String Algorithms and Trie

Youtube link :

https://youtu.be/Fn9u8CMTVEw

Contents:

String Matching Algorithms

Q. Given a string S and a pattern P, find all the occurrences of P in S.

Brute Force / Naive Approach

S: a b a b c a b c a b a b a b d
P: a b a b d

✔✔✔✔ x

S: a b a b c a b c a b a b a b d
P: a b a b d

x

S: a b a b c a b c a b a b a b d
P: a b a b d

✔✔x

S: a b a b c a b c a b a b a b d
P: a b a b d
x
We are sliding pattern by 1 step right, if a mismatch occurs.
Visualise at:
http://whocouldthat.be/visualizing-string-matching/

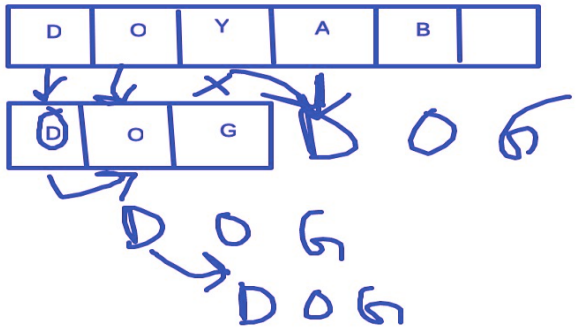
s.length(); // gives length of string s

```
// m = length of given string S
// n = length of given pattern P

for(int i=0; i<=m-n; i++)
{
    for(int j=0; j<m; j++)
    {
    if(S[i]!=P[j])
      break;
  }
if(j==m)
{
  cout<<"Pattern found at "<<i;
}
}
```

Time Complexity: O(n * m)

Intuition for KMP



In the example on left, we can shift the pattern by 2 steps, instead of shifting by 1 (since all characters of pattern are distinct)

```
   0 1 2  3  4 5 6
S : D E A D E L E P Q S R
P : D E A D E Y E
   0 1  2  3  4 5 6
   ✔ ✔  ✔  ✔  ✔ x
```

We can move i to 5 and j to 2 and repeat this process, in above example. (Instead by shifting the pattern only by 1 step)

Intuition: In pattern, find where the prefix of pattern is repeating.

LPS Array (Longest Prefix Suffix Array):

(also called "Pi array" or "Failure Function")

LPS[i] = Longest proper prefix of the string that is also a suffix of string[0….i]

Poper prefix: Prefix of a string which is not the same as the given string. Prefix means any substring from the starting.
Suppose , s="ababcdef"
Proper Prefixes of s are a, ab, aba, ababc, ababcd, etc.
But ababcdef is not a proper prefix.

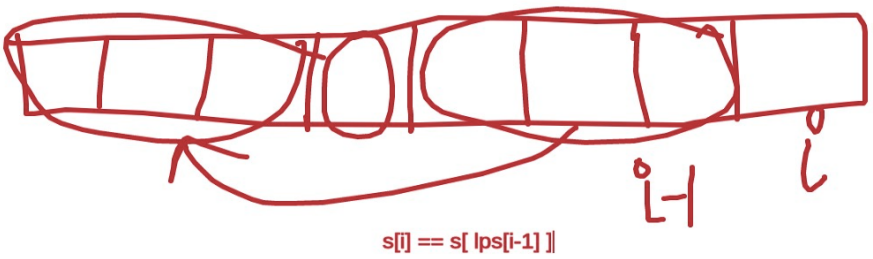
Suffix: Any substring from the end of the string
Suppose , s="ababcdef"
Suffixes of s are def, abcdef, babcdef, ef, f

i	0	1	2	3	4	5	6	7	8	9
	a	b	c	d	a	b	e	a	b	f
lps[i]:	0	0	0	0	1	2	0	1	2	0

i	0	1	2	3	4	5	6	7	8	9
	a	a	b	c	a	d	a	a	b	e
lps[i]:	0	1	0	0	1	0	1	2	3	0

Note: lps[i] can **increase** by a maximum of 1.

Finding the LPS array



Idea: 1. lps[0]=0; // Since, there is no proper prefix of s[0]
2. if(s[i] == s[lps[i-1]]) lps[i]=lps[i-1]+1;

```
int n=s.length();
vector<int> lps(n);
lps[0]=0;
for(int i=1; i<n; i++)
{
    int j=lps[i-1];
    while(j>0 && s[i]!=s[j] )
   {
      j=lps[ j-1 ];
   }

    if(s[i] == s[j])
    {
        j++;
    }
   lps[i]=j;
}
```

Time complexity of finding LPS array: O(n)

