

B+ Trees

Drawback of B Tree


- Difficulty of traversing the keys sequentially
- It stores the data pointer (a pointer to the disk file block containing the key value), corresponding to a particular key value, along with that key value in each node of a B-tree.
- This technique, greatly reduces the number of entries that can be packed into a node of a B-tree, thereby contributing to the increase in the number of levels in the B-tree, hence increasing the search time of a record.

Soln-

- B+ tree eliminates the above drawback by storing data pointers only at the leaf nodes of the tree.

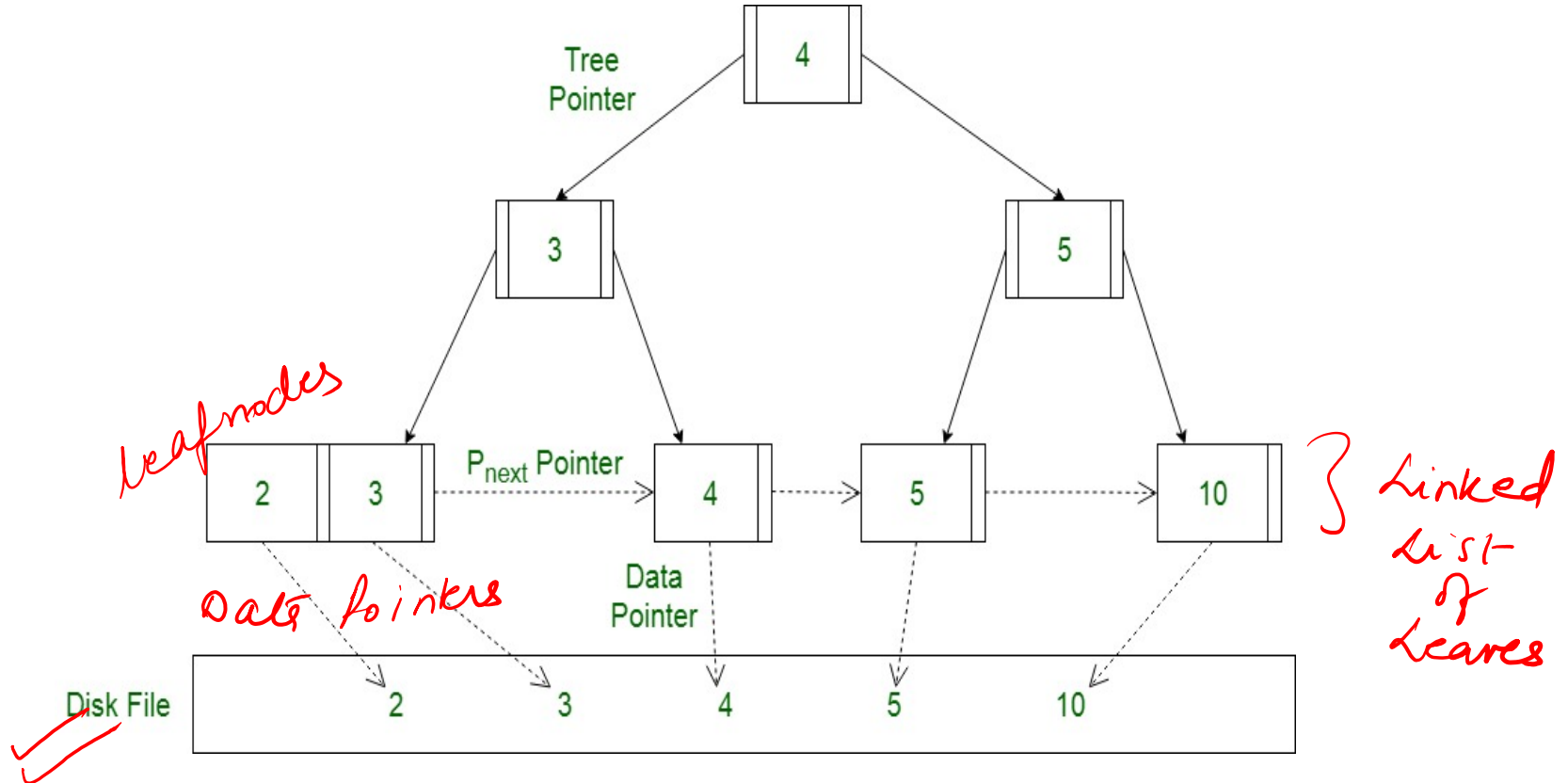
Drawback of B Tree

Soln-

-  B+ Tree
- A variant of the basic B-Tree structure, B+ Tree retains the rapid random access property of the B Tree,
- Also allowing Rapid Sequential Access

