LECTURE-27

Ties – Prefix and Suffix Tries

For Poll Ev

CS202: Data Structures (Fall 2025)

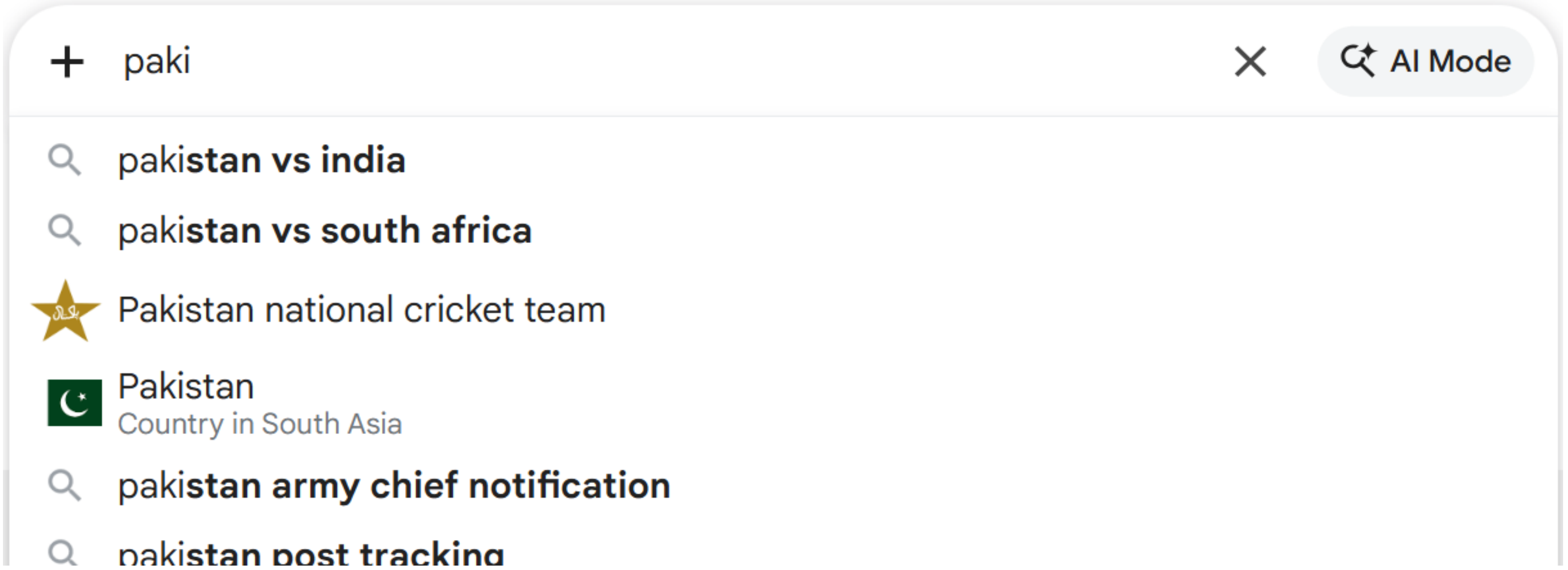
Dr Maryam Abdulghafur, Momina Khan
Department of Computer Science, SBASSE

Agenda

- Introduction to Tries
- Prefix and Suffix Tries – implementation and uses
- Compressed Tries

Tries

A trie (pronounced "**try**" from "re**trie**val") is a tree-based data structure for **fast pattern matching and prefix matching!**



