Data Structures and its Applications

V R BADRI PRASAD
Department of Computer Science & Engineering

DATA STRUCTURES AND ITS APPLICATIONS

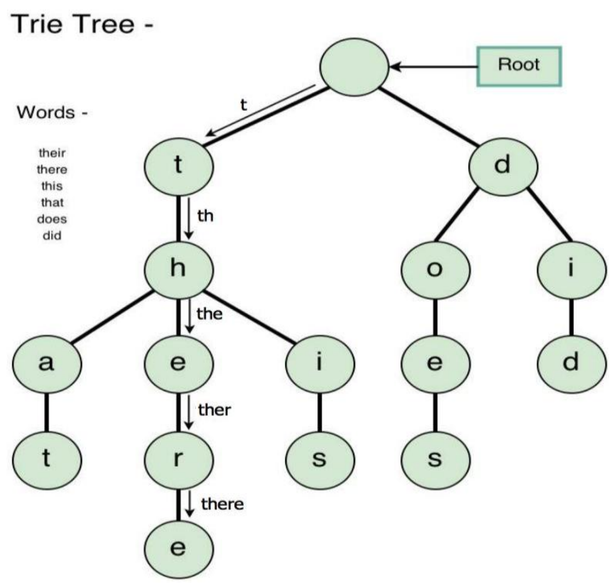
Introduction to TRIE Trees

V R BADRI PRASAD
Department of Computer Science & Engineering

TRIE Trees – An Introduction

- TRIE tree is a digital search tree, need not be implemented as a binary tree.
- Each node in the tree can contain 'm' pointers – corresponding to 'm' possible symbols in each position of the key.
- Generally used to store strings.

Examples:



TRIE Trees – An Introduction

- A trie, pronounced “try”, is a tree that exploits some structure in the keys

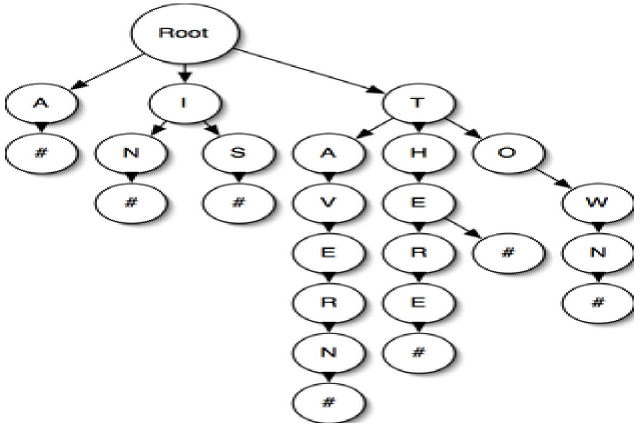
  - e.g. if the keys are strings, a binary search tree would compare the entire strings

    but a trie would look at their individual characters

  -A trie is a tree where each node stores a bit indicating whether the string

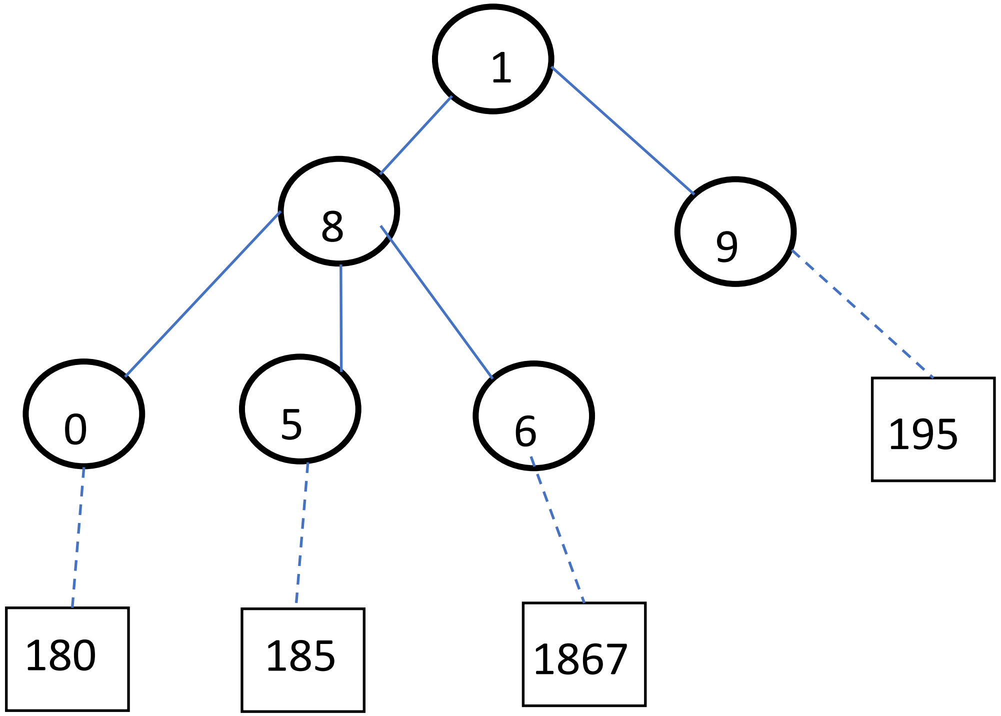
    spelled out to this point is in the set

  -**Examples:**



TRIE Trees – Numeric Keys : Example2

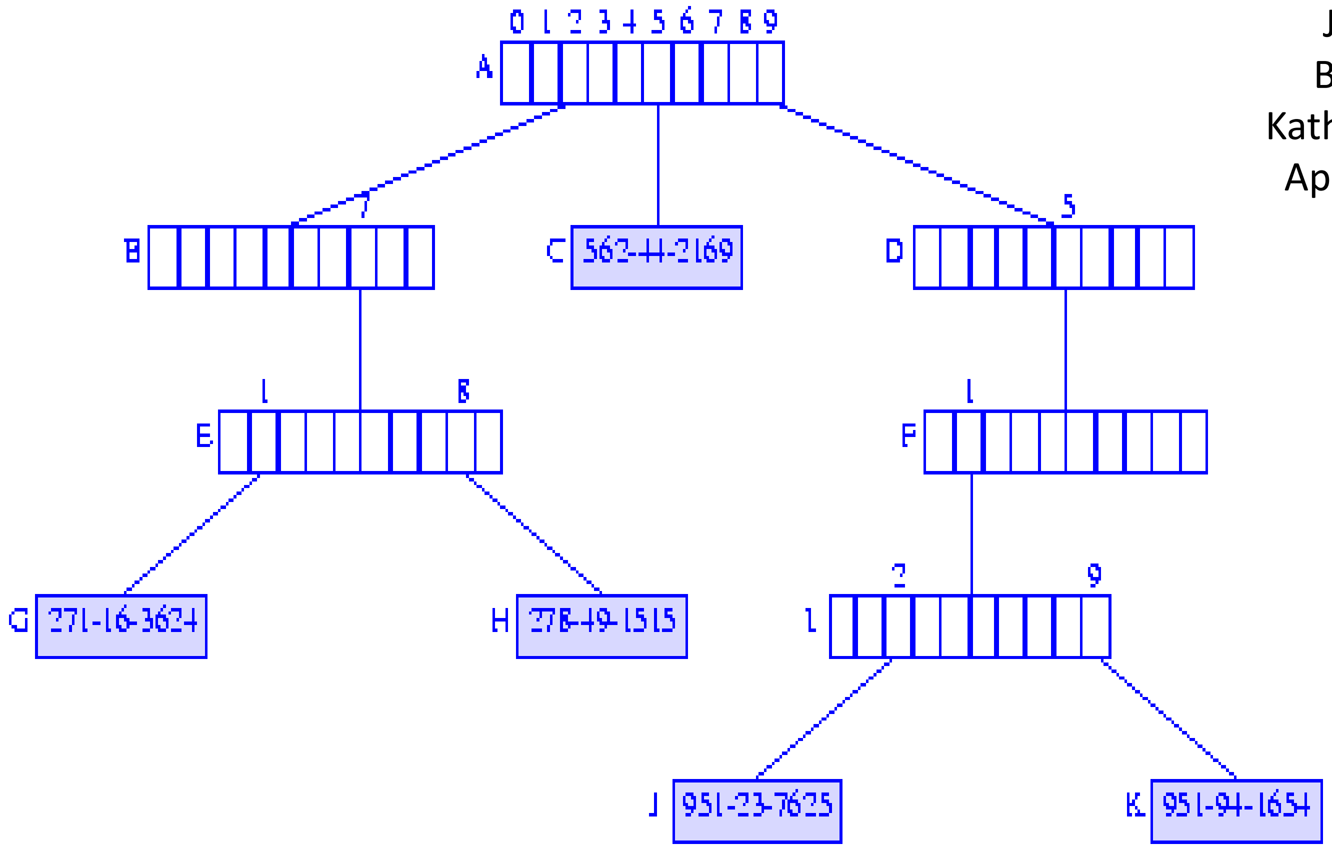
- If the keys are numeric, there would be 10 pointers in a node.