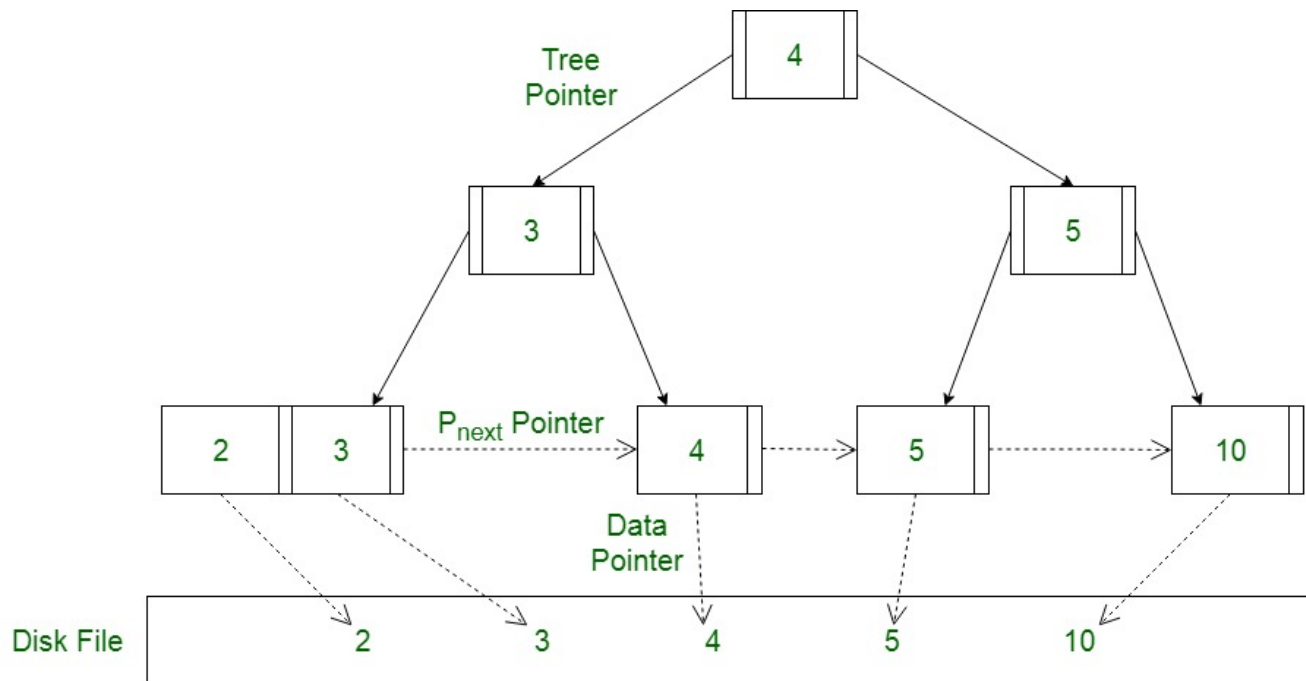
B+ Tree

- In B+ Tree, Since data pointers are present only at the leaf nodes,
- The leaf nodes must necessarily store all the key values along with their corresponding data pointers to the disk file block, in order to access them.



B+ Tree

- All the keys are maintained in the leaves and are replicated in non-leaf nodes to define paths for loading individual records
- The Leaves are linked together to provide a sequential path for traversing the keys in the tree
- The linked list of leaves is called a Sequence set.



Advantages of B+ Tree

- A B+ tree with 'l' levels can store more entries in its internal nodes compared to a B-tree having the same 'l' levels.
- This accentuates the significant improvement made to the search time for any given key.
- Having lesser levels and presence of P_{next} pointers imply that B+ tree are very quick and efficient in accessing records from disks.

