Tries

Data structure

Lesson 3

Prefix match with other strings

Given an array of strings **arr[]** of size **n** and given a string **str** and an integer **k**. The task is to find the count of strings in **arr[]** whose prefix of length **k** matches with the **k** length prefix of **str**.

Input:
n = 6
arr[] = {"abba", "abbb", "abbc", "abbd", "abaa", "abca"}
str = "abbg"
k = 3
Output: 4
Explanation:
"abba", "abbb", "abbc" and "abbd" are the matching strings.

APPROACH 1(naive approach):

In O(N*M), simply we traverse the array and check the string whether prefix of the string is same as given string ,if yes increase the count,else continue.

APPROACH 2(USING TRIES):

IN O(N) complexity

We will form a trie and insert all the strings in the trie and we will create another variable (frequency) for each node which will store the frequency of prefix of the given strings. Now to get the count of strings whose prefix matches with the given string to a given length k we will have to traverse the

trie to the length k from the root, the frequency of the Node will give the count of such strings.

CODE:

```
struct node{
    int frequency;
    node* children[26];
};
struct node* getnode(void)
{
    node *p=new node();
    for(int i=0;i<26;i++)
    {
        p->children[i]=NULL;
    }
void *insert(struct node *root, string s)
{
    node *p=root;
    for(int i=0;i<s.length();i++)
    {
        int index=s[i]-'a';
        if(!p->children[index])
        {
            p->children[index]=getnode();
        }
        p->children[index]->frequency++;
        p=p->children[index];
    }
```

```
25.   return root;
26.}
27.int find(struct node *root,string s,int k)
28.{
29.   node *p=root;
30.   int count=0;
31.   for(int i=0;i<s.length();i++)
32.   {
33.       int index=s[i]-'a';
34.       if(p->children[index]==NULL)
35.           return 0;
36.      p=p->children[index];
37.      count++;
38.      if(count==k)
39.      {
40.          return p->frequency;
41.      }
42.   }
43.   return 0;
44.}
45.class Solution{
46.public:
47.   int klengthpref(string arr[], int n, int k, string str){
48.       struct node* root=getnode();
49.       for(int i=0;i<n;i++)
50.       {
51.           insert(root,arr[i]);
52.       }
```

```
53.      int count=find(root,str,k);
54.      return count;
55.   }
56.};
```

Quelink:https://practice.geeksforgeeks.org/problems/prefix-match-with-other-strings/1/?category[]=Trie&category[]=Trie&problemStatus=solved&page=1&query=category[]TrieproblemStatussolvedpage1category[]Trie

Unique rows in boolean matrix

Given a binary matrix your task is to find all unique rows of the given matrix.

Example 1:

Input:
row = 3, col = 4
M[][] = {{1 1 0 1},{1 0 0 1},{1 1 0 1}}
Output: 1 1 0 1 $1 0 0 1 $
Explanation: Above the matrix of size 3x4
looks like
1 1 0 1
1 0 0 1
1 1 0 1
The two unique rows are 1 1 0 1 and 1 0 0 1 .

ALGORITHM:

Naive approach:

Traverse the matrix row-wise, For each row check if there is any similar row less than the current index.
If any two rows are similar then do not print the row.
Else print the row.

Using tries:

Create a Trie where rows can be stored.
Traverse through the matrix and insert the row into the Trie.
Trie cannot store duplicate entries so the duplicates will be removed
Traverse the Trie and print the rows.

Algorithm:

1. Create a Trie where rows can be stored.
2. Traverse through the matrix and insert the row into the Trie.
3. Trie cannot store duplicate entries so the duplicates will be removed
4. Traverse the Trie and print the rows.

Code:

```
1. struct trieNode
2. {
3.    trieNode *child[2];
```

```
4.    bool leaf;
5.    trieNode(){
6.            child[0]=NULL;
7.            child[1]=NULL;
8.            leaf=false;
9.    }
10.};
11. bool insert(trieNode *root ,vector<int> arr,int col){
12.   trieNode *curr=root;
13.   for(int i=0;i<col;i++){
14.       if(curr->child[arr[i]]==NULL){
15.           curr->child[arr[i]]=new trieNode();
16.       }
17.       curr=curr->child[arr[i]];
18.   }
19.   if(curr->leaf)
20.       return false;
21.   else{
22.           curr->leaf=true;
23.       return true ;
24.       }
25.    }
26.    vector<vector<int>> uniqueRow(int M[MAX][MAX],int row,int col)
27.    {
28.       trieNode *head=new trieNode;
```

```
29.      vector<vector<int>>ans;
30.      for(int i=0;i<row;i++){
31.      // int arr[col];
32.          vector<int>arr;
33.          for(int j=0;j<col;j++){
34.              arr.push_back(M[i][j]);
35.          }
36.          if(insert(head,arr,col)){
37.              // for(int j=0;j<col;j++)
38.              ans.push_back(arr);
39.          }
40.      }
41.  return ans;
42.    }
43.   }
```

Que_link:

https://practice.geeksforgeeks.org/problems/unique-rows-in-boolean-matrix/1/?category[]=Trie&category[]=Trie&page=1&query=category[]Triepage1category[]Trie

CamelCase pattern matching

Given a dictionary of words where each word follows CamelCase notation, print all words in the dictionary that match with a given pattern consisting of uppercase characters only.