Q. Given a string S and a pattern P, find all the occurrences of P in S.

KMP Algorithm

1. Find LPS array of pattern p
2. Just similar to naive algorithm,
 let i is index in string s
 let j is index in pattern p
 if(s[i] == p[j])

```
{
   i++;
   j++;
}
else
{
  if(j>0)
        j = lps[j-1]
  else
       i++
}
```

i=0
j=0
S: a b a b c a b c a b a b a b d
P: **a** b **a** b d
lps: 0 0 1 2 0
T T T T

i=4
j=2

S: a b a b c a b c a b a b a b d
P: **a** b **a** b d
lps: 0 0 1 2 0

i=4
j=0
Since, j=0, we can't decrease j further.
So, increase i by 1

i=5
j=0
0 1 2 3 4 5
S: a b a b c a b c a b a b a b d
P: **a** b **a** b d
lps: 0 0 1 2 0

i=6
j=0

Visualise at:
https://algorithm-visualizer.org/dynamic-programming/knuth-morris-pratts-string-search

Code:

```
int n=p.length();
vector<int> lps(n);
lps[0]=0;
```

```
for(int i=1; i<n; i++)
{
    int j=lps[i-1];
    while(j>0 && p[i]!=p[j] )
   {
      j=lps[ j-1 ];
   }

    if(p[i] == p[j])
    {
        j++;
    }
   lps[i]=j;
}

int m=s.length();

int i=0,j=0;

while(i<m)
{

  if(s[i]==p[j])
  {
    i++;
    j++;
```

```
    if(j==n)
    {
    cout<<"Pattern found at "<<i-j; // 1-based index
    }
  }
else if(i<m && s[i] != p[j] ) {
if(j>0)
    j=lps[j-1];
else
   i++;
}

}
```

Time complexity: O(m + n)
Building lps array of pattern takes O(n)
and search will take O(m)

Time Complexity of KMP Intuition

Consider the worst case scenario, which would be when the pattern appears the maximum number of times, in the string

0 1 2 3 4 5 6 7 8 9 10
S: a a a a a a a a a a a
P: a a a a a
lps: 0 1 2 3 4

Taking input when number of testcases is not given

When no. of testcases is not given, use cin>> in while loop.
(Eg. in problem SPOJ NHAY)
Reference:
https://www.geeksforgeeks.org/using-return-value-cin-take-unknown-number-inputs-c/

```
int n;
while(cin>>n)
{
string p;
cin>>p;
string s;
cin>>s;
//....
}
```

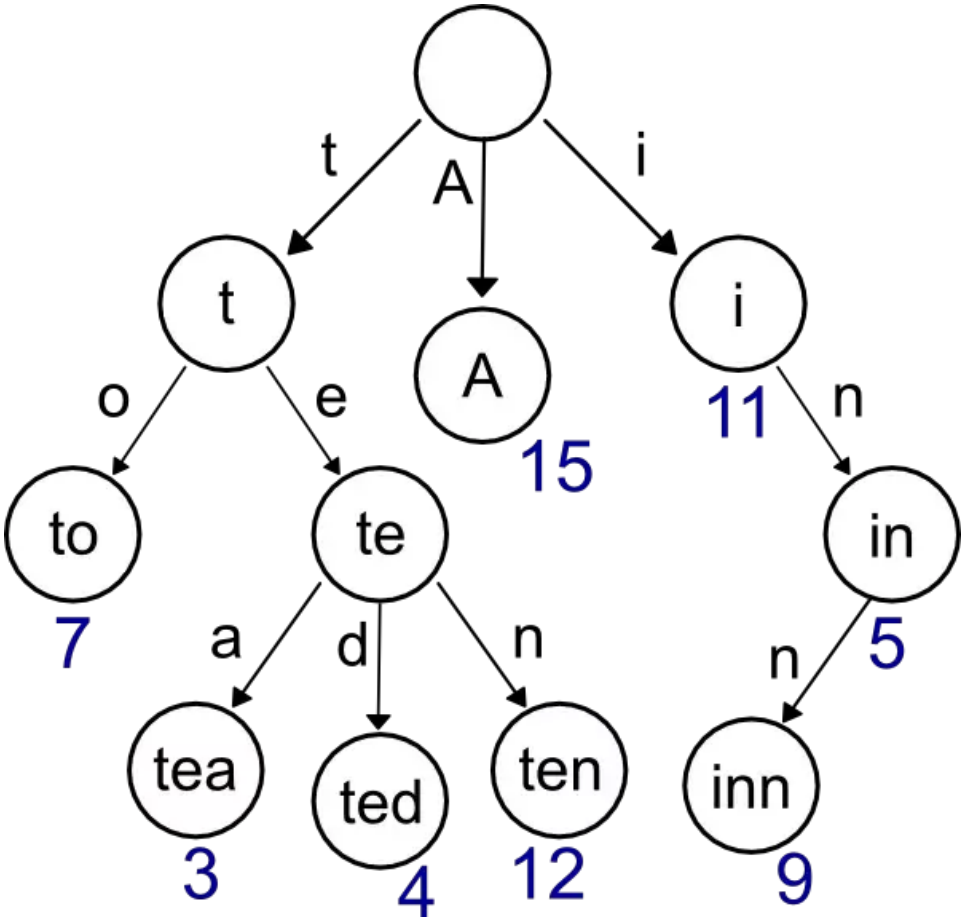
Practice Problems

1. https://www.spoj.com/problems/NHAY/
2. https://www.spoj.com/problems/PERIOD/
3. https://www.codechef.com/problems/BORDER
4. https://www.spoj.com/problems/EDIST/

Don't use in-built functions for string matching, they use naive algorithm. Always use KMP algorithm.