paki

AI Mode

pakistan vs india

pakistan vs south africa

Pakistan national cricket team

Pakistan
Country in South Asia

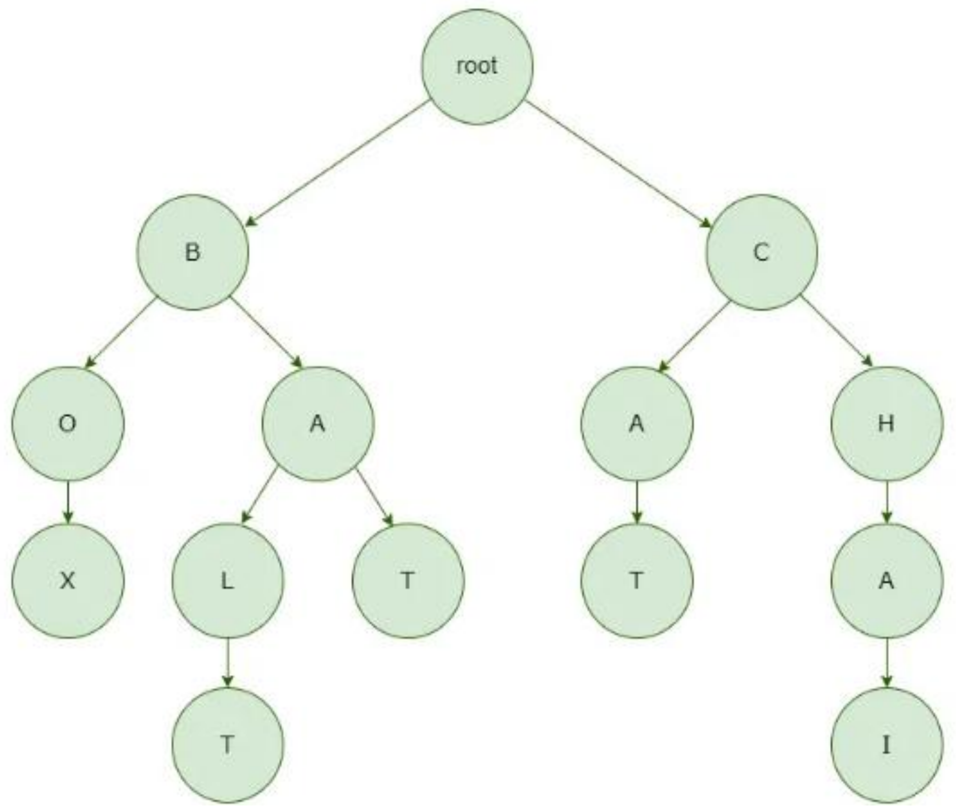
pakistan army chief notification

pakistan post tracking

Tries

- A trie represents a set of strings.
- It is constructed as a tree with every node holding a letter as the key except the root node.
- Every string is represented along a path.
- A node can have as many children as there are letters in the alphabet.

Trie Data Structure

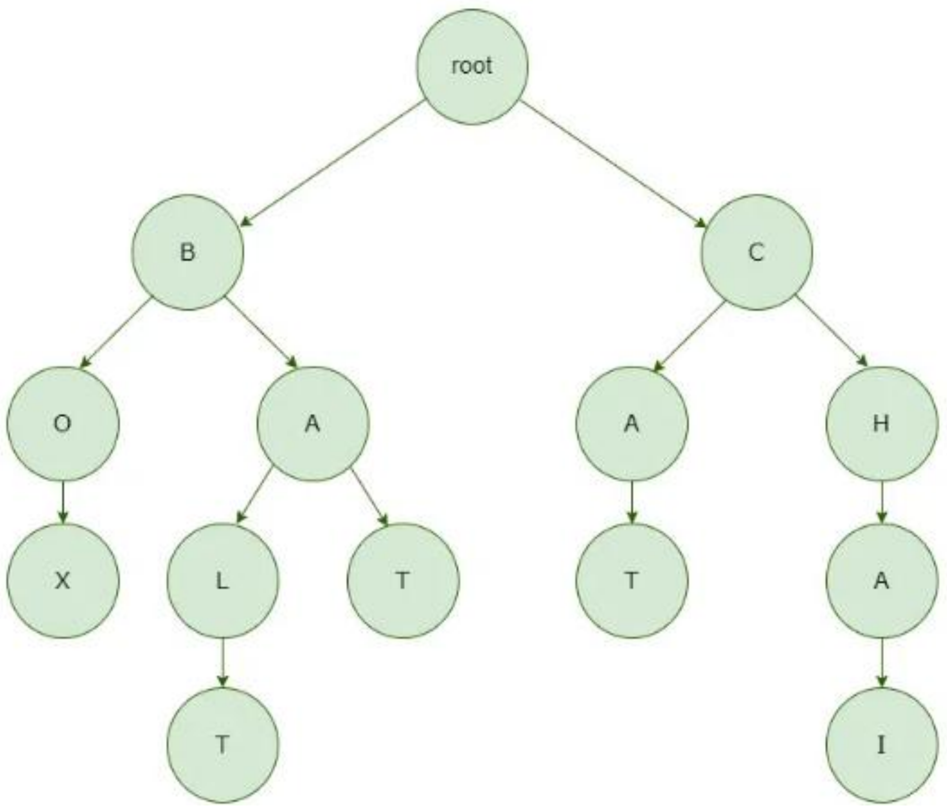


Tries

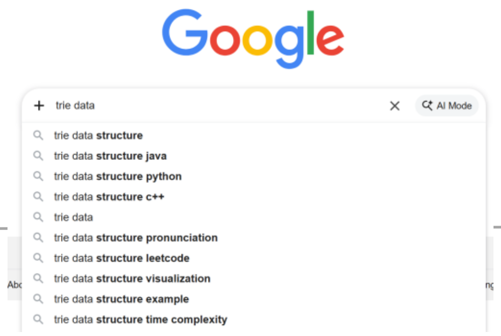
- What is the set of strings or words that are represented by the trie on the right?

