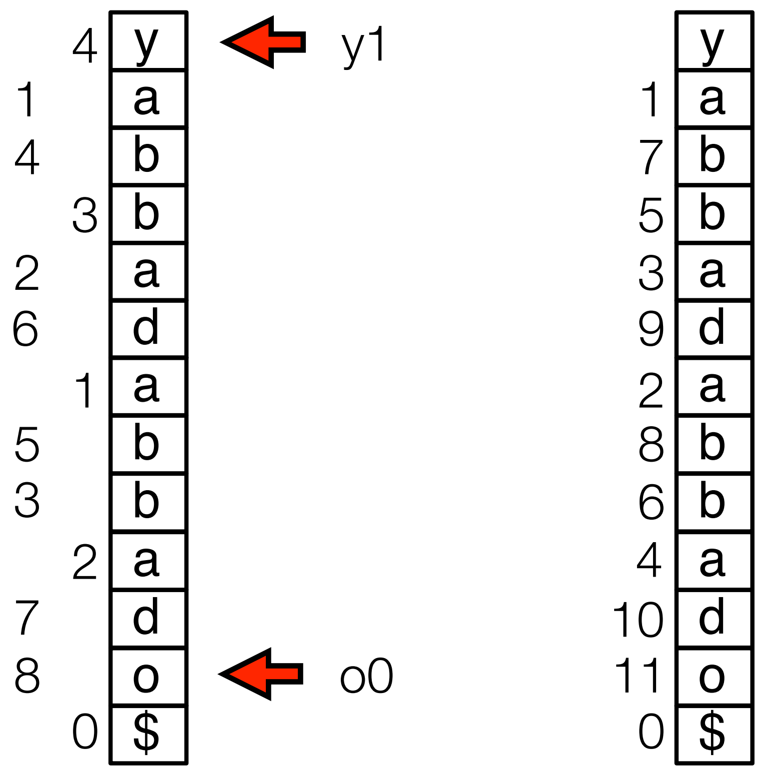
4 y
1 a
4 b
3 b
2 a
6 d
1 a
5 b
3 b
2 a
7 d
8 o
0 $
ya4
da5
y
1 a
7 b
5 b
3 a
9 d
2 a
8 b
6 b
4 a
d
o
0 $

Step 3: Merge



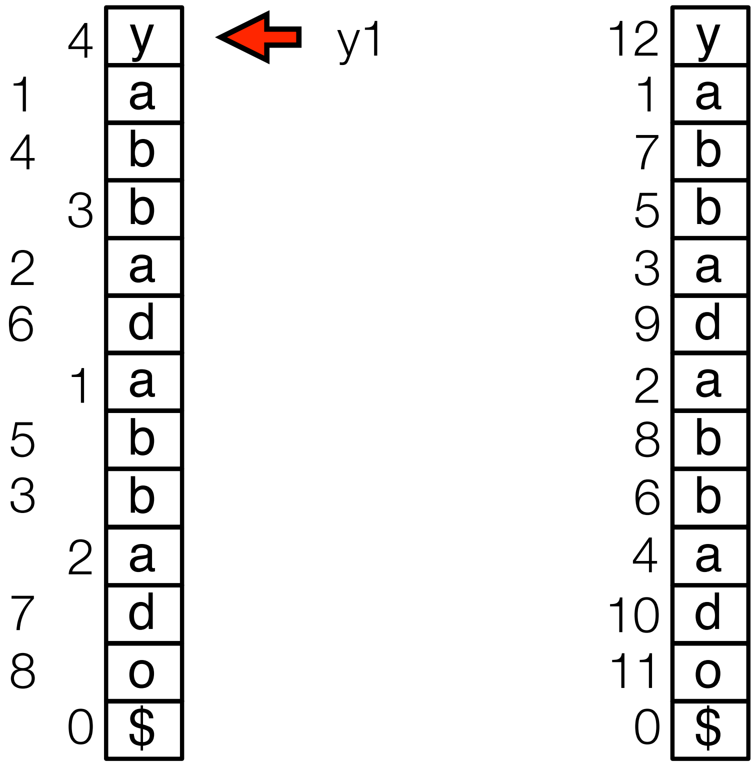
4 y
1 a
4 b
3 b
2 a
6 d
1 a
5 b
3 b
2 a
7 d
8 o
0 $
y1
d8
y
1 a
7 b
5 b
3 a
9 d
2 a
8 b
6 b
4 a
10 d
o
0 $

Step 3: Merge



4 y
1 a
4 b
3 b
2 a
6 d
1 a
5 b
3 b
2 a
7 d
8 o
0 $
y1
o0
y
1 a
7 b
5 b
3 a
9 d
2 a
8 b
6 b
4 a
10 d
11 o
0 $

Step 3: Merge



4 y
1 a
4 b
3 b
2 a
6 d
1 a
5 b
3 b
2 a
7 d
8 o
0 $
y1
12 y
1 a
7 b
5 b
3 a
9 d
2 a
8 b
6 b
4 a
10 d
11 o
0 $

Solution 3: Difference Cover Sampling

- DC3 Algorithm. Sort suffixes in three steps:
  - Step 1. Sort sample suffixes.
    - Sample all suffixes starting at positions i = 1 mod 3 and i = 2 mod 3. O(n)
    - Recursively sort sample suffixes. T(2n/3)
  - Step 2. Sort non-sample suffixes.
    - Sort the remaining suffixes (starting at positions i = 0 mod 3). O(n)
  - Step 3. Merge.
    - Merge sample and non-sample suffixes. O(n)
- T(n) = time to suffix sort a string of length n over alphabet of size n

- Time. T(n) = T(2n/3) + O(n) = O(n)

Solution 3: Difference Cover Sampling

- Theorem. We can suffix sort a string of length n over alphabet Σ of size n in time O(n).
- Larger alphabets?
- Theorem. We can suffix sort a string of length n over alphabet Σ O(sort(n, |Σ|)) time.
- Bound is optimal.

Suffix Trees

- String Dictionaries
- Tries
- Suffix Trees
- Suffix Sorting