Tries

A trie is a tree-like data structure that stores strings. **Each node is a string, and each edge is a character.** The root is the empty string, and every node is represented by the characters along the path from the root to that node. **This means that every prefix of a string is an ancestor of that string's node.**



26+26*26+26*26*26+...26^l

input="tea"
input="ted"

find "tex"

te->freq?
int a=10;

```
struct node
{
 int freq;
 node* child[26];//array of pointers to children
  node()//constructor
  {
   freq=0;
   for(int i=0;i<26;i++)
    child[i]=NULL;
  }
 };



node* root=new node(); //pointer to a new node
node* tmp;
tmp->freq
while(n--)
```

```
{
  string s;
  cin>>s;
  tmp=root;

  for(int i=0;i<s.size();i++)
  {
   if(tmp->child[(s[i]-'a')]==NULL)
  {
     node* z=new node();//child create
     tmp->child[(s[i]-'a')]=z;
  }
  tmp=tmp->child[(s[i]-'a')];
  tmp->freq++;
}

tmp=root;
s->input..
int res=0;
for(int i=0;i<s.size();i++)
{
  if(tmp->child[(s[i]-'a')]==NULL)
   break;
 tmp=tmp->child[(s[i]-'a')]
 if(i==(s.size()-1))
  res=tmp->freq;
}
cout<< res;
```

```
}
```

Try this problem:
https://www.hackerrank.com/challenges/contacts/problem
(If you are stuck, look at my submission, given at last of this doc)
Note: Instead of making child as an array of size 26, you can also make it, an unordered_map<int,node*> to save space.

Space Complexity of Trie:

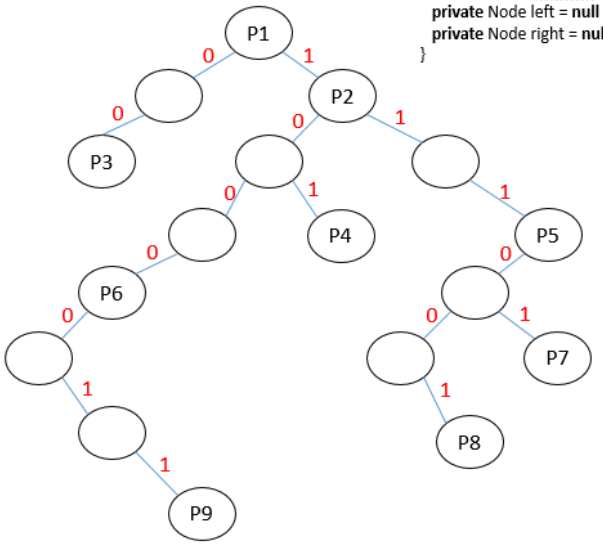
Just see how many nodes are possible in the trie. And multiply that with the size of 1 node.
Like in the above problem, **if you sum up the number of characters of all the strings**, there would be n*21 total characters. So, in the worst case, we need n*21 Trie nodes. And if size of 1 node is 26, space complexity is approximately: O(n*21*26) ≈ 10^7

Bit manipulation problems using Trie

7->111
5->101
4->100
s[0]-->1<<30

Prefix database

P1	*
P2	1*
P3	00*
P4	101*
P5	111*
P6	1000*
P7	11101*
P8	111001*
P9	1000011*




```
private static class Node {
  private String nextHop = null;
  private Node left = null ;
  private Node right = null ;
}
```

//If a prefix node
//Left node with an address bit 0
//Right node with an address bit 1

110
111

Q)n->length array

2<=n<=10^5

1<=arr[i]<=10^9

i,j(i!=j)

arr[i] xor arr[j]-->min

Link:

https://practice.geeksforgeeks.org/problems/minimum-xor-value-pair/0/

(Look at my submission at the end of the doc, only if you are stuck)

log2(arr[i])+1->111

log2(10^2)==32

O(32*n)-->time trie construction..

arr[i]=1010101000

arr[j]=1001100011

arr[k]=0111111111

ans1=2^8+...
ans2=2^7+...
2^3
2^2+2^1+2^0
arr[i]->string convert(s)
for(int j=30;j>=0;j--)
 if((1<<j)&arr[i])
  s+='1';
else
 s+='0'

1101011
2^x

Also try these problems:
1. https://www.hackerrank.com/challenges/no-prefix-set/problem
2. https://www.hackerrank.com/challenges/maximum-xor/problem

My submission for Geeks Minimum XOR value:
https://csacademy.com/code/tkmdyzSq/

My submission for Hackerrank Contacts:
(Don't look, if you have not tried by yourself)
https://csacademy.com/code/yZgfTzc9/

My submission for Hackerrank No Prefix Sets:
(Don't look, if you have not tried by yourself)
https://csacademy.com/code/6buXyAYv/

My submission for Hackerrank No Prefix Sets:
(Don't look, if you have not tried by yourself)
https://csacademy.com/code/AHZcCYhO/