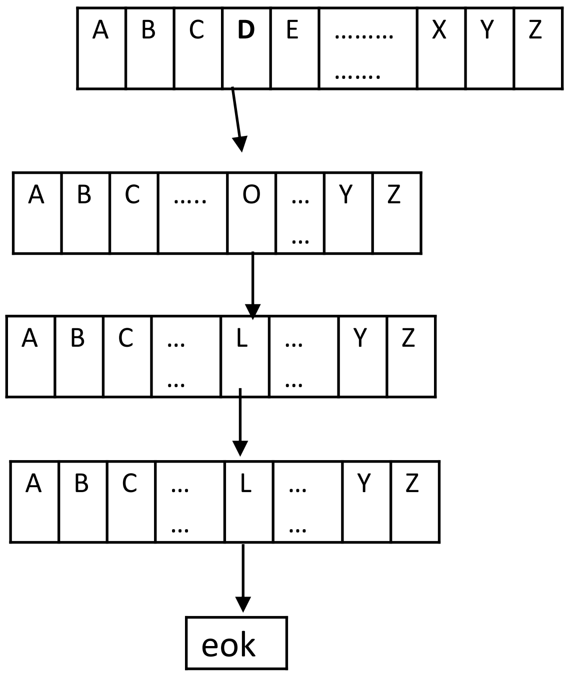
TRIE Trees – Numeric Keys : Example1

- If the keys are numeric, there would be 10 pointers in a node.
- Consider the SSN number as shown.

Name	Social Security Number (SS#)
Jack	951-94-1654
Jill	562-44-2169
Bill	271-16-3624
Kathy	278-49-1515
April	951-23-7625



- If the keys are Alphabetic, there would be 26 pointers.



Ex: The word DOLL has been stored as shown in the figure.

TRIE Trees – An Introduction

- An extra pointer corresponding to eok (end of key) or a flag with each pointer indicating that it point to a record rather than to a tree node. (normally $ symbol is used).
- A pointer in the node is associated with a particular symbol value based on its position in the node.
    - First pointer corresponds to the lowest value.
    - Second pointer to the second lowest and so forth.
- This way of implementation of a digital search tree is called a **TRIE** tree.
- The word ***TRIE*** is extracted from re***trie***val word.

TRIE Trees – Structure

- Tries are extremely special and useful data-structure that are based on the *prefix of a string*.
- Strings are stored in a top to bottom manner on the basis of their prefix in a TRIE.
- All prefixes of length 1 are stored at until level 1, all prefixes of length 2 are sorted at until level 2 and so on.

Suffix Trie:

- Suffix Trie is a space-efficient data structure to store a string that allows many kinds of queries to be answered quickly.
- Example:
  Text is "**banana\$**" where '\$' is the string terminating character.

Suffix Trie – Building

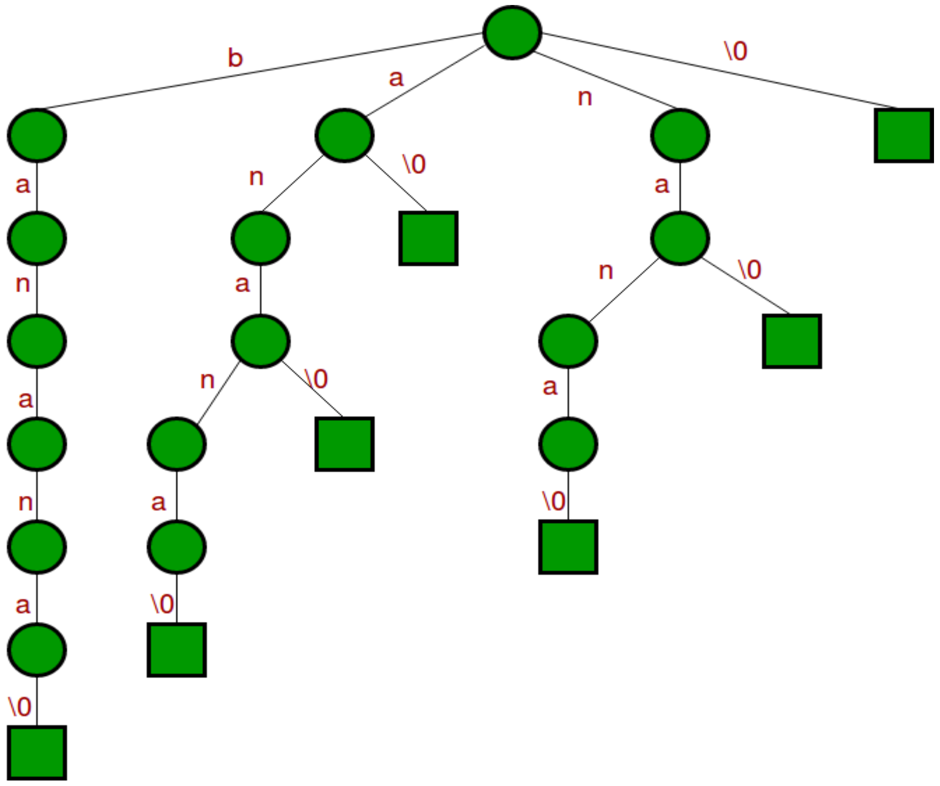
- A Trivial Algorithm for building a suffix tree.
  - Step1 : Generate all suffixes of a given text
  - Step2: Consider all suffixes as individual words and build a compressed trie.

  - Example1:
    Text is "**banana\$**" where '\$' is the string terminating character.

Following are the suffixes of Text

"banana/$"
"anana/$"
"nana/$"
"ana/$"
"na/$"
"a/$"
"/$"

Suffix trie



Suffix Trie – Building

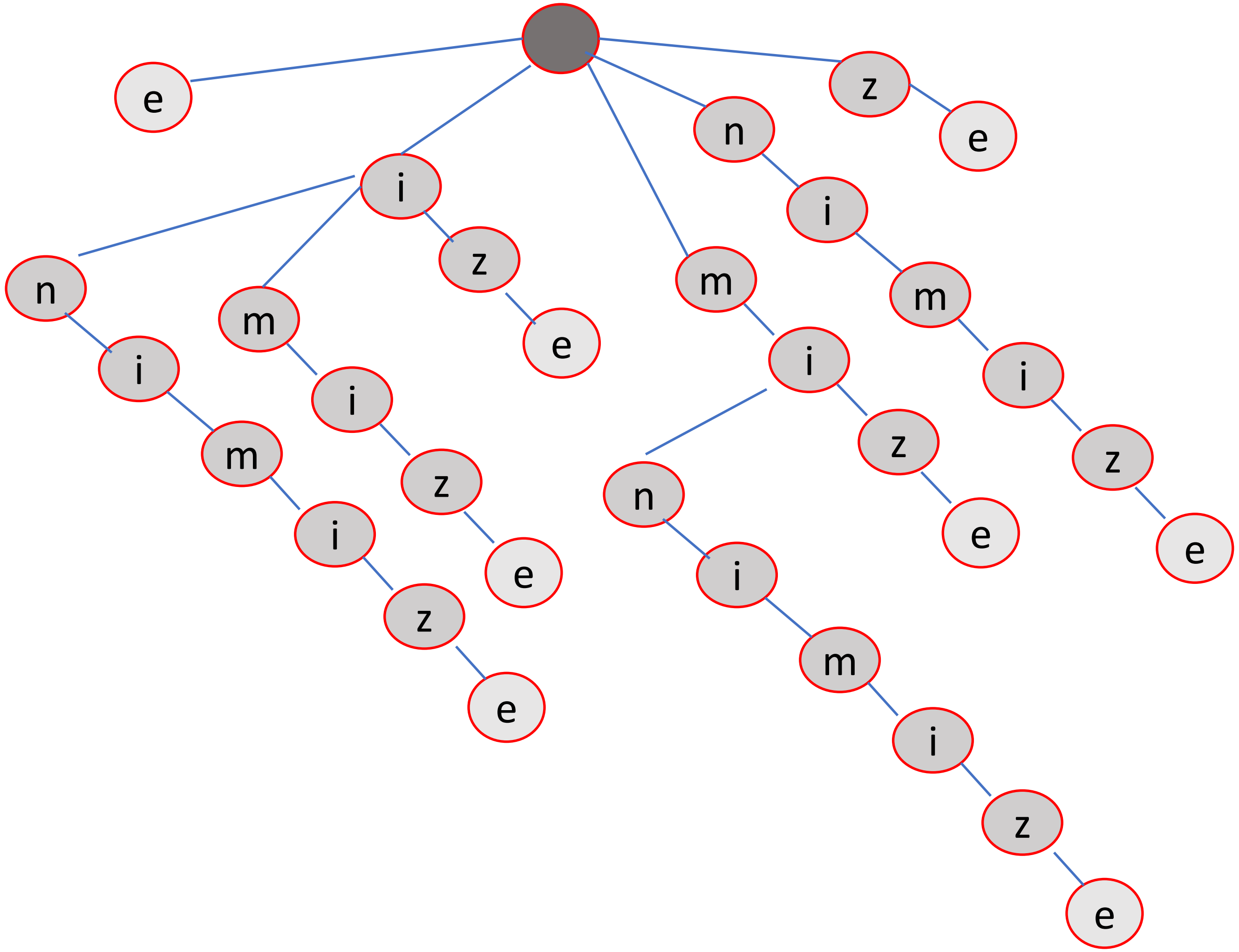
Example 2 : Generate the suffix trie for the word **minimize**

Step1. Generate all the suffixes of the word **minimize**.

e
ze
ize
mize
imize
nimize
inimize
minimize

S - set of strings to include in the suffix trie.

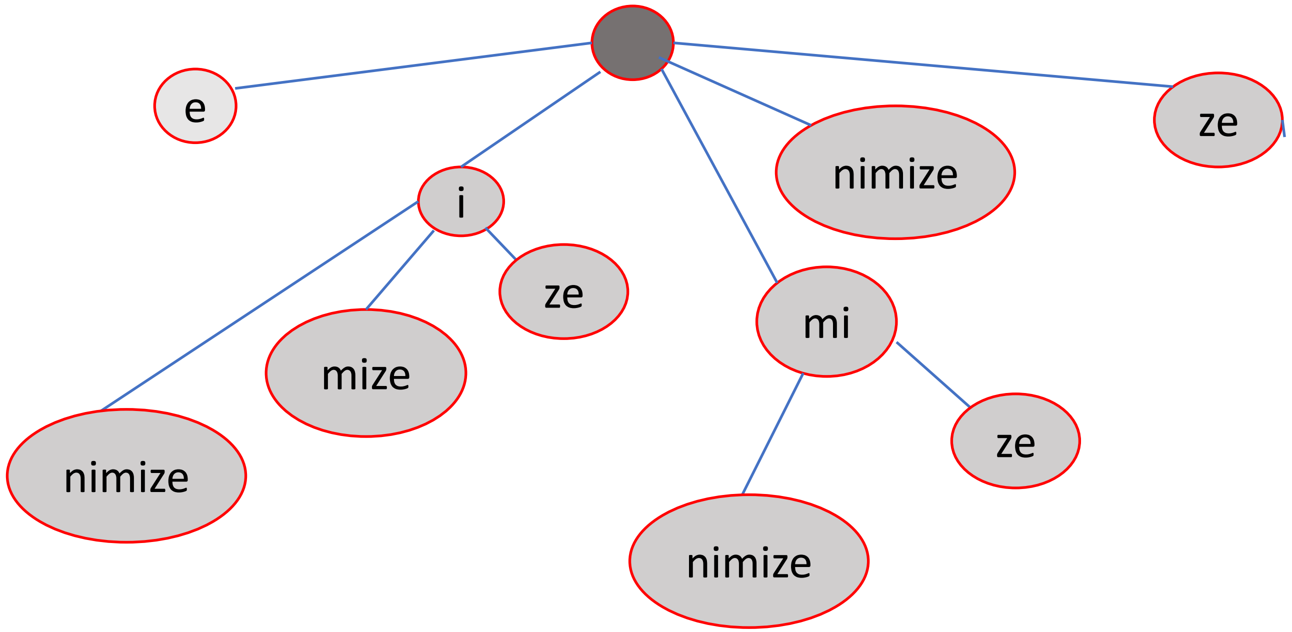
Suffix Trie – Building - for the word **minimize**