Application of B+ Trees:

- Multilevel Indexing
- Faster operations on the tree (insertion, deletion, search)
- Database indexing

Insertion in B+ Trees:

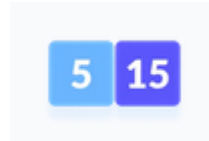
Order=3

Max elements=2

- Insert 5



- Insert 15 ✓

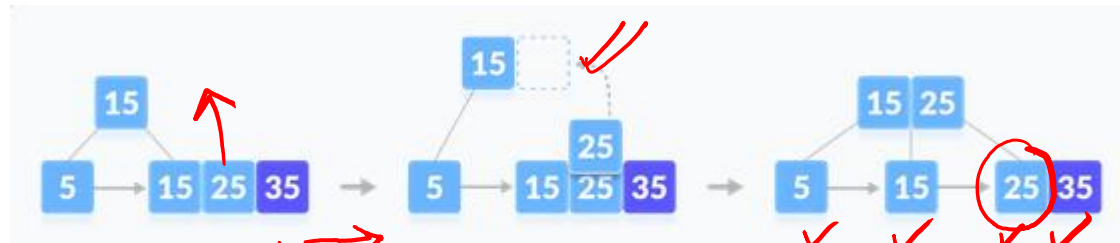


Order = 3
max element = 2

- Insert 25-**Case 1**: Overflow in leaf node-Median key goes up to the parent and in the Right Child created both



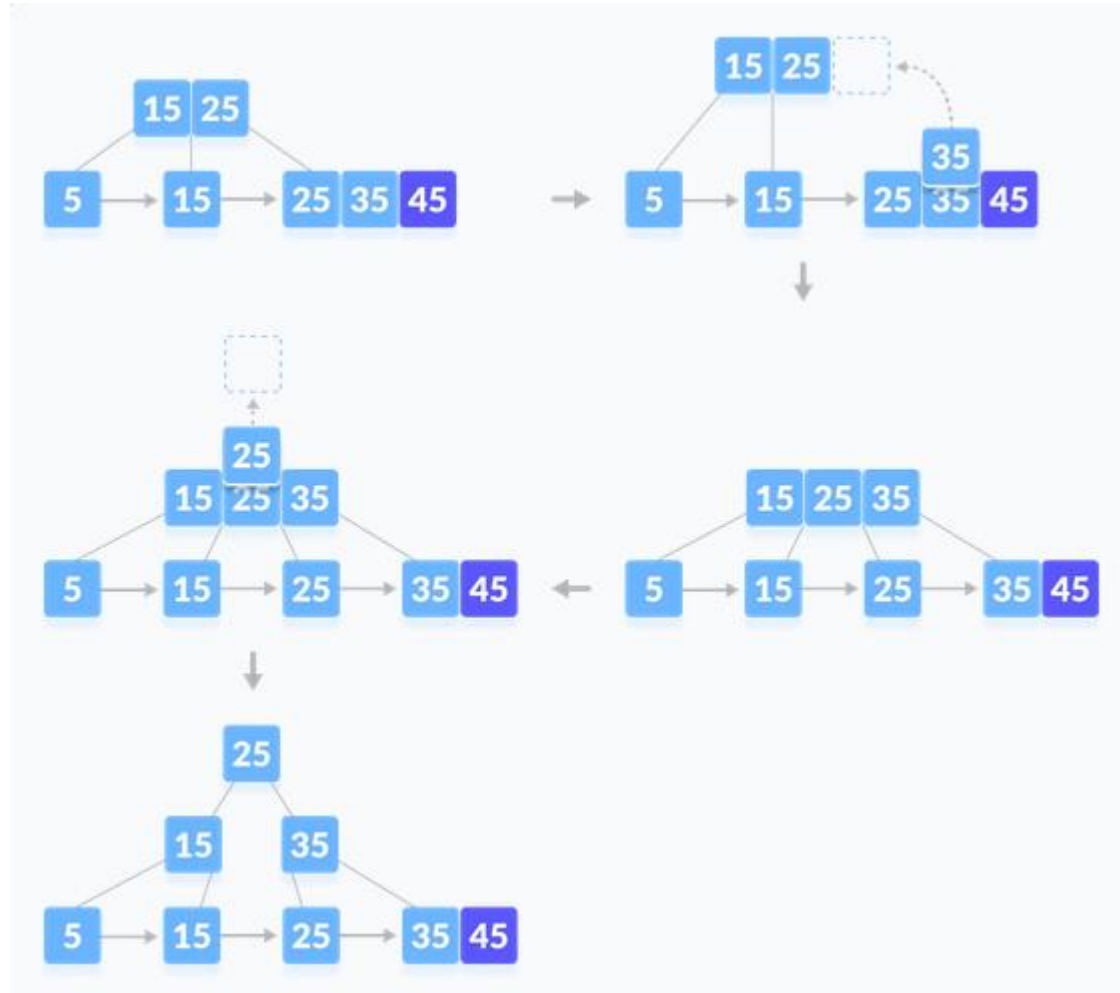
- Insert 35-**Case 1**: Overflow in leaf node-Median key goes up to the parent and in the Right Child created both



