Longest common substring problem

Problem definition

- Str S(len m) and str T(len n)
- Generalized: set of strings $S = \{S_1, \ldots, S_K\}$, where $|S_i| = n_i$, $\Sigma n_i = N$. Find for each $2 \leq k \leq K$ the longest strings which occur as substrings of at least k strings.
- DP: Time Complexity: O(m*n), Auxiliary Space: O(m*n)
- Suffix tree: O(m+n)

Dynamic Programming approach

- The idea is to find length of the longest common suffix for all substrings of both strings and store these lengths in a table.

- The longest common suffix has following optimal substructure property.

  If last characters match, then we reduce both lengths by 1

  LCSuff(X, Y, m, n) = LCSuff(X, Y, m-1, n-1) + 1 if X[m-1] = Y[n-1]

  If last characters do not match, then result is 0, i.e.,

  LCSuff(X, Y, m, n) = 0 if (X[m-1] != Y[n-1])

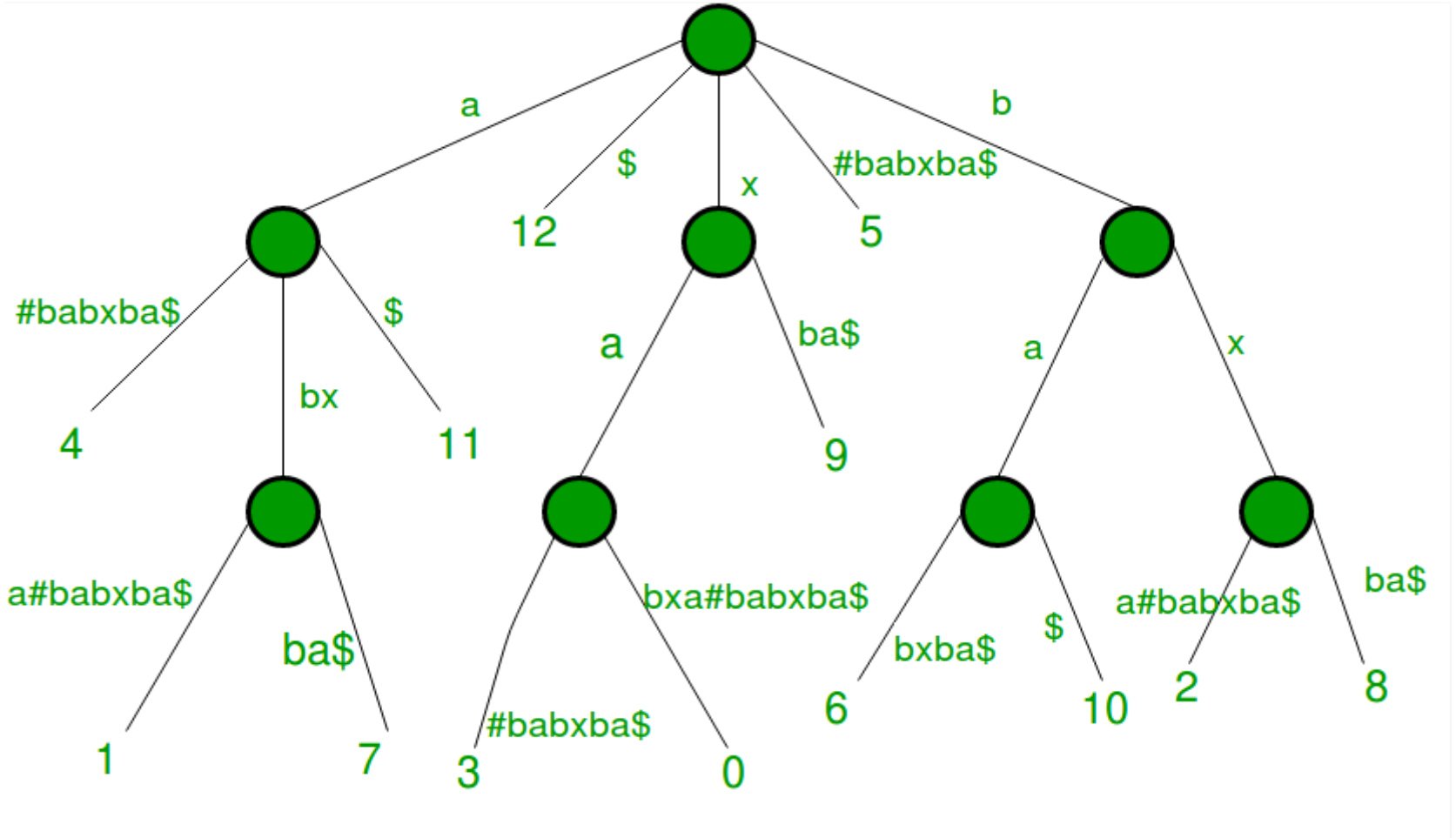
  Now we consider suffixes of different substrings ending at different indexes.

  The maximum length Longest Common Suffix is the longest common substring.