CamelCase is the practice of writing compound words or phrases such that each word or abbreviation begins with a capital letter. Common examples include: "PowerPoint" and "WikiPedia", "GeeksForGeeks", "CodeBlocks", etc.

Example:

Input:

2

8

Hi Hello HelloWorld HiTech HiGeek HiTechWorld HiTechCity HiTechLab

HA

3

WelcomeGeek WelcomeToGeeksForGeeks GeeksForGeeks

WTG

Output:

No match found

WelcomeToGeeksForGeeks

ALGORITHM: The idea is to insert all dictionary keys into the Trie one by one. Here key refers to only Uppercase characters in original word in CamelCase notation. If we encounter the key for the first time, we need to mark the last node as leaf node and insert the complete word for that key into the vector associated with the leaf node. If we encounter a key that is already in the trie, we update the vector associated with the leaf node with current word. After all dictionary words are processed, we search for the pattern in the trie and print all words that matches the pattern.

CODE:

```
1.  struct node{
2.      struct node *children[26];
3.      bool eow;
4.      vector<string>v;
5.  };
6.
7.  struct node* getnode(void)
8.  {
9.      node *p=new node;
10.     for(int i=0;i<26;i++)
11.     {
12.         p->children[i]=NULL;
13.     }
14.     p->eow=false;
15.     return p;
16. }
17.
18. void insert(struct node *root,string s)
19. {
20.         node *p=root;
21.     for(int i=0;i<s.length();i++)
22.         {
23.             if(islower(s[i]))
24.                 continue;
25.           int index=s[i]-'A';
26.           if(!p->children[index])
```

```
27.         {
28.             p->children[index]=getnode();
29.         }
30.         p=p->children[index];
31.     }
32.         p->eow=true;
33.         (p->v).push_back(s);
34.     }
35.     void printword(struct node *p)
36.     {
37.         if(p->eow)
38.         {
39.             for(int j=0;j<((p->v).size());j++)
40.             {
41.             cout<<p->v[j]<<" ";
42.             }
43.         }
44.         for(int i=0;i<26;i++)
45.         {
46.             node *child=p->children[i];
47.             if(child)
48.             {
49.                 printword(child);
50.             }
51.     }
52.     }
53.     bool search(struct node *root,string s)
```

```
54.    {    node *p=root;
55.       for(int i=0;i<s.length();i++)
56.       {
57.          int index=s[i]-'A';
58.          if(!p->children[index])
59.            return false;
60.          p=p->children[index];
61.    }
62.       printword(p);
63.       return true;
64.    }
65.    void findallword(string arr[],int n,string s)
66.    {
67.       node *root=getnode();
68.       for(int i=0;i<n;i++)
69.       {
70.          insert(root,arr[i]);
71.       }
72.       if(!search(root,s))
73.         cout<<"No match found";
74.    }
75.    int main() {
76.       int t;
77.       cin>>t;
78.       while(t--)
79.       {
80.          int n;
```

```
        cin>>n;
        string arr[n],s;
        for(int i=0;i<n;i++)
        {
            cin>>arr[i];
        }
        cin>>s;
        findallword(arr,n,s);
        cout<<endl;
    }
    return 0;
}
```

Que link:

https://practice.geeksforgeeks.org/problems/camelcase-pattern-matching/0/?category[]=Trie&category[]=Trie&problemStatus=unsolved&page=1&query=category[]TrieproblemStatusunsolvedpage1category[]Trie#

SHORTEST UNIQUE PREFIX FOR EVERY WORD

Given an array of words, find all shortest unique prefixes to represent each word in the given array. Assume that no word is prefix of another.

Example 1:

Input:

arr[] = {"zebra", "dog", "duck", "dove"}

Output: z dog du dov

Explanation:

Zebra:z, dog:dog, duck:du, dove:dov

A Simple Solution is to consider every prefix of every word (starting from the shortest to largest), and if a prefix is not prefix of any other string, then print it.

APPROACH using tries:

Time Complexity: O(N*length of each word)

Auxiliary Space: O(N*length of each word)

Construct a trie of all words. Also maintain frequency of every node (Here frequency is number of times node is visited during insertion). Time complexity of this step is O(N) where N is the total number of characters in all words.

Now, for every word, we find the character nearest to the root with frequency as 1. The prefix of the word is path from root to this character. To do this, we can traverse Trie starting from root. For every node being traversed, we check its frequency. If frequency is one, we print all characters from root to this node and don't traverse down this node.

Time complexity if this step also is O(N) where N is the total number of characters in all words.