✓ ✓ ✓ ✓ linked list

Insertion in B+ Trees:

- Insert 45



Case 2: Overflow in non-leaf node--Median key goes up to the parent only

Tries

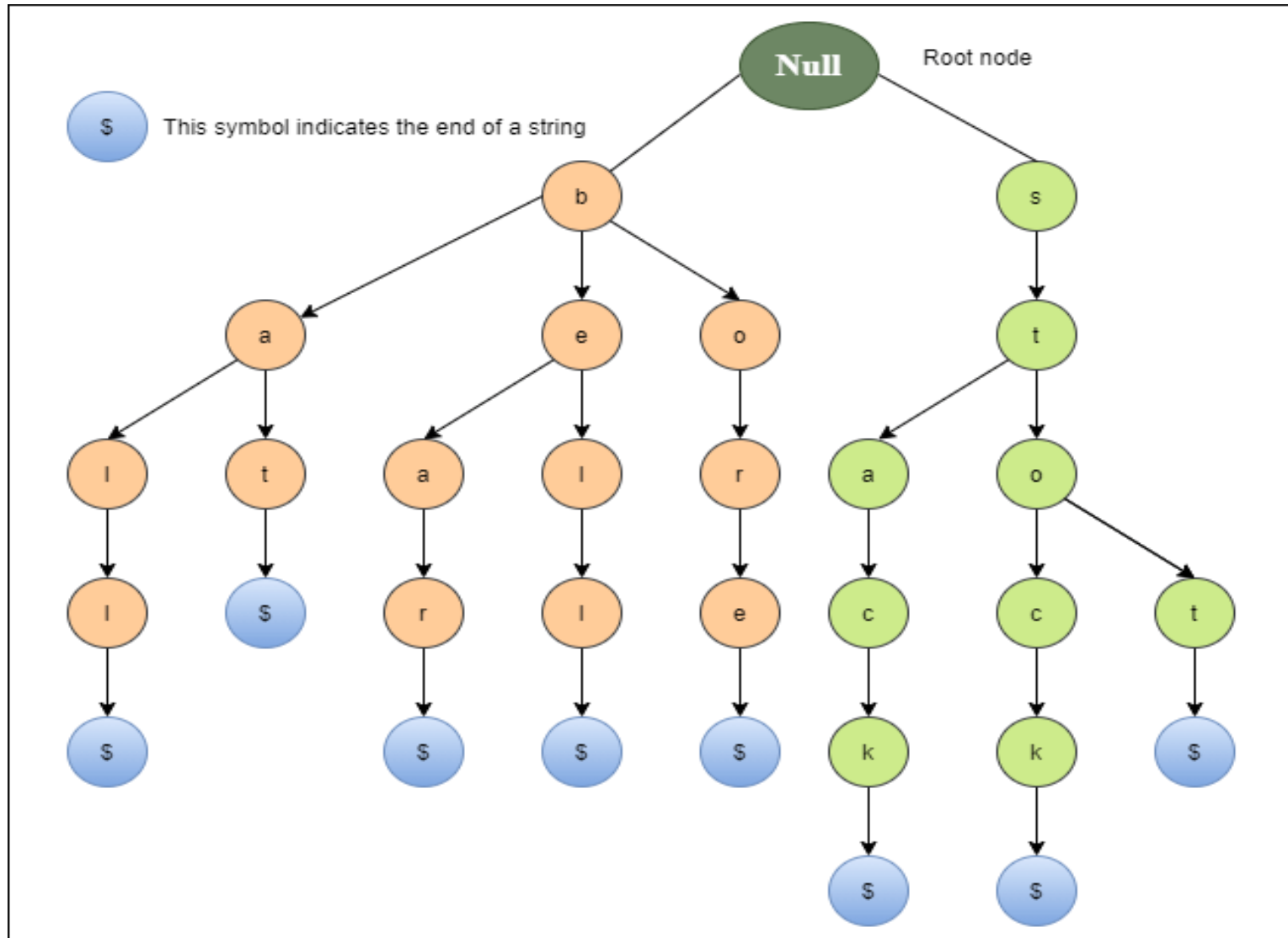
- The word "**Trie**" is an excerpt from the word "**retrieval**".
- Trie is a sorted tree-based data-structure that stores the set of strings

Properties of the Trie for a set of the string:

- The root node of the trie always represents the null node.
- Each child of node is sorted alphabetically.
- Each node can have a maximum of **26** children (A to Z).
- Each node (except the root) can store one letter of the alphabet.

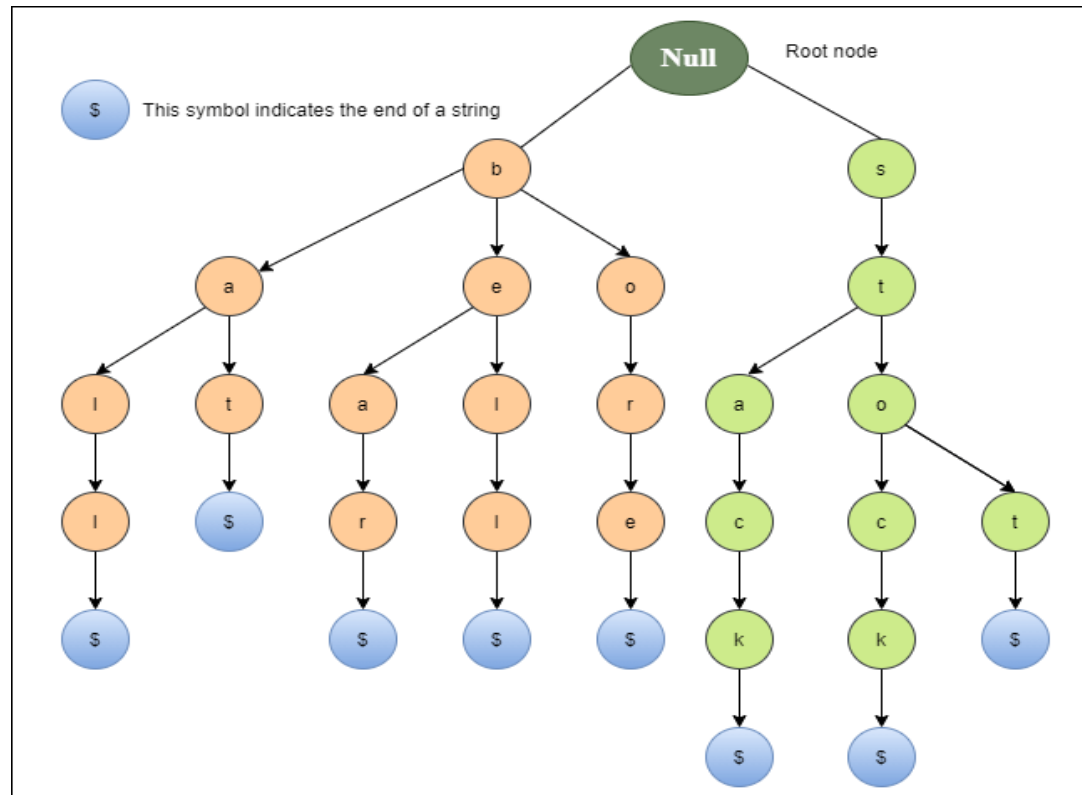
Properties of the Trie for a set of the string:

A trie representation for the bell, bear, bore, bat, ball, stop, stock, and stack.



Tries

- Every node of Trie consists of multiple branches. Each branch represents a possible character of keys.
- We need to mark the last node of every key as end of word node. A Trie node field *isEndOfWord* is used to distinguish the node as end of word node.

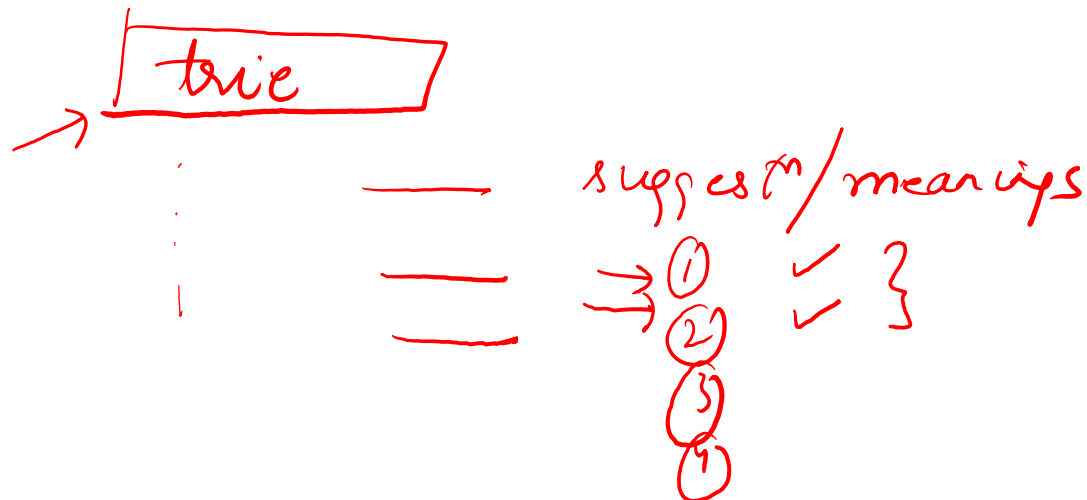


Applications of Trie

1. Spell Checker

Spell checking is a three-step process.

- 1) First, look for that word in a dictionary,
- 2) generate possible suggestions, and
- 3) then sort the suggestion words with the desired word at the top.



Applications of Trie

1. Spell Checker

- Trie is used to store the word in dictionaries.
- The spell checker can easily be applied in the most efficient way by searching for words on a data structure.
- Using trie not only makes it easy to see the word in the dictionary, but it is also simple to build an algorithm to include a collection of relevant words or suggestions.

Applications of Trie

✓ 2. Auto-complete

- Auto-complete functionality is widely used on text editors, mobile applications, and the Internet.
- It provides a simple way to find an alternative word to complete the word for the following reasons.
- It provides an alphabetical filter of entries by the key of the node.
- We trace pointers only to get the node that represents the string entered by the user.
- As soon as you start typing, it tries to complete your input.

Applications of Trie

3. Browser history

It is also used to complete the URL in the browser.

The browser keeps a history of the URLs of the websites you've visited.

Advantage of Trie

- 1) With Trie, we can insert and find strings in $O(L)$ time where L represent the length of a single word.
 - This is obviously faster than BST.
 - This is also faster than Hashing because of the ways it is implemented. We do not need to compute any hash function. No collision handling is required (like we do in open addressing and separate chaining)
- 2) Another advantage of Trie is, we can easily print all words in alphabetical order which is not easily possible with hashing.
- 3) We can efficiently do prefix search (or auto-complete) with Trie.

Drawbacks of Trie

- The main disadvantage of tries is that they need a lot of memory for storing the strings.
- For each node we have too many node pointers (equal to number of characters of the alphabet)