- What determines the height of the trie?
  - The number of strings in the set? OR
  - The length of the longest string in the set?

Trie Data Structure



Prefix Tries


trie data

trie data **structure**
trie data **structure java**
trie data **structure python**
trie data **structure c++**
trie data
trie data **structure pronunciation**
trie data **structure leetcode**
trie data **structure visualization**
trie data **structure example**
trie data **structure time complexity**

A Prefix trie is used for auto complete where as soon as you start typing you are **magically** suggested words that complete your entry!

$ sign after a node signals end of a **valid** word!

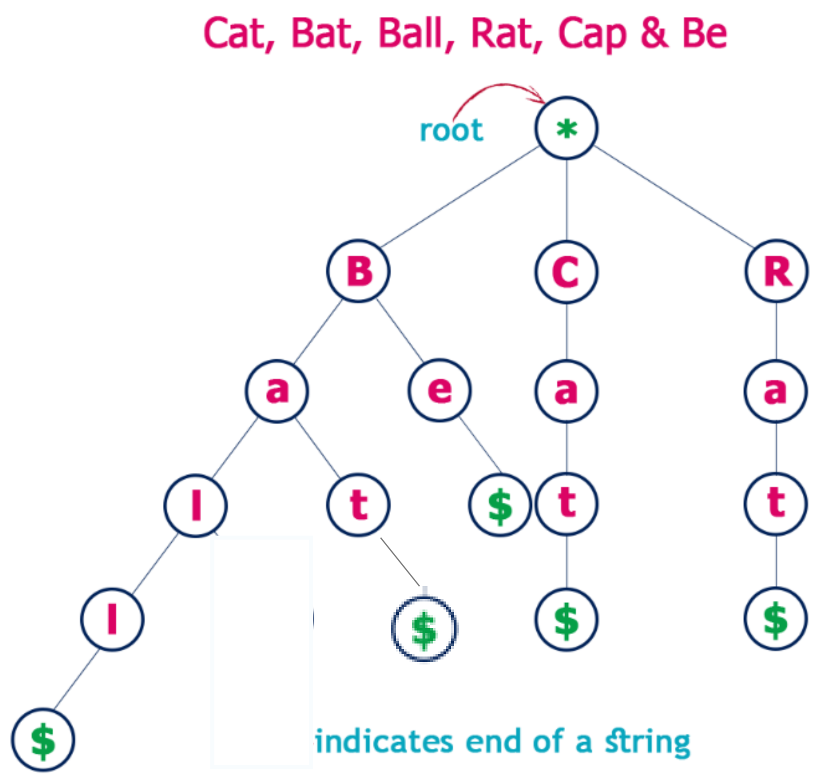


Image of node structure

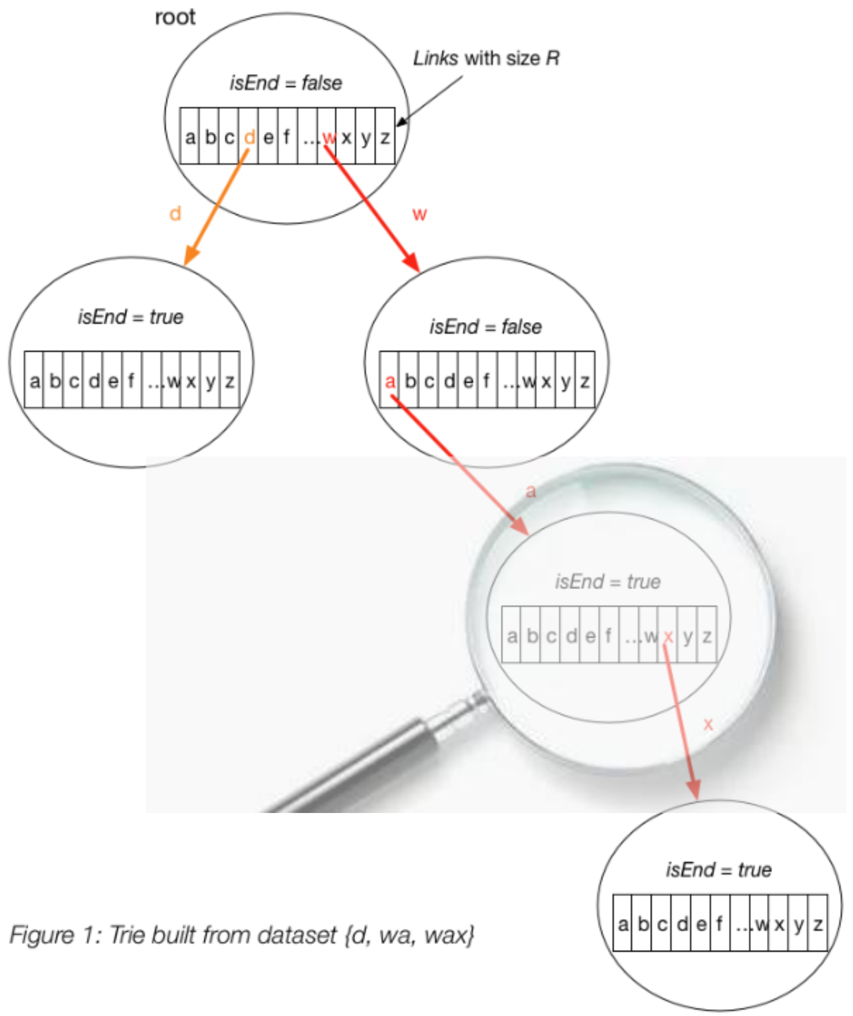
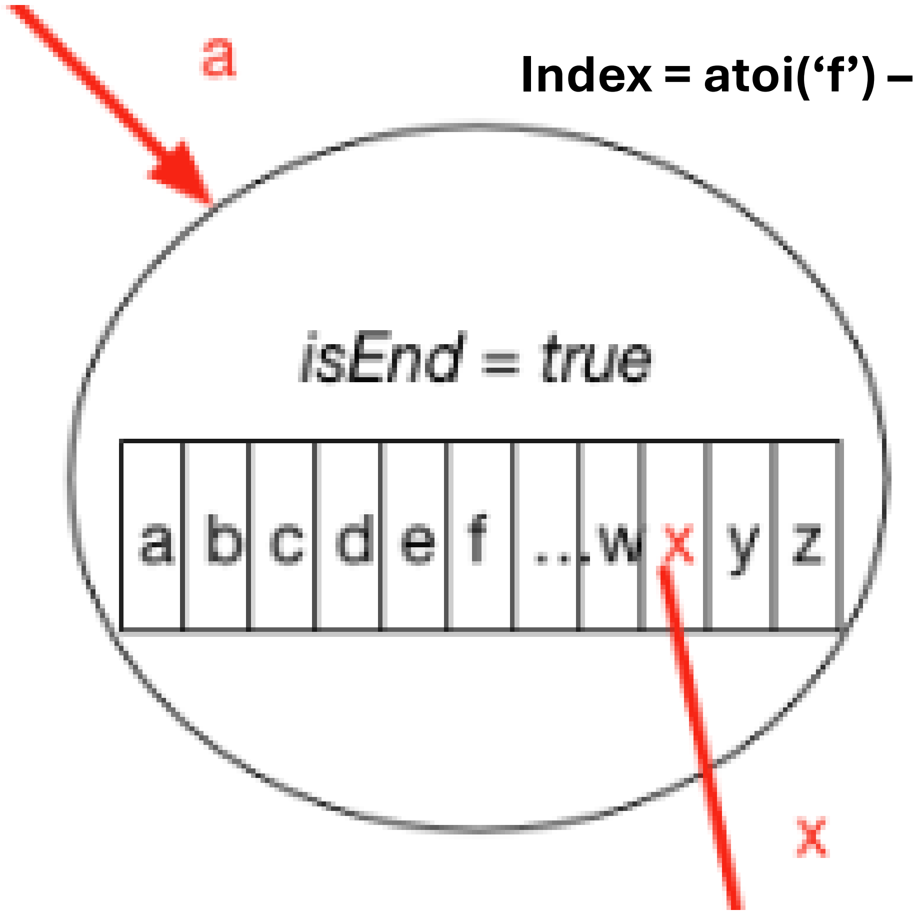


Figure 1: Trie built from dataset {d, wa, wax}

Index = atoi('f') – atoi('a')



