Unit-3

Page No.
Date

Data structure Transformations

que- Explain making structures Dynamic and making structures Persistent.

⇒

1) Making structure Dynamic :-

- When we say we're making a data structure dynamic, it means that the structure can change in size during program execution.
- In other words, elements can be added or removed as needed.
- This is in contrast to static data structures where the size is fixed at compile time.

Example :- consider a dynamic array. unlike a static array, which has a fixed size, a dynamic array can grow or shrink as elements are added or removed. This flexibility allows for efficient memory usage and adaptability to changing program requirements.

2) Making structure Presistent :-

- Making a data structure persistent refers to the ability to retain previous versions of the structure even after it has been modified.
- In other words, you can access or revert to previous states of the structure.

Example :- Persistent data structures are commonly used in functional programming. For instance, in a persistent linked list, when you modify the list, the original version of the list remains intact.

- This allows you to keep track of different versions of the data structure and revert to any previous version if needed, without affecting the current version.

Que- Explain the structure of tries how searching is possible for strings?

⟹

Trie :-

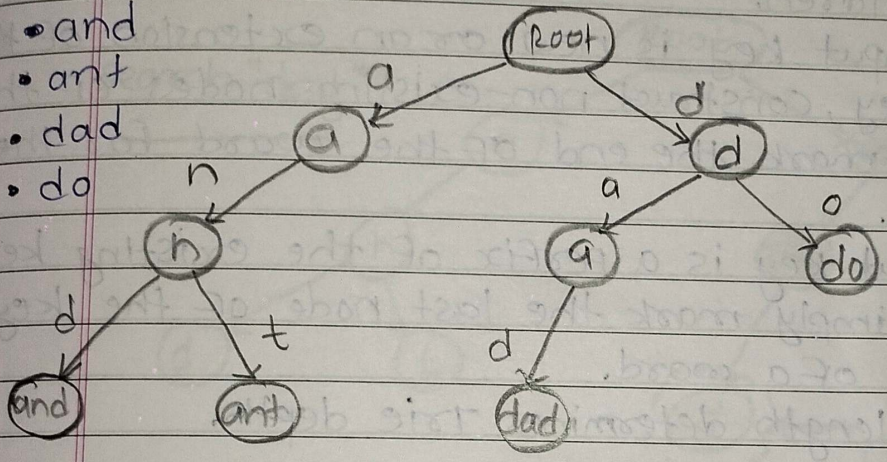
- Trie is a type of K-ary search tree used for storing and searching a specific key from a set. using Trie, search complexities can be brought to optimal limit (key length)

Definition :- A trie (derived from retrieval) is a multiway tree data structure used for storing strings over an alphabet. It is used to store a large amount of strings. The pattern matching can be done efficiently using tries.

- The trie shows words like allot, alone, ant, and are, bat, bad.
- The idea is that all strings sharing common prefix should come from a common node. The tries are used in spell checking programs.
- Preprocessing pattern improves the performance of pattern matching algorithm. But if a text is very large then it is better to preprocess text instead of pattern for efficient search.
- A trie is a data structure that supports pattern matching queries in time proportional to the pattern size.

- If we store keys in a binary search tree, a well balanced BST will need time proportional to $M * \log N$, where M is the maximum string length and N is the number of keys in the tree.
- Using Trie, the key can be searched in O(m).
- Trie is also known as digital tree or prefix tree.