  LCSubStr(X, Y, m, n) = Max(LCSuff(X, Y, i, j)) where 1 <= i <= m and 1 <= j <= n

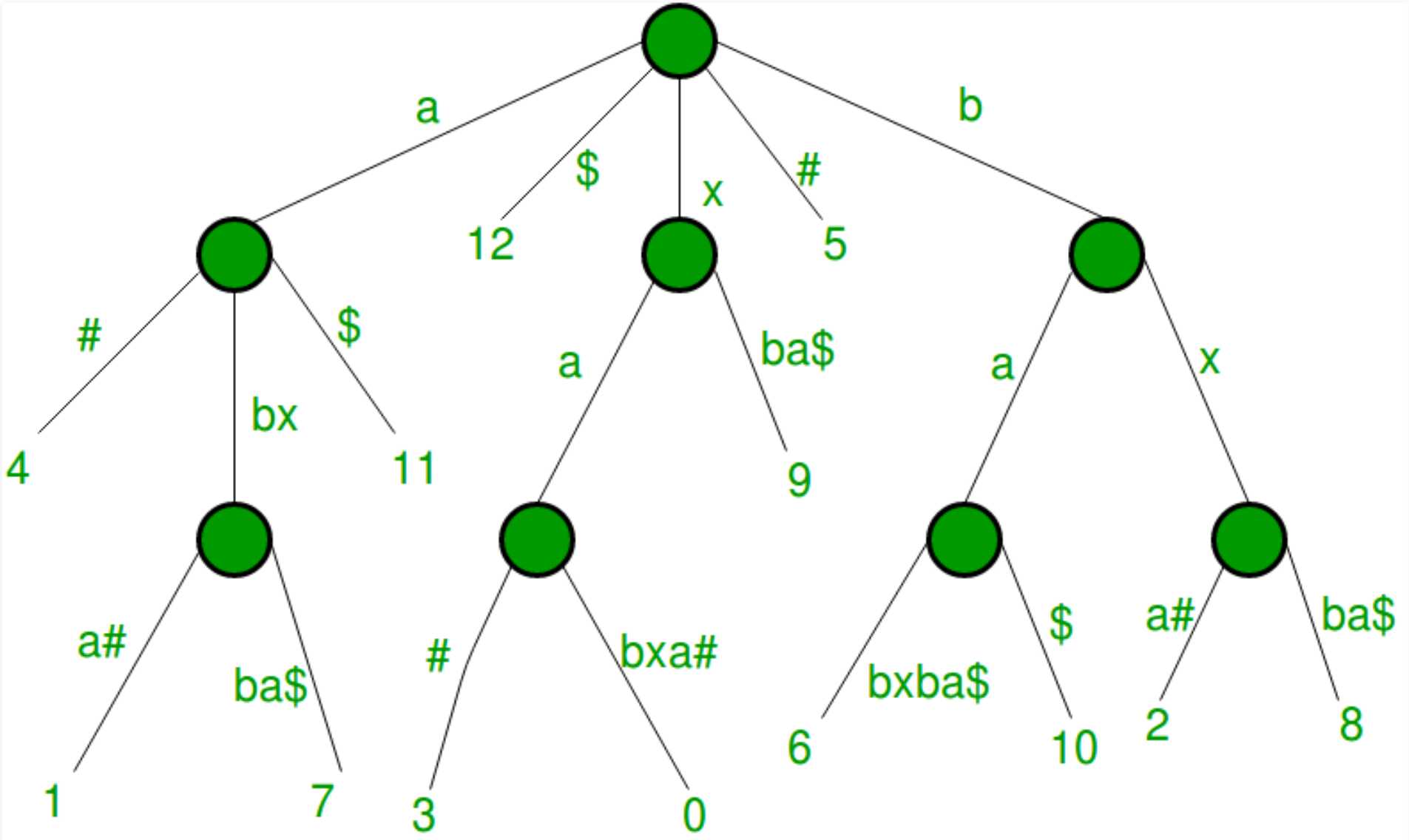
Suffix tree

- Lets say X = xabxa, and Y = babxba, X#Y$ = **xabxa#babxba$**



Refining with rm [babxba$]

- We can use this tree to solve some of the problems, but we can refine it a bit by removing unwanted substrings on a path label. A path label should have substring from only one input string, so if there are path labels having substrings from multiple input strings, we can keep only the initial portion corresponding to one string and remove all the later portion. For example, for path labels #babxba$, a#babxba$ and bxa#babxba$, we can remove babxba$ and then new path labels will be #, a# and bxa# respectively.


a
$
x
#
b
12
5
#
$
a
ba$
a
x
4
bx
11
9
a#
ba$
#
bxa#
6
bxba$
$
a#
ba$
1
7
3
0
10
2
8

Analysis

- If two strings are of size M and N, this implementation will take O(M+N) time and space.
- If input strings are not concatenated already, then it will take 2(M+N) space in total, M+N space to store the generalized suffix tree and another M+N space to store concatenated string.

- Followup: Extend above implementation for more than two strings (concatenate all strings using unique terminal symbols and then build suffix tree for concatenated string)