Tries lesson 2

What is trie?

Trie is an efficient information re*trie*val data structure. It is a tree that stores strings. The maximum number of children of a node is equal to the size of the alphabet. Trie supports search, insert and delete operations in O(L) time where L is the length of the key.

Why do we need trie data structure?

1. With Trie, we can insert and find strings in *O(L)* time where *L* represents 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 .
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 with tries.

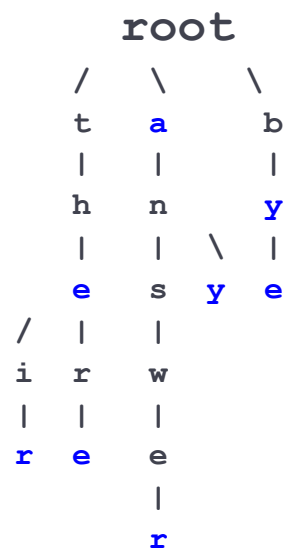
Structure of Trie Node:

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.

A simple structure to represent nodes of the English alphabet can be as following,

```
// Trie node
struct TrieNode
{
    struct TrieNode *children[ALPHABET_SIZE];
    // isEndOfWord is true if the node represents end of a word
    bool isEndOfWord;
};
```

Example:



Operations in tries:

1. Insert
2. Search
3. Delete

Insertion & searching in tries:

```
#include <bits/stdc++.h>
using namespace std;
#define ALPHABET_SIZE (26)
struct TrieNode {
    struct TrieNode *children[ALPHABET_SIZE];
// isLeaf is true if the node represents
    // end of a word
    bool isLeaf;
};
struct TrieNode *getNode(void) {
    struct TrieNode *pNode = NULL;
    pNode = (struct TrieNode *) malloc(sizeof(struct TrieNode));
    if (pNode) {
        int i;
        pNode->isLeaf = false;
        for (i = 0; i < ALPHABET_SIZE; i++)
                pNode->children[i] = NULL;
    }
    return pNode;}
```

```
20.     void insert(struct TrieNode *, string);
21. bool search(struct TrieNode *, string);
22.    // Driver
23.    int main() {
24.       int t;
25.       cin >> t;
26.       while (t--) {
27.          // Input keys (use only 'a' through 'z' and lower case)
28.          int n;
29.          cin >> n;
30.          string keys[n];
31.       for (int i = 0; i < n; i++) {
32.             cin >> keys[i];
33.          }
34.          struct TrieNode *root = getNode();
35.          // Construct trie
36.          for (int i = 0; i < n; i++) insert(root, keys[i]);
37.           string abc;
38.          cin >> abc;
39.          if (search(root, abc))
40.             cout << "1\n";
41.       else
42.             cout << "0\n";
43.       }
44.       return 0;
45.    }
```

```
46.  void insert(struct TrieNode *root, string key) {
47.      // code here
48.      struct TrieNode *a=root;
49.      for(int i=0;i<key.length();i++)
50.      {
51.      int index=key[i]-'a';
52.          if(a->children[index]==NULL)
53.          {
54.              a->children[index]=getNode();
55.          }
56.              a=a->children[index];
57.      }
58.      a->isLeaf=true;
59.  }
60.
61.bool search(struct TrieNode *root, string key) {
62.      // code here
63.      struct TrieNode *a=root;
64.      for(int i=0;i<key.length();i++)
65.      {
66.          int index=key[i]-'a';
67.          if(a->children[index]==NULL)
68.          {
69.              return false;
70.          }
71.          a=a->children[index];
```

```
72.      }
73.      if(a->isLeaf)
74.      {
75.          return true;
76.      }
77.   }
```

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

Ques link:
https://practice.geeksforgeeks.org/problems/trie-insert-and-search0651/1/?category[]=Trie&category[]=Trie&page=1&query=category[]Triepage1category[]Trie

Deletion in trie:

Trie is an efficient information retrieval data structure. This data structure is used to store Strings and search strings, String stored can also be deleted. Given a Trie root for a larger string super and a string key, delete all the occurences of key in the Trie.

Example 1:

Input:
N = 8
super = "the a there answer any by bye their"

key = "the"

Algorithm:

1. Key may not be there in trie. Delete operation should not modify trie.
2. Key present as unique key (no part of key contains another key (prefix), nor the key itself is prefix of another key in trie). Delete all the nodes.
3. Key is the prefix key of another long key in trie. Unmark the leaf node.
4. Key present in trie, having atleast one other key as prefix key. Delete nodes from end of key until first leaf node of longest prefix key.

CODE:

```
1. bool isEmpty(trie_node_t * root)
2. {
3.     for (int i = 0; i <ALPHABET_SIZE; i++)
4.         if (root->children[i])
5.             return false;
6.     return true;
```

```
7. }
8.
9. trie_node_t* deletenode (trie_node_t *root, string s,int depth)
10.{
11.   if(!root)
12.      return NULL;
13.
14.    if(depth == s.length())
15.    {
16.        if(root->value==1)
17.        {
18.            root->value=0;
19.        }
20.            if(isEmpty(root))
21.        {
22.             delete(root);
23.             root=NULL;
24.            }
25.            return root;
26.         }
27.        int index=s[depth]-'a';
28.
    root->children[index]=deletenode(root->children[index],s,depth+1);
29.        if (isEmpty(root) && root->isEndOfWord == false) {
30.            delete (root);
31.            root = NULL;
```

```
32.   }
33.         return root;
34.   }
35.
36.   void deleteKey(trie_node_t *root, char key[])
37.   {
38.      int len=strlen(key);
39.      string s;
40.      for(int i=0;i<len;i++)
41.  {
42.          s+=key[i];
43.      }
44.    root= deletenode(root,s,0);
45.   }
```

Time complexity for deletion of a string is O(N) where N is the length of string & space complexity is O(26*key_length*No_of_key),where key_length is the length of longest string ,no_of_key is no strings.

Ques link: https://practice.geeksforgeeks.org/problems/trie-delete/1

Longest prefix matching using tries:

Given a dictionary of words and an input string, find the longest prefix of the string which is also a word in the dictionary .

Examples:

Let the dictionary contains the following words:
{are, area, base ,cat, cater, children, basement}

Below are some input/output examples:

Input String	Output
caterer	cater
basemexy	base
child	<empty>

Algorithm :

We build a Trie of all dictionary words. Once the Trie is built, traverse through it using characters of the input string. If prefix matches a dictionary word, store current length and look for a longer match. Finally, return the longest match.

CODE:

```
1. #include <bits/stdc++.h>
2. #include <iostream>
3. using namespace std;
4. struct node{
```

```
5.    struct node* children[26];
6.     bool eow;
7.     int result;
8.  };
9.
10.struct node* getnode(void)
11.{
12.    node *p=new node;
13.    for(int i=0;i<26;i++)
14.    {
15.        p->children[i]=NULL;
16.    }
17.    p->eow=false;
18.    return p;
19.}
20.     void insert(struct node *root,string key,int j)
21.{
22.        struct node*p=root;
23.        for(int i=0;i<key.length();i++)
24.        {
25.            int index=key[i]-'a';
26.            if(!p->children[index])
27.                p->children[index]=getnode();
28.            p=p->children[index];
29.        }
30.        p->eow=true;
```

```
31.     p->result=j;
32.    }
33.    int search(struct node *root,string s)
34.    {   if(root==NULL)
35.         return 0;
36.        struct node *p=root;
37.        for(int i=0;i<s.length();i++)
38.        {
39.            int index=s[i]-'a';
40.            if(p->children[index])
41.                  p=p->children[index];
42.            else
43.                return 0;
44.        }
45.        return p->result;
46.    }
47.    int main()
48.    { string arr[]={"hello","are","area","base","basement","xyz"};
49.        int n=sizeof(arr)/sizeof(arr[0]);
50.        node *root=getnode();
51.  for(int i=0;i<n;i++)
52.        {
53.            insert(root,arr[i],i);
54.        }
55.        string a;
56.        cin>>a;
```

```
57.       string s="";
58.        int res=-1;
59.       for(int i=0;i<a.length();i++)
60.       {   s.push_back(a[i]);
61.      if(search(root,s))
62.          { res=search(root,s);
63.          }
64.       }
65.       if(res==-1)
66.          cout<<"-1"<<endl;
67.       else
68.         cout<< arr[res];
69.        return 0;
70.    }
```

Code link:https://sapphireengine.com/@/uq8la1