# Session - 17

## Trie

## CONTENTS

## 01

## Introduction to Trie

## Introduction to Trie

### What is a Trie?

- ➢ Trie = Tree-based data structure used for **storing strings**.
- ➢ Each node represents a character of a word.
- ➢ Common prefixes are shared to save space.
- ➢ Also known as: **Prefix Tree / Digital Tree**

*Example: Words "top", "toy", "team", "tea" share nodes in the same tree.*

## Introduction to Trie

### Why use Trie?

- ➢ Fast prefix-based search (0(L) time where L is word length)
- ➢ Efficient for autocomplete / spell checking
- ➢ Better than HashMap for prefix lookups
- ➢ Avoids collisions unlike hash tables

## 02

## Basic Operations

## Basic Operations

- **Insert(String word)** – Add word into Trie
- **Search(String word)** – Check if exact word exists
- **StartsWith(String prefix)** – Check if any word starts with prefix
- **Deletion** (rarely used, tricky due to shared prefixes)

## 03

## Trie Comparison

## Trie Comparison

**Advantage:** Search complexity with Trie is linear in terms of word (or key) length to be searched.

**Disadvantage:** The only problem with Tries is they require extra space.

### Comparison

**With BST -**

- ➢ If we store keys in balanced BST, we will need time proportional to M * log N for lookup.
- ➢ Where M is maximum string length and N is number of keys in tree. Using trie, we can search the key in O(M) time. So it is much faster than BST.

**With Hashing -**

- ➢ Hashing also provides word search in O(n) time on average.
- ➢ But the advantages of Trie are there are no collisions (like hashing) so worst case time complexity is O(n).
- ➢ Also, the most important thing is Prefix Search.
- ➢ With Trie, we can find all words beginning with a prefix. (This is not possible with Hashing)

## 04

## Time and Space Complexity

## Time and Space Complexity

| Operation | Time Complexity | Space Complexity |
|---|---|---|
| Insert | O(L) | O(N * L) |
| Search | O(L) | - |
| StartsWith | O(L) | - |

- L = length of word, N = total words
- Space may look huge, but common prefixes optimize it.

05

## Applications of Trie

## Applications of Trie

- Autocomplete / Auto-suggest
- Spell Checker
- IP Routing (binary Trie)
- T9 Predictive Text (old phones!)
- Word Games (e.g., Boggle, Scrabble solver)
- Longest Prefix Match

## 06

## Common Mistakes

## Common Mistakes

- Not checking for null nodes during search
- Overusing Trie for small datasets
- Confusing Trie with Tree or HashMap
- Ignoring space overhead in interview solutions

## 07

## Activity

## Activity Graph

- Implement Trie (Prefix Tree)
- Word Break
- Design Add and Search Words Data Structure