Amrita School of Engineering

B. Tech CSE-Artificial Intelligence

DATA STRUCTURES AND ALGORITHMS –2

Semester –3

Term Project

AMRITA VISHWA VIDYAPEETHAM

AMRITA SCHOOL OF ENGINEERING, COIMBATORE -641105

BONAFIDE CERTIFICATE

This is to certify that the major project report entitled for "Data Structures And Algorithms –2"

Submitted by

Batch A -group 09

Sangam Ganesh Babu	[CB.EN.U4AIE21056]
Challa Yoganandha Reddy	[CB.EN.U4AIE21008]
Rangisetti Sai Raghavendra	[CB.EN.U4AIE21049]
Gajula Sri Vatsanka	[CB.EN.U4AIE21010]

In partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology in Data Structures and algorithms-2 is a Bonafede record of the work carried out under my guidance and supervision at Amrita School of Engineering, Coimbatore.

Signature

Dr. Sachin Kumar,

Department of CEN.

This project was evaluated by us on ……

What is a Trie?

- Trie basically comes from the word Retrieval.
- The main purpose of this data structure is to retrieve stored information very fast.

Trie Data Structure :

- A Trie (prefix tree) is a tree-based data structure that is used to store an associative array where the keys are sequences (usually strings).
- It is also known as a prefix tree because it stores the prefixes of keys.
- Each node in the Trie represents a single character in a key, and each edge represents a character transition from one node to another.
- The Trie is a compact data structure that can be used to implement many different types of data structures, such as dictionaries and spell-checking algorithms.

Properties:

- The root node represents an empty string and doesn't have any data.
- Each subsequent level of the Trie represents a single character in the key.
- The last node of a key represents the end of the key and is marked as such.
- The Trie can be searched quickly because each character in the key is used to traverse the Trie.
- The Trie can be used to implement many different types of data structures, such as dictionaries, spell-checking algorithms, and autocomplete features.

Advantages:

1. Space-efficient
2. Fast Prefix Search
3. Easy Insertion and Deletion
4. Suitable for Dictionary or Autocomplete
5. Suitable for IP routing

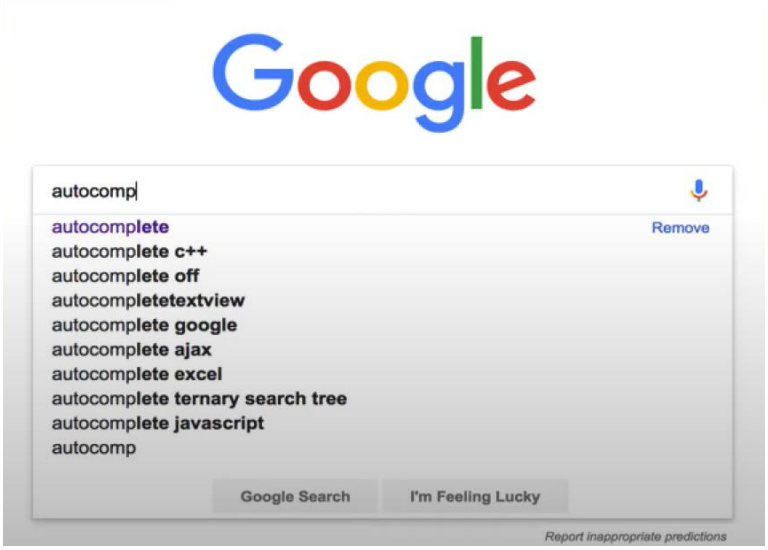
Disadvantages:

1. Increased Space Complexity
2. Higher Time Complexity for Searching
3. Extra space
4. Not suitable for large datasets
5. Not suitable for exact match

Applications –

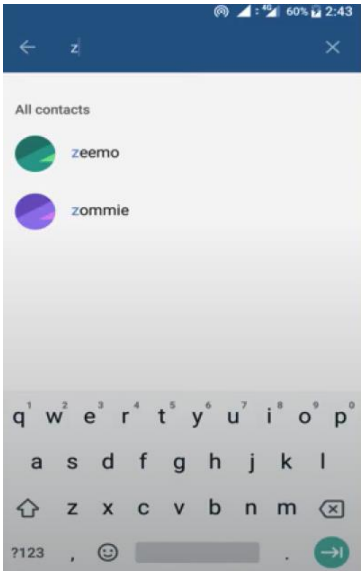
1. Auto-Complete words

- Autocomplete feature is implemented by Tries.
- Many websites have used autocomplete feature, which suggest user rest of the word, while user is typing



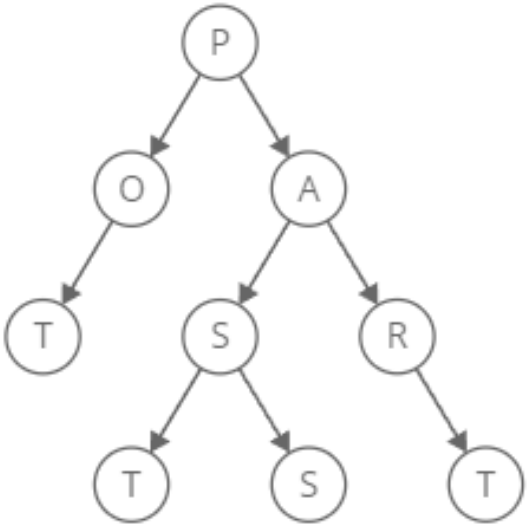
2. Search Contacts in phone

- Searching a person contact number in contact list is efficiently implemented by Trie. As soon as user enters letters the application auto suggest the name of the person.



Example:

POT,PAST,PART,PASS

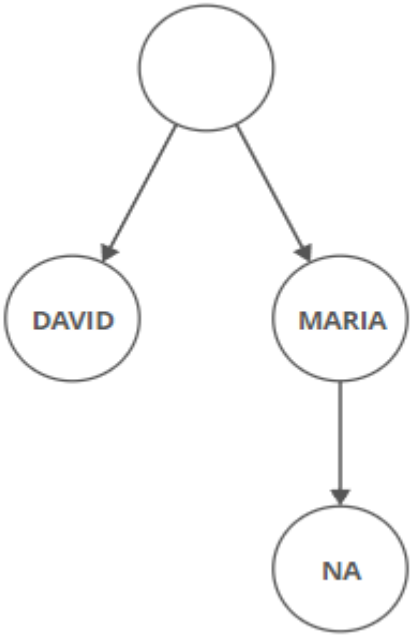


Radix Trees:

A Radix tree is also like a trie data structure ,but it saves space by combining nodes together if they only have one child.

Example: -

Maria ,Marina ,David



Phonebook:

A phonebook, also known as a telephone directory, is a list of phone numbers and associated information such as names, addresses, and other contact details. Phonebooks can be in the form of physical books or digital databases. They provide a way for people to look up phone numbers and contact information for individuals or businesses. Phonebooks are typically organized alphabetically by last name or by category (such as businesses or government agencies).

CODE

```java
import java.util.ArrayList;    TrieNode.java is a non-projec
import java.util.HashMap;

class trienode {
    HashMap<Character, trienode> child;
    boolean isLast;

    public trienode() {
        child = new HashMap<Character, trienode>();
        for (char i = 'a'; i <= 'z'; i++)
            child.put(i, value: null);
        isLast = false;
    }
}

// phonebook directory
class Directory {
    HashMap<String, Integer> phonebook;
    Trie trie;

    public Directory() {
        phonebook = new HashMap<>();
        trie = new Trie();
    }

    public void addContact(String name, Integer phoneno) {
        phonebook.put(name, phoneno);
        trie.insert(name);
    }

    public void Combinations(String query) {
        trie.getContacts(query, phonebook);
    }
}
```

```java
class Trie {
    trienode root;

    public Trie() {
        root = new trienode();
    }

    // Insert a Contact into the Trie
    public void insert(String s) {
        int len = s.length();
        // 'itr' is used to iterate the Trie Nodes
        trienode itr = root;
        for (int i = 0; i < len; i++) {
            // Check if the s[i] is already present in Trie
            trienode nextNode = itr.child.get(s.charAt(i));
            if (nextNode == null) {
                // If not found then create a new trienode
                nextNode = new trienode();
                // Insert into the HashMap
                itr.child.put(s.charAt(i), nextNode);
            }
            // Move the iterator('itr') ,to point to next Trie Node
            itr = nextNode;
            // If its the last character of the string 's' then mark 'isLast' as true
            if (i == len - 1)
                itr.isLast = true;
        }
    }
```

```java
    public void displayContactsUtil(trienode curNode,
            String prefix, ArrayList<String> contactsWithPrefix) {
        // If yes then display the string found so far
        if (curNode.isLast) {
            contactsWithPrefix.add(prefix);
        }
```

```java
        for (char i = 'a'; i <= 'z'; i++) {
            trienode nextNode = curNode.child.get(i);
            if (nextNode != null) {
                displayContactsUtil(nextNode, prefix + i, contactsWithPrefix);
            }
        }
    }

    void getContacts(String str, HashMap<String, Integer> phonebook) {
        trienode prevNode = root;
        String prefix = "";
        int len = str.length();
        int i;
        for (i = 0; i < len; i++) {
            // 'str' stores the string entered so far
            prefix += str.charAt(i);
            // Get the last character entered
            char lastChar = prefix.charAt(i);
            trienode curNode = prevNode.child.get(lastChar);
            if (curNode == null) {
                i++;
                break;
            }
            ArrayList<String> contactsWithPrefix = new ArrayList<>();
            displayContactsUtil(curNode, prefix, contactsWithPrefix);
            contactsWithPrefix.forEach(contact -> {
                System.out.println("Contact number of " + contact + " is: " + phonebook.get(contact));
            });
```

```
                prevNode = curNode;
            }
        }
    }

    class PhoneBook {
        Run | Debug
        public static void main(String args[]) {
            Directory directory = new Directory();
            directory.addContact(name: "Rahul", phoneno: 898485565);
            directory.addContact(name: "Ram", phoneno: 589542444);
            directory.addContact(name: "Ramgopal", phoneno: 76483423);
            directory.addContact(name: "don", phoneno: 988232323);
            directory.addContact(name: "Ramhari", phoneno: 849383432);
            directory.addContact(name: "dongli", phoneno: 978933323);
            directory.addContact(name: "donsenu", phoneno: 92348726);

            String query = "don";
            directory.Combinations(query);
        }
    }
```

Output:

we gave the starting query as “Ram”,

```
Contact number of Rahul is: 898485565
Contact number of Ram is: 134242444
Contact number of Ramhari is: 988232323
```

we gave the starting query as “don”,

```
Contact number of don is: 988232323
Contact number of dongli is: 978933323
Contact number of donsenu is: 92348726
```

Thank you

~ THE END