- and
- ant
- dad
- do

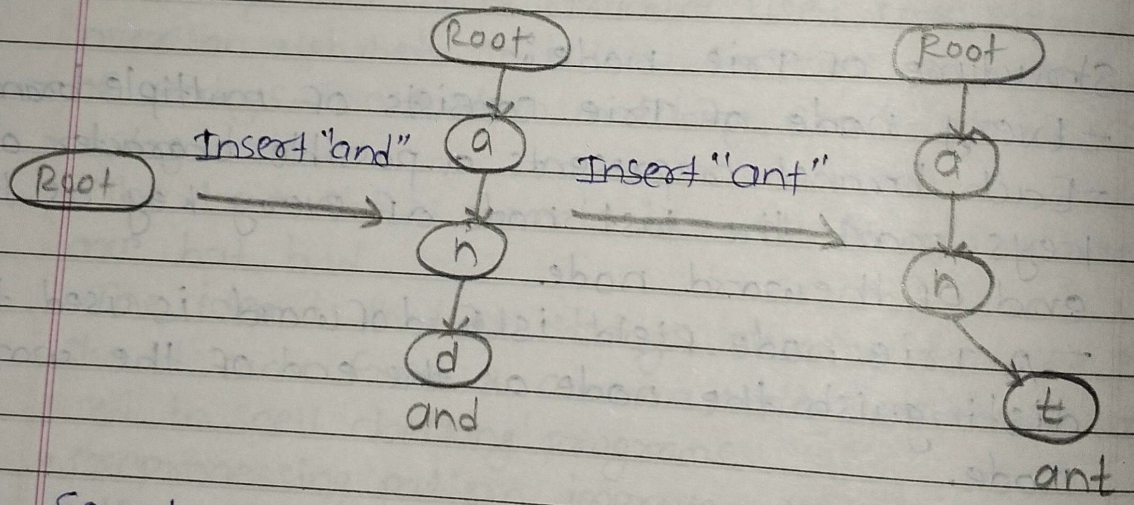


Structure of Trie node:-
- Every node of Trie consists of multiple branches.
- Each branch represents a possible character of keys. mark the last node of every key as the end of the word node.
- A Trie node field is EndofWord is used to distinguish the node as the end of the word node.

```
Struct TrieNode
 {
   Struct TrieNode *children[ALPHABET_SIZE];
    bool isEndofWord;
};
```

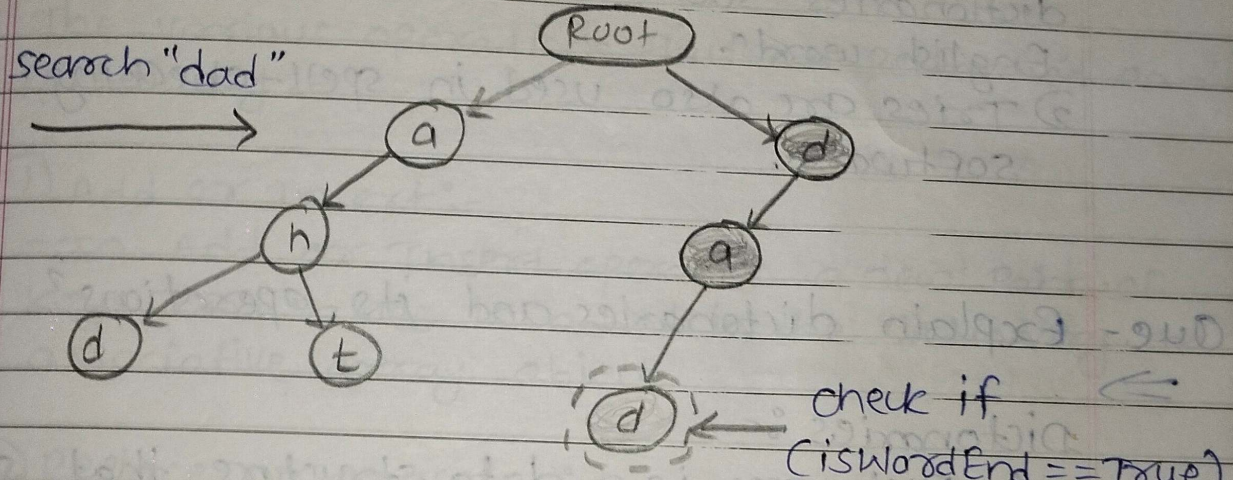
Date

- Insert operation in Trie :-
  - Inserting a key into Trie is a simple approach.
  - Every character of the input key is inserted as an individual Trie node. Note that the children is an array of pointers to next-level trie nodes.
  - The key character acts as an index to the array children.
  - If the input key is new or an extension of the existing key, construct non-existing nodes of the key, and mark the end of the word for the last node.
  - If the input key is a prefix of the existing key in Trie, simply mark the last node of the key as the end of a word.
  - The key length determines Trie depth.



- Search operation in Trie :-
  - searching for a key is similar to the insert operation.
  - However, It only compares the characters and moves down.
  - The search can terminate due to the end of a string or lack of key in the trie.

- In the former case, if the isEndofword Field of the last node is true, then the key exists in the trie.
- In the second case, the search terminates without examining all the characters of the key, since the key is not present in the trie.



Note :- Insert and search costs O(key_length), however, the memory requirements of Trie is O(ALPHABET_SIZE * key_length * N) where N is the number of keys in Trie.

complexity :-
Time complexity :-
1) Insertion - O(n)
2) Searching - O(n)

- Advantages of tries :-
1) In tries the keys are searched using common prefixes. Hence it is faster.
2) Tries take less space when they contain a large number of short strings. As nodes are shared between the keys.

5) Tries help with longest prefix matching. when we want to find the key.

•Applications of tries:-
1) Tries has an ability to insert, delete or search for the entries. Hence they are used in building dictionaries such as entries for telephone numbers, English words.
2) Tries are also used in spell-checking softwares.

Que- Explain dictonaries and its operations?
⇒

Dictonaries :-
-A Dictonary is a data structure that stores key-value pairs.
-Each key in the dictionary is unique and associated with exactly one value.

-Dictionaries can be implemented using various underlying data structures with hash tables being a common choice due to their efficient lookup time.

Features :-
1) Efficient Retrieval : Dictionaries Provide fast access to values based on their associated keys.
2) Hashing :- Dictionaries use hashing functions to map keys to indices within the underlying data structure, facilitating direct access to values.

-A dictionary or associative array is a general-purpose data structure that is used for the storage of a group of objects.
- Dictionary is used to store data in the key-value format.

- operations :-
The various operations that are performed on a Dictionary :-

1) Add or Insert :-
- In Add or Insert operation, a new pair of keys and values is added in the Dictionary or associative array object.

2) Replace or reassign :-
- In the Replace or reassign operation, the already existing value that is associated with a key is changed or modified.
- In other words, a new value is mapped to an already existing key.

3) Delete or remove :-
- In the Delete or remove operation, the already present element is unmapped from the Dictionary or associative array object.

4) Find or Lookup :-
- In the Find or Lookup operation, the value associated with a key is searched by passing the key as a search argument.

Page No.
Date

Que- Explain the structure and creation of suffix trie?

⇒

Suffix trie :-
- A suffix tree is a compressed trie data structure that efficiently represents all the suffixes of a given string.
- In other words, it's a tree-like data structure where each path from the root to a leaf node represents a suffix of the original string.

key features of
1) Efficient storage
2) Fast substring search.

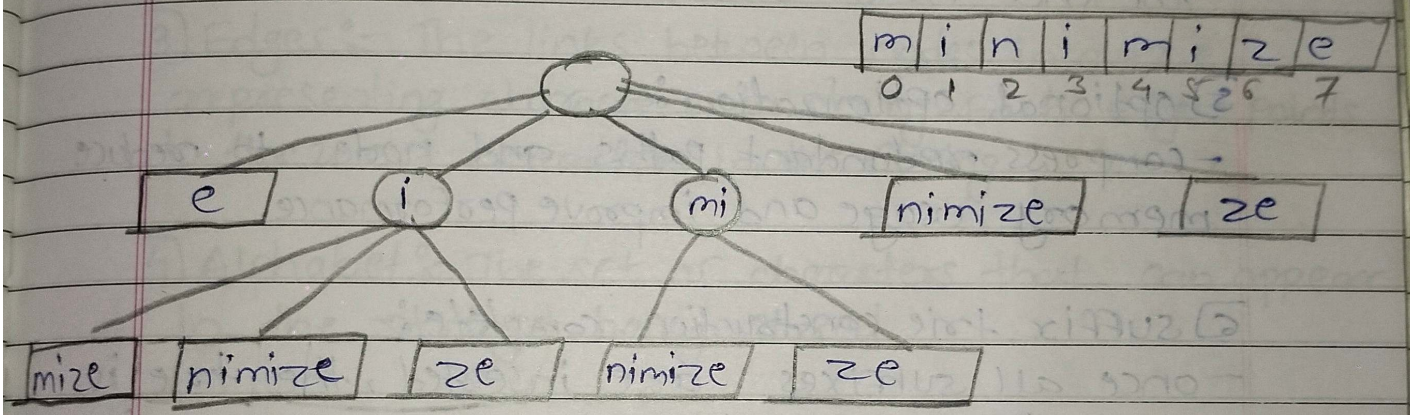
Structure :-
1) Node :-
- A node represents a substring of the original string.
- Each node contains the following components :-
- A starting index :- This denotes the position in the original string where the substring represented by the node begins.
- An ending index :- This indicates the position in the original string where the substring represented by the node ends.
- Pointers to child nodes :- Each node may have multiple child nodes, representing different suffixes that extend the current substring.

2) Edge :- An edge represents a character or a substring of characters between two nodes.

- The concatenation of characters along the path from the root to any node forms a substring of the original string.

3) Root Node :- The root node represents an empty string or the entire original string.

4) Leaf Node :- A Leaf node represents a suffix of the original string. It has no children.



Properties of Suffix tree :-
- The suffix tries for a text x of size n from an alphabet of size a.
- stores all the $n(n-1)/2$ suffixes of x in $O(n)$ space.
- Pattern matching.

creation of suffix trie :-

1) Start with the original string :-
- Take the string for which you want to create the suffix trie.

Date

2) Generate suffixes :-
- Create a list of all suffixes of the string

3) Insert suffixes into the trie :-
- Begin with an empty trie.
- Insert each suffix into the trie, creating nodes as needed.

4) mark end of suffixes :-
- use a special character or marker to denote the end of each suffix in the trie.

5) optional optimization :-
- compress redundant paths and nodes to reduce memory usage and improve performance.

6) suffix trie construction complete :-
- once all suffixes are inserted, the trie is ready for efficient substring search operations.

Que- What are the components of tries and explain it for organizing strings?

⇒

A trie, also known as a prefix tree, is a tree based data structure used for efficiently storing and retrieving a set of strings or sequences.

The main components of tries :-

1) Node :- Each node in a trie represents a single character of a string.

Page No.
Date

Nodes typically contain the following components :-
- Data :- The character associated with the node.
- Pointer(s) to child nodes :- These pointers represent the passible next characters in the string.
- The number of pointers is typically equal to the size of the alphabet, making tries efficient for alphabetic strings.

2) Root Node :- The top-level node of the trie, representing an empty string or null.

3) Edges :- The links between nodes in a trie, representing characters in string. Each edge points to the next character in the string.

4) Alphabet :- The set of characters that can appear in the strings being stored.

- Tries are organized in a way that makes it efficient to search for, insert and delete strings.
- By leveraging the structure of shared prefixes among strings, tries can reduce the time complexity of these operations compared to other data structure like hash tables or binary search tree.

Page No.
Date

Que Perform insertion, deletion and searching operations in dictionaries?

=> Operations:-

[a] Insertion:-

[i] Start with an Empty Trie:-

Begin with empty trie, represented by just the root node.

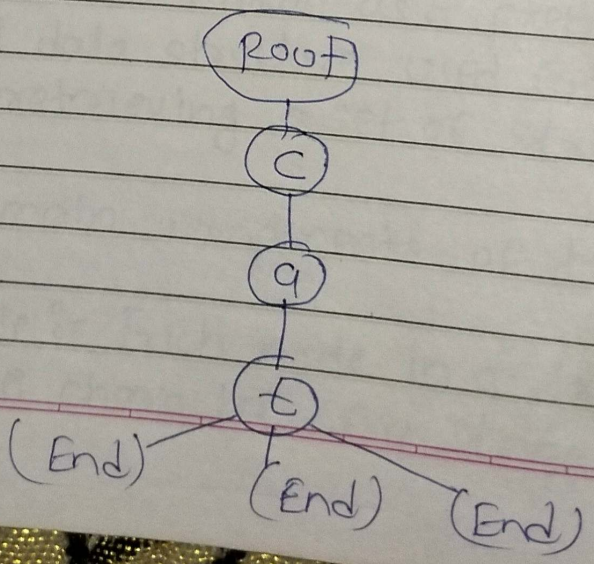
[ii] Insert a word:-

- To insert a word into the dictionary, traverse the trie based on each character of the word.
- If a node corresponding to a character does not exist, create a new node and connect to it the parent node.
- Repeat this process for each character of the word until the entire world is inserted.

[iii] Mark End of word:-

- once all characters of word are inserted, mark the last node as end of the word by setting or flag or value to indicate the presence of complete word.

Example:- Let's insert word "cat" into Trie.



Page No.
Date

b) Deletion :-

1] Locate the word :-

-To delete a word from dictionary, traverse trie to locate nodes corresponding to each character of word.

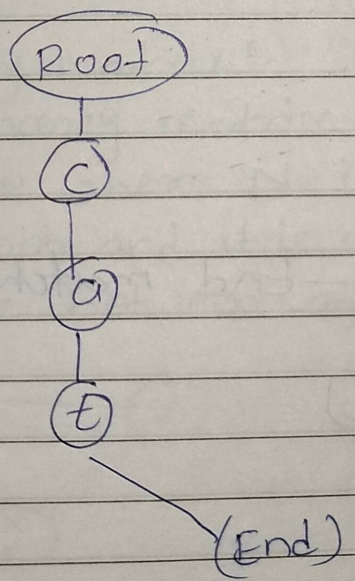
2] Mark End of word :-

-once the last character of word is reached remove flag or value indicating the end of word.

3] Remove Unnecessary Nodes :-

-If the word being deleted is only word that ends at particular node, remove unnecessary node.

Example :- Delete word "Cat" from the trie.



c) Searching :-

1] Start at the Root :-

-Begin search operation at root node of trie.

2) Traverse the trie:-
- For each character in target word, follow the corresponding edge in the trie.
- If an edge corresponding to character does not exist, the word is not present in dictionary and search operation terminates.

3) Check End of word:-
- once all characters of target word are traversed, check if last node reached marks the end of word.
- If it does, the word is present in dictionary otherwise it is not.

Example:- Search "cat".