e
ze
ize
mize
imize
nimize
inimize
minimize

S - set of strings to include in the suffix trie

Suffix Trie – Compressed Trie

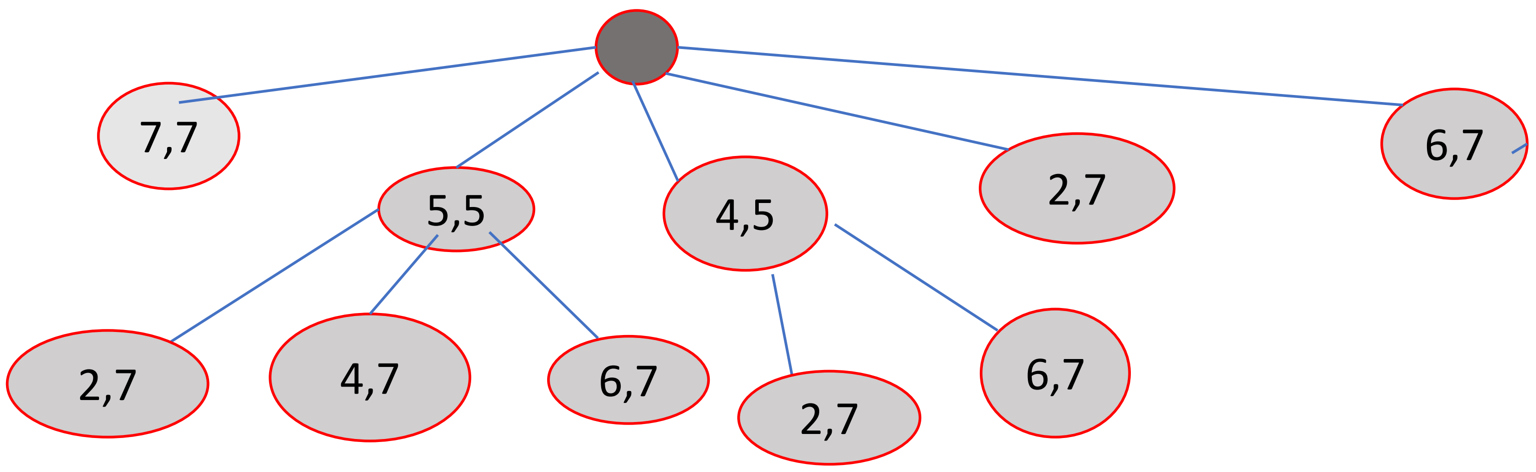


Data Structures and its Applications

Suffix Trie – Compressed Trie – using numbers

- Representation of Compressed trie using numbers - (**Indexes**)
- The indexes of the word is ...

0	1	2	3	4	5	6	7
m	**i**	**n**	**i**	**m**	**i**	**z**	**e**



Suffix Trees – Introduction

- A simple data structure for string searching
- It's a compressed Trie Tree
- Allow many fast implementations of many important string operations

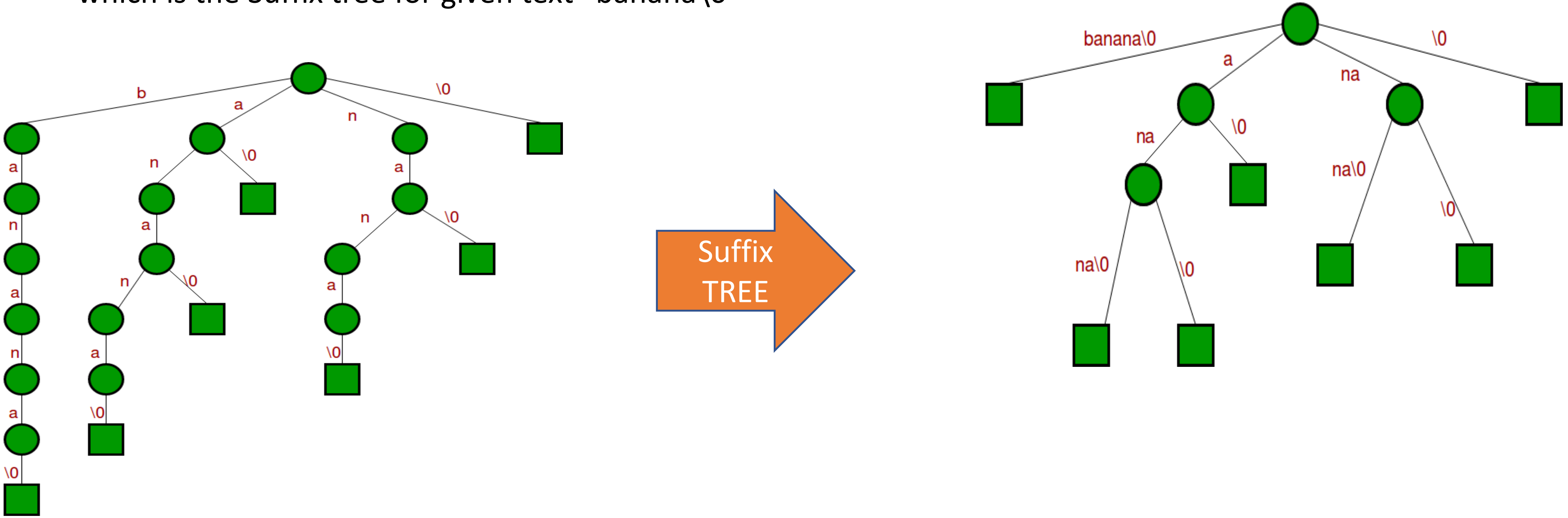
- Properties of a suffix trees:
  - ✓ A suffix tree for a text X of size n from an alphabet of size d.
  - ✓ Stores all the n(n-1) suffixes of X.
  - ✓ Supports arbitrary pattern matching and prefix matching queries

Example – Banana$

Data Structures and its Applications

Suffix Trees – Introduction

- Join chains of single nodes,to get the following compressed trie, which is the Suffix tree for given text “banana\0”



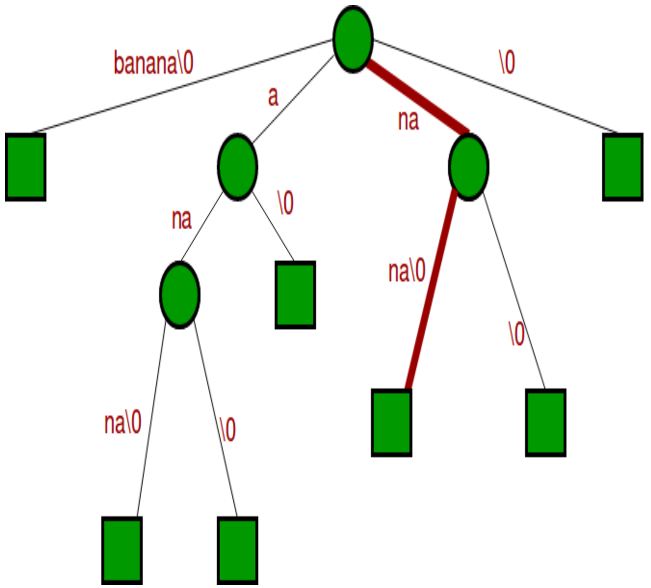
Search for a substring in a Suffix Tree

1) Starting from the first character of the pattern and root of Suffix Tree, do following for every character.

   i) For the current character of pattern, if there is an edge from the current node of suffix tree, follow the edge.

   ii) If there is no edge, print “pattern doesn’t exist in text” and return.

2) If all characters of pattern have been processed, i.e., there is a path from root for characters of the given pattern, then print “Pattern found”.



TRIE Trees – Applications, advantages and disadvantages

Applications:

- English dictionary
- Predictive text
- Auto-complete dictionary found on Mobile phones and other gadgets.

Advantages:

- Faster than BST
- Printing of all the strings in the alphabetical order easily.
- Prefix search can be done (Auto complete).

Disadvantages:

- Need for a lot of memory to store the strings,
- Storing of too many node pointers.

THANK YOU

V R BADRI PRASAD
Department of Computer Science & Engineering

badriprasad@pes.edu