Prefix Tries

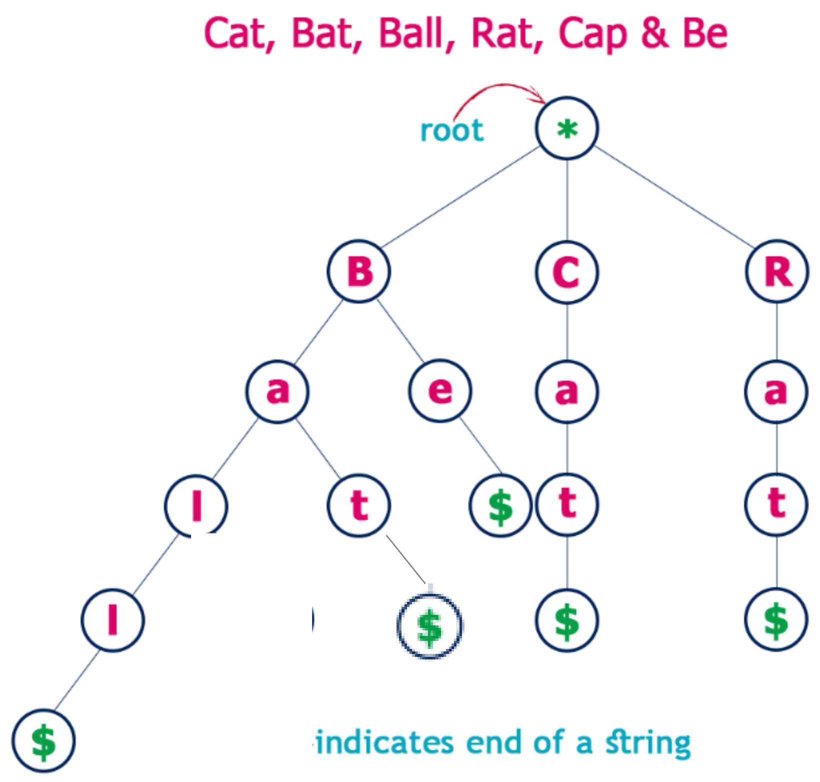
Q1. What operation do we have to do on the tree to reach all valid word suggestions for ‘B’?

Ans: We will start from root and go down the branch labelled ‘B’. All valid suggestions are strings found out after subtree travseral rooted at node ‘B’.

Q2. Suggestions for “Ba”?

Q3. How do I add a new word in the trie?

Ans: Walk the path from root to a substring that matches prefix of string you want to add. Wherever the path ends join a new set of nodes to complete path for new string.



How are tries implemented

1. Could be implemented using **array of pointers** in a node

OR

Could be implemented using a **Hash Table of pointers** in a node

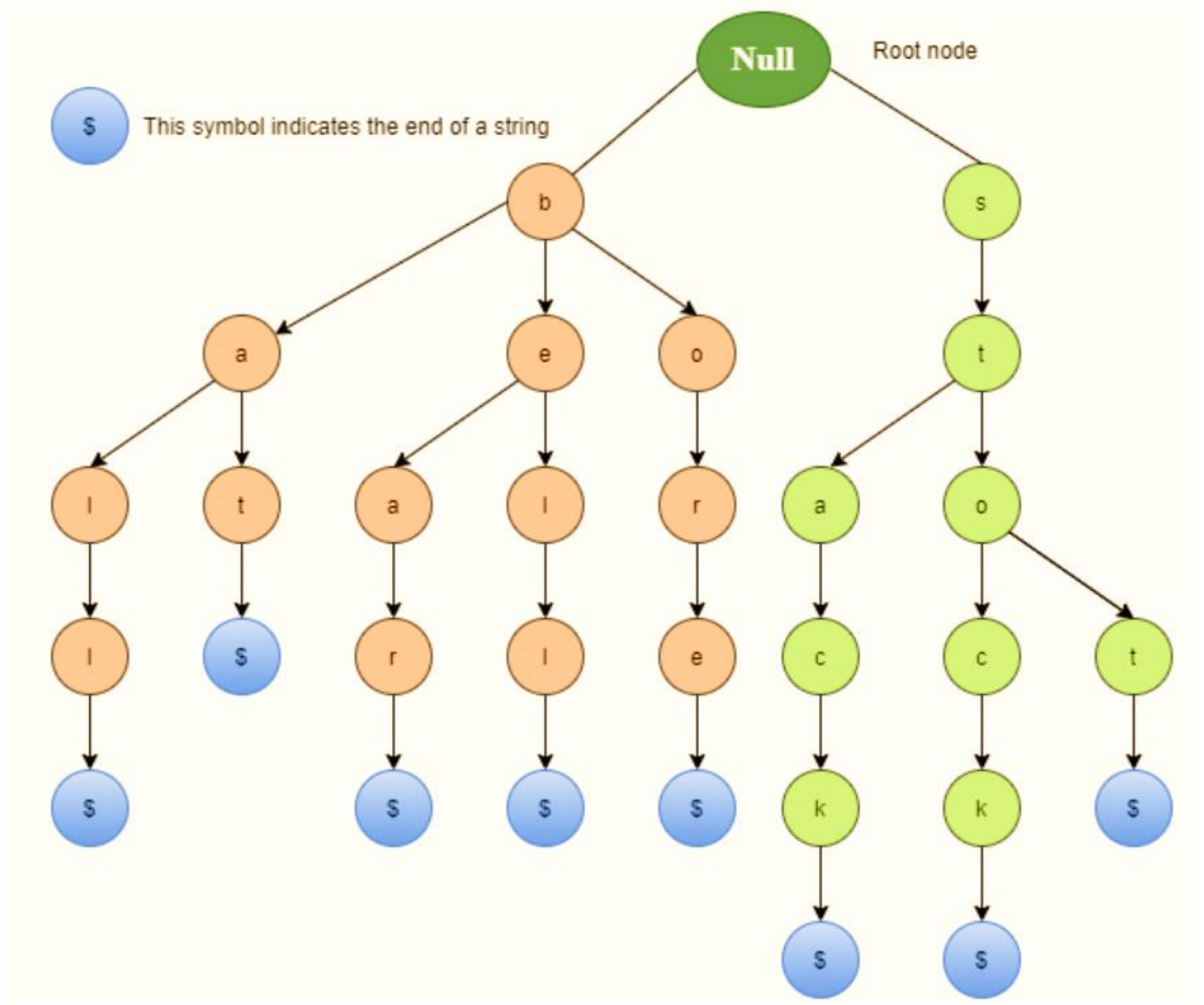
OR

Could be implemented using a **Balanced BSTree of pointers** in a node

Compressed Tries (Radix Tries)

Can we **reduce the space this trie is taking up** while representing the same set of strings?

We make a new node only when the node has atleast 2 children!



Node structure of a Trie

1. An Uncompressed Trie Node

   I. Letter marking the node for e.g. ‘F’
   II. Array of pointers sized at 26. (To accommodate entire alphabet set)
   III. isWord (A Boolean flag to show that a valid word ends at this node)

2. A Compressed Trie Node (Also called a Radix or Patricia Trie)

   I. String of letters that mark the collapsed node after compression eg. A node can be marked ‘FAA’ if three nodes with only one child each, ‘F’, ‘A’ and ‘A’ were merged into one node.
   II. Array of pointers sized at 26. (To accommodate entire alphabet set)
   III. isWord (A Boolean flag to show that a valid word ends at this node)

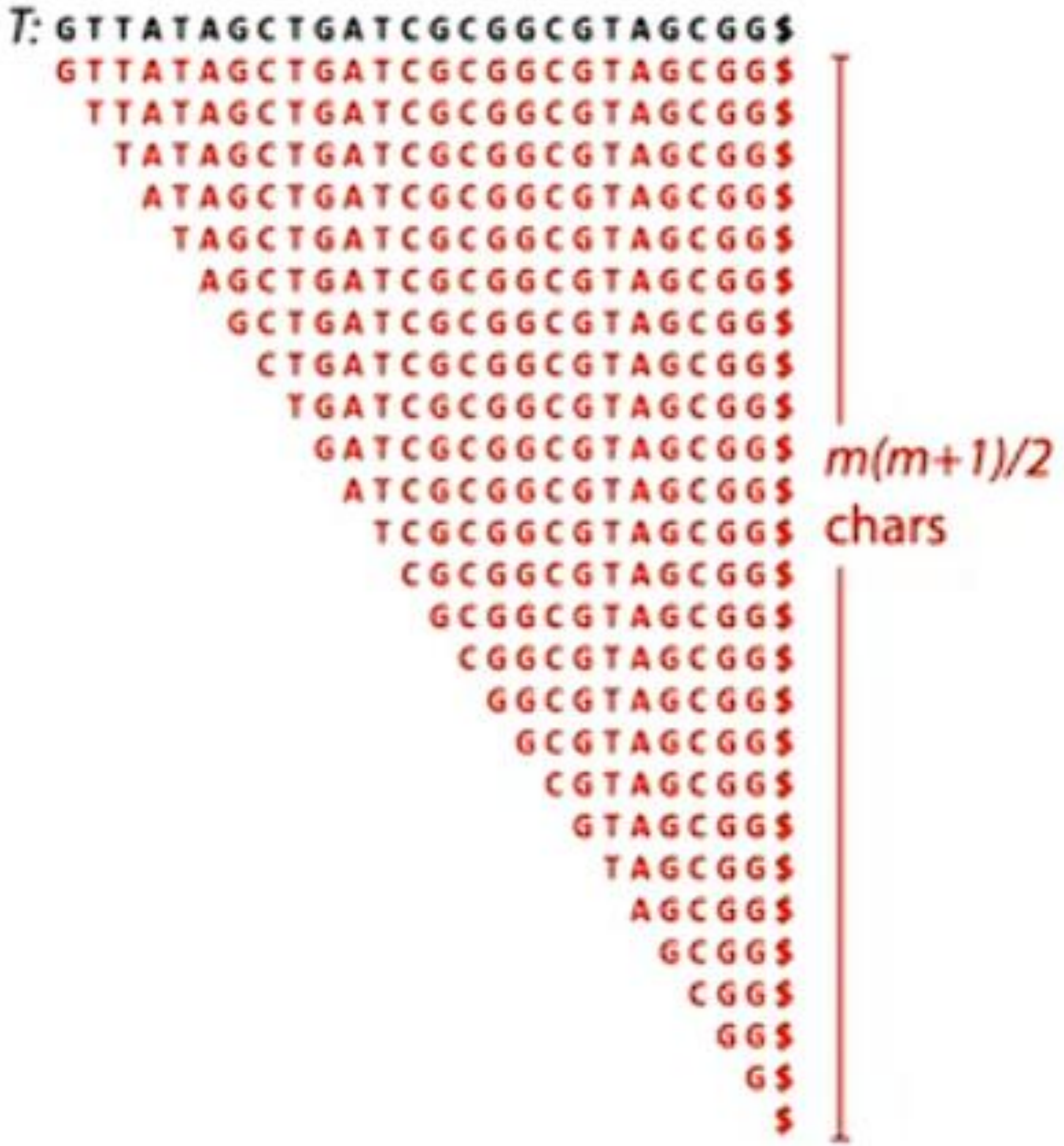
Suffix Tries

This is a trie that has **all suffixes** of a string **T**

String **T** Reminds you of?

A **DNA** sequence

Human Genome



```
T: GTTATAGCTGATCGCGGCGTAGCGG$
   GTTATAGCTGATCGCGGCGTAGCGG$
    TTATAGCTGATCGCGGCGTAGCGG$
     TATAGCTGATCGCGGCGTAGCGG$
      ATAGCTGATCGCGGCGTAGCGG$
       TAGCTGATCGCGGCGTAGCGG$
        AGCTGATCGCGGCGTAGCGG$
         GCTGATCGCGGCGTAGCGG$
          CTGATCGCGGCGTAGCGG$
           TGATCGCGGCGTAGCGG$
            GATCGCGGCGTAGCGG$
             ATCGCGGCGTAGCGG$
              TCGCGGCGTAGCGG$
               CGCGGCGTAGCGG$
                GCGGCGTAGCGG$
                 CGGCGTAGCGG$
                  GGCGTAGCGG$
                   GCGTAGCGG$
                    CGTAGCGG$
                     GTAGCGG$
                      TAGCGG$
                       AGCGG$
                        GCGG$
                         CGG$
                          GG$
                           G$
                            $
```

Suffix Tries

- Each root to leaf path is a suffix!

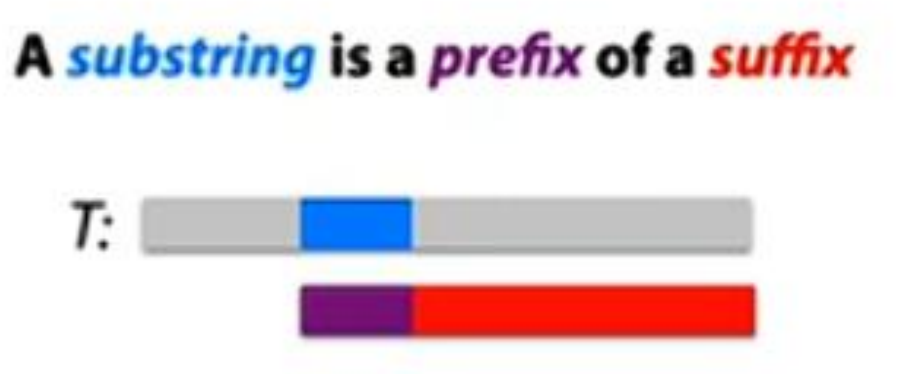
- $ or terminal node to a path marks all words in the trie!

  OR

  A Boolean flag '**isWord**'

Suffix Tries – Queries

- Is a **substring** present in T?



Each of substring of T is a prefix of some suffix of T!

- Count of times a **substring** occurs in T?

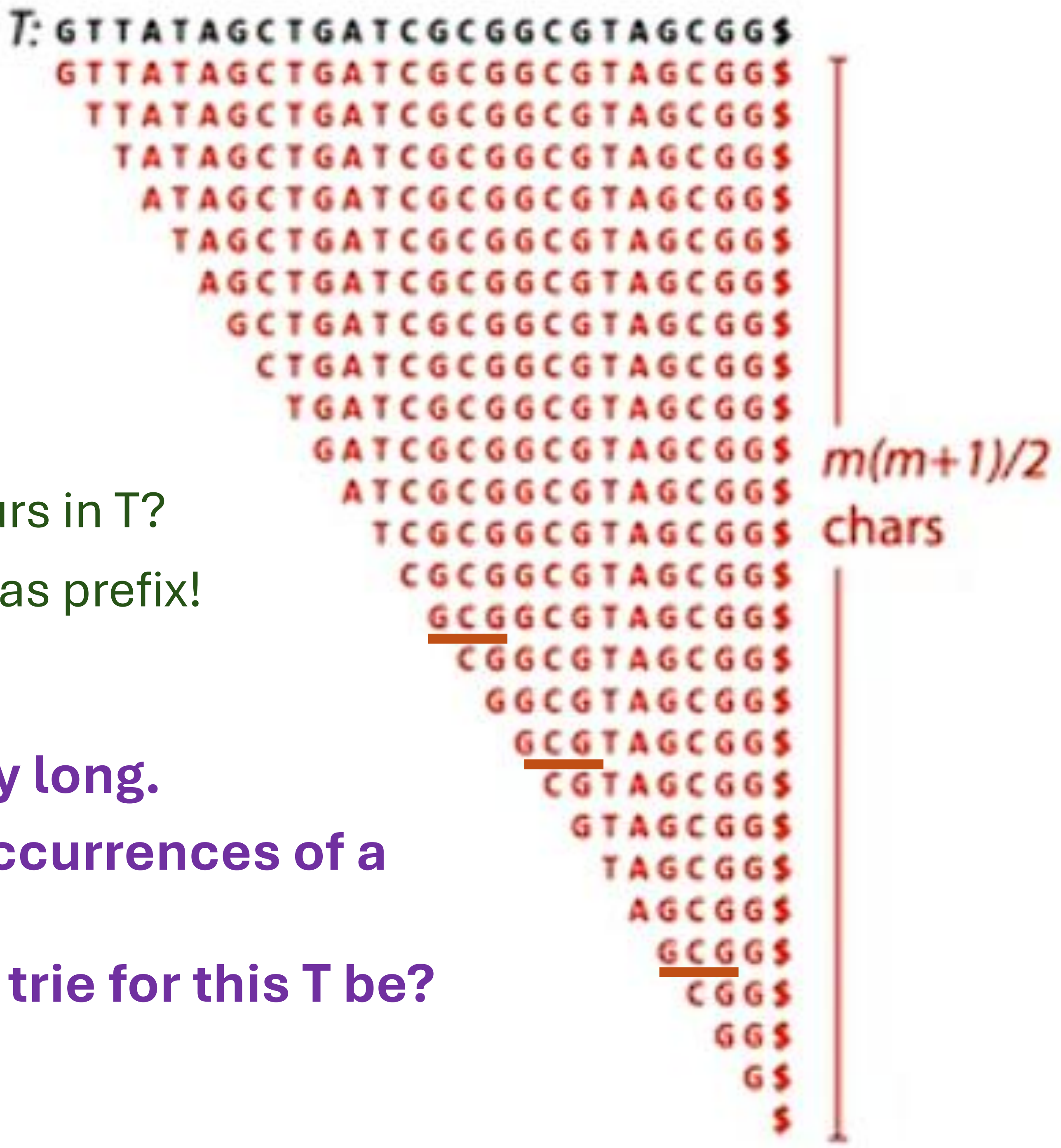
Suffix Tries

This is a trie that has **all suffixes** of a string **T**

Query:

Count of times a **substring "GCG"** occurs in T?

Ans: Count all suffixes that have "GCG" as prefix!



A human genome string is very very long.

- **It will be very quick to find all occurrences of a substring!**
- **How much space will the suffix trie for this T be?**