CODE:

```
1. struct node{
```

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

```
27.         p->frequency++;
28.     }
29.     p->eow=true;
30.   }
31.
32.   string search(struct node *root,string key)
33.   {
34.     string res;
35.     node *p=root;
36.     for(int i=0;i<key.length();i++)
37.     {
38.         res.push_back(key[i]);
39.         p=p->children[key[i]-'a'];
40.         if(p->frequency==1)
41.         break;
42.     }
43.     return res;
44.   }
45.
46.   class Solution
47.   {
48.     public:
49.     vector<string> findPrefixes(string arr[], int n)
50.     {
51.      struct node *root=getnode();
```

```
        for(int i=0;i<n;i++)
        {
            insert(root,arr[i]);
        }
        vector<string>v;
        for(int i=0;i<n;i++)
        {
            v.push_back(search(root,arr[i]));
        }
     return v;
      }
   };
```

Que_link: https://practice.geeksforgeeks.org/problems/shortest-unique-prefix-for-every-word/1/?category[]=Trie&category[]=Trie&page=1&query=category[]Triepage1category[]Tr ie

Count of distinct substrings

Given a string of length N of lowercase alphabet characters. The task is to complete the function countDistinctSubstring(), which returns the count of total number of distinct substrings of this string.

Input:

2

ab

ababa

Output:

4

10

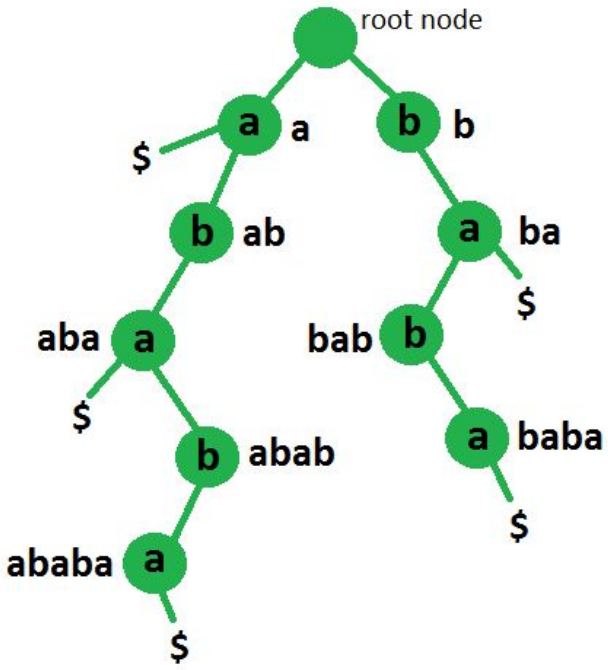
Exaplanation:

Testcase 1: For the given string "ab" the total distinct substrings are: "", "a", "b", "ab".

ALGORITHM:

Using tries:

- Each root to node path of a trie represents a prefix of words present in Trie. Here we words are suffixes. So each node represents a prefix of suffixes.
- Every substring of a string “str” is a prefix of a suffix of “str”.



Trie for string ababa with corresponding substring for each node

CODE:

```
1. struct node
2. {
3.  struct node *children[26];
4.  bool eow;
5. };
6.
7. struct node *getnode(void)
8. {
9.    node *p=new node;
```

```
10.   for(int i=0;i<26;i++)
11.   {
12.   p->children[i]=NULL;
13.   }
14.   p->eow=false;
15.return p;
16.}
17.void insert(struct node*root,string s)
18.{
19.  node *p=root;
20.      for(int i=0;i<s.length();i++)
21.  {
22.          int index=s[i]-'a';
23.          if(!p->children[index])
24.          {
25.              p->children[index]=getnode();
26.          }
27.          p=p->children[index];
28.      }
29.      p->eow=true;
30.    }
31.int count(struct node *root)
32.    {
33.       if(root==NULL)
34.         return 0;
35.         int countnode=0;
```

```
36.      for(int i=0;i<26;i++)
37.      {
38.         if(root->children[i])
39.           {
40.               countnode+=count(root->children[i]);
41.        }
42.      }
43.      return countnode+1;
44.   }
45.   int construct (struct node *root,string s)
46.   {
47.    for(int i=0;i<s.length();i++)
48.    {   string a="";
49.        for(int j=i;j<s.length();j++)
50.        {
51.        a.push_back(s[j]);
52.        }
53.        insert(root,a);
54.    }
55.    int b=count(root);
56.    return b;
57.   }
58.   int countDistinctSubstring(string s)
59.   {  node *root=getnode();
60.     int a=construct(root,s);
61.  return a;
```

```
62.   }
```

Que link:https://practice.geeksforgeeks.org/problems/count-of-distinct-substrings/1/?category[]=Trie&category[]=Trie&problemStatus=solved&page=1&query=category[]TrieproblemStatussolvedpage1category[]Trie

Problems for practice:

Geeks and strings: https://practice.geeksforgeeks.org/problems/geek-and-strings3030/1

Duplicate rows in boolean strings: https://practice.geeksforgeeks.org/problems/find-duplicate-rows-in-a-binary-matrix/1/?category[]=Trie&category[]=Trie&page=1&query=category[]Triepage1category[]Trie