GeeksforGeeks
A computer science portal for geeks

IDE   Q&A   GeeksQuiz

Pattern Searching | Set 8 (Suffix Tree Introduction)

Given a text txt[0..n-1] and a pattern pat[0..m-1], write a function search(char pat[], char txt[]) that prints all occurrences of pat[] in txt[]. You may assume that n > m.

Preprocess Pattern or Preoprocess Text?

We have discussed the following algorithms in the previous posts:

KMP Algorithm

Rabin Karp Algorithm

Finite Automata based Algorithm

Boyer Moore Algorithm

All of the above algorithms preprocess the pattern to make the pattern searching faster. The best time complexity that we could get by preprocessing pattern is O(n) where n is length of the text. In this post, we will discuss an approach that preprocesses the text. A suffix tree is built of the text. After preprocessing text (building suffix tree of text), we can search any pattern in O(m) time where m is length of the pattern. Imagine you have stored complete work of William Shakespeare and preprocessed it. You can search any string in the complete work in time just proportional to length of the pattern. This is really a great improvement because length of pattern is generally much smaller than text.

Preprocessing of text may become costly if the text changes frequently. It is good for fixed text or less frequently changing text though.
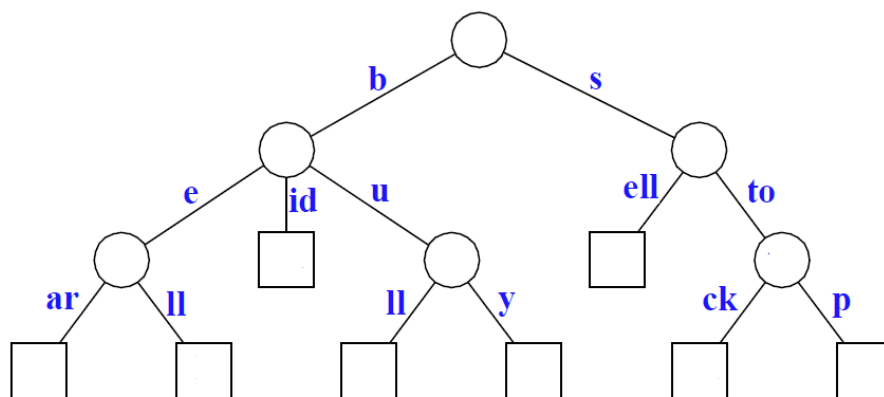
A Suffix Tree for a given text is a compressed trie for all suffixes of the given text. We have discussed Standard Trie. Let us understand **Compressed Trie** with the following array of words.

```
{bear, bell, bid, bull, buy, sell, stock, stop}
```

Following is standard trie for the above input set of words.
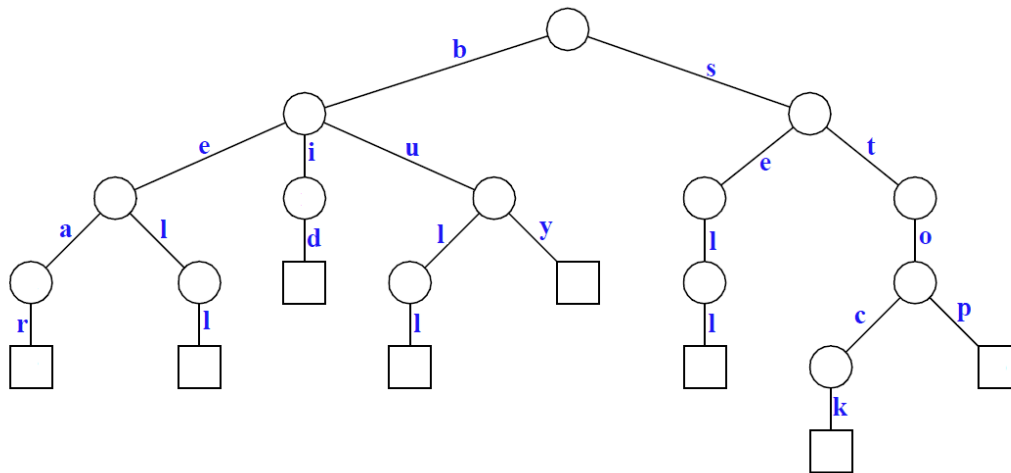
Following is the compressed trie. Compress Trie is obtained from standard trie by joining chains of single nodes. The nodes of a compressed trie can be stored by storing index ranges at the nodes.


How to build a Suffix Tree for a given text?

As discussed above, Suffix Tree is compressed trie of all suffixes, so following are very abstract steps to build a suffix tree from given text.

1) Generate all suffixes of given text.

2) Consider all suffixes as individual words and build a compressed trie.

Let us consider an example text "banana\0″ where '\0′ is string termination character. Following are all suffixes of "banana\0″

```
banana\0
anana\0
nana\0
ana\0
na\0
a\0
\0
```

If we consider all of the above suffixes as individual words and build a trie, we get following.
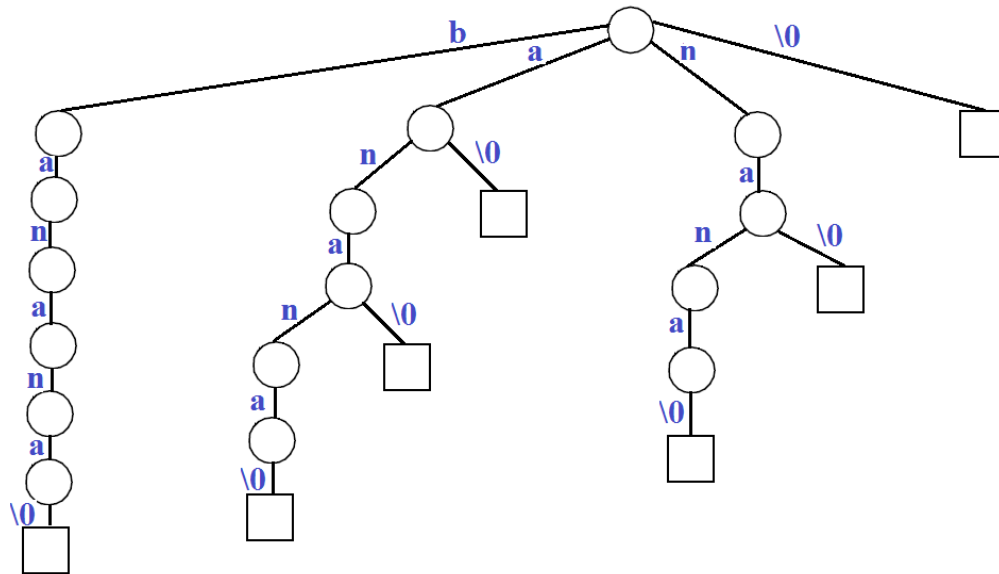
If we join chains of single nodes, we get the following compressed trie, which is the Suffix Tree for given text "banana\0"
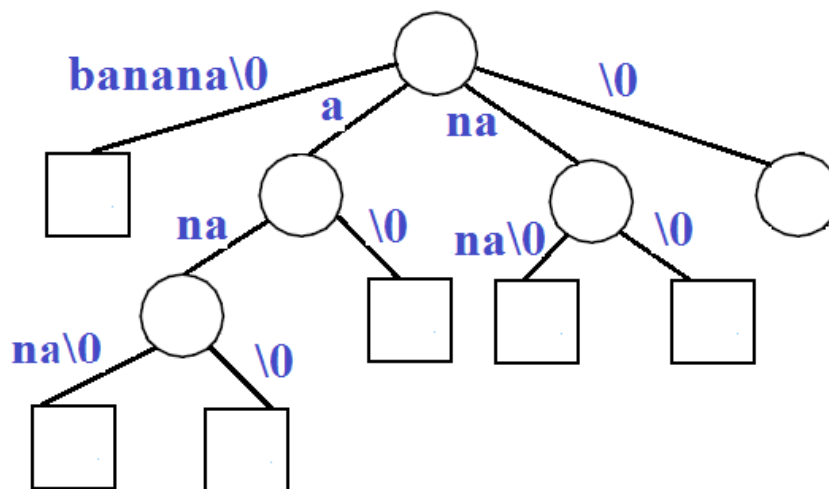

Please note that above steps are just to manually create a Suffix Tree. We will be discussing actual algorithm and implementation in a separate post.

How to search a pattern in the built suffix tree?
We have discussed above how to build a Suffix Tree which is needed as a preprocessing step in pattern searching. Following are abstract steps to search a pattern in the built Suffix Tree.
1) Starting from the first character of the pattern and root of Suffix Tree, do following for every character.
…..**a)** For the current character of pattern, if there is an edge from the current node of suffix tree, follow the edge.
…..**b)** If there is no edge, print "pattern doesn't exist in text" and return.
2) If all characters of pattern have been processed, i.e., there is a path from root for characters of the given pattern, then print "Pattern found".

Let us consider the example pattern as "nan" to see the searching process. Following diagram shows the path followed for searching "nan" or "nana".
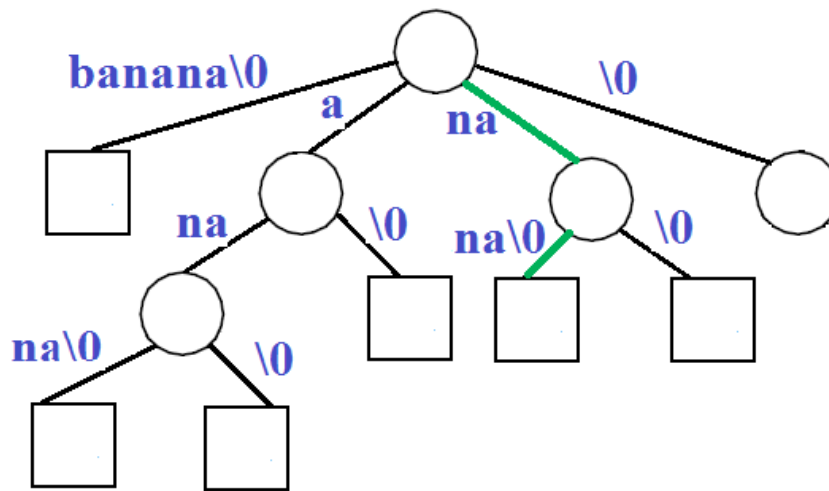
How does this work?

Every pattern that is present in text (or we can say every substring of text) must be a prefix of one of all possible suffixes. The statement seems complicated, but it is a simple statement, we just need to take an example to check validity of it.

Applications of Suffix Tree

Suffix tree can be used for a wide range of problems. Following are some famous problems where Suffix Trees provide optimal time complexity solution.

1) Pattern Searching

2) Finding the longest repeated substring

3) Finding the longest common substring

4) Finding the longest palindrome in a string

There are many more applications. See this for more details.

Ukkonen's Suffix Tree Construction is discussed in following articles:

Ukkonen's Suffix Tree Construction – Part 1

Ukkonen's Suffix Tree Construction – Part 2

Ukkonen's Suffix Tree Construction – Part 3

Ukkonen's Suffix Tree Construction – Part 4

Ukkonen's Suffix Tree Construction – Part 5

Ukkonen's Suffix Tree Construction – Part 6

References:

http://fbim.fh-regensburg.de/~saj39122/sal/skript/progr/pr45102/Tries.pdf

http://www.cs.ucf.edu/~shzhang/Combio12/lec3.pdf

http://www.allisons.org/ll/AlgDS/Tree/Suffix/

15 Comments Category: Misc Tags: Advance Data Structures , Advanced Data Structures , Pattern Searching

Related Questions:

- Minimum steps to reach a destination

- Count numbers from 1 to n that have 4 as a a digit
- Compute sum of digits in all numbers from 1 to n
- Find memory conflicts among multiple threads
- K'th largest element in a stream
- Print all distinct permutations of a given string with duplicates
- Euler's Totient Function
- Given p and n, find the largest x such that p^x divides n!

Like ⟨ 19 ⟩   **Tweet** ⟨ 4 ⟩ **g+1** ⟨ 2 ⟩

D   Your Disqus account has been created! Learn more about using Disqus on your favorite communities.   Get Started

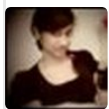   Dismiss ✖

15 Comments   **GeeksforGeeks**   💬 **Priyanka Khire** ⌄

♥ **Recommend**   ⤴ **Share**   Sort by Newest ⌄

Join the discussion…

prashant jha · 4 months ago
here is my c code for suffix tree(compressed)
http://ideone.com/GDZsjH
⌃ | ⌄ · Reply · Share ›

Guest · 4 months ago
here is my c++ implementation of compressed suffix tree
http://ideone.com/oDlfTB
⌃ | ⌄ · Reply · Share ›

Ayush Jain · a year ago
@GeeksforGeeks could you please post it's implementation also?
⌃ | ⌄ · Reply · Share ›

Anurag Singh ➜ Ayush Jain · 8 months ago
We have added articles on suffix tree construction and link is provided above in the end of current article
⌃ | ⌄ · Reply · Share ›

Guest_1923 · a year ago
How does making the tree compressed help, and what is the trade off if we use a normal tries?
⌃ | ⌄ · Reply · Share ›

Anurag Singh ➜ Guest_1923 · 8 months ago

The main issue with normal trie is the huge space it takes. Constructing a suffix trie takes O(N2) time and space. This quadratic performance rules out the use of suffix tries sometimes where they are needed the most: to search through long blocks of data. On the other hand, suffix tree can be created in linear time and space. We have discussed Ukkonen's Suffix Tree Construction in separate articles and it's link is provided at the end of the article above.

ᐱ | ᐯ · Reply · Share ›

Tanu Saxena · a year ago

This pattern searching in the given suffix tree takes O(m) time, where m is the length of the pattern. What about getting all the occurrences of the pattern in the text?? I've read that if there are k occurrences of a pattern, then time complexity of finding them all in the text is O(m+k). [Definitely , pre -processing time is O(n): n being the length of the text].

3 ᐱ | ᐯ · Reply · Share ›

Arpit Baheti · 2 years ago

This is not complete implementation but it build trie for each suffix of a given pattern just how to compress the trie for the node having only one child is missing.please suggest or give code.

```cpp
#include <string>

#include <iostream>

#include <vector>

using namespace std;

typedef struct trie

{

    char data;
```

see more

ᐱ | ᐯ · Reply · Share ›

minoz · 2 years ago

could you please post the implementation of suffix trees soon?

```
/* Paste your code here (You may delete these lines if not writing code) */
```

3 ᐱ | ᐯ · Reply · Share ›

Anurag Singh ➔ minoz · 8 months ago

We have added articles on suffix tree construction and it's link is provided above in the end of current article.

ᴧ | ᴠ · Reply · Share ›

Gangadhara · 2 years ago

```cpp
#include<iostream>
using namespace std;

char* Sub(char* main,char* sub)
{
        int i=0,k=0,test=0,j=0;
        while(*(main+i))
        {
                if(!*(sub+j))break;

                k=i++;
                if(*(main+(k++))==*(sub+(j++)))
                {
                        cout<<(main+i)<<endl;

                        while(1)
                        {
```

see more

ᴧ | ᴠ · Reply · Share ›

Asap · 2 years ago

```c
 #include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct tree
{
int si;
int ei;
struct tree* below;
struct tree* right;
};
typedef struct tree* node;

node newnode(int start,int end)
{
node temp=(node)malloc(sizeof(struct tree));
temp->below=NULL;
temp->right=NULL;
```

```
temp->si=start;
```

see more

∧ | ∨ · Reply · Share ›

rahul → Asap · 2 years ago

please provide little explanation of this code.

```
/* Paste your code here (You may delete these lines if not writing c
```

∧ | ∨ · Reply · Share ›

atul · 3 years ago

eagerly waiting implementation part

```
/* Paste your code here (You may delete these lines if not writing code) */
```

3 ∧ | ∨ · Reply · Share ›

Anurag Singh → atul · 8 months ago

We have added articles on suffix tree construction and link is provided above in the end of current article. Please see if that helps.

∧ | ∨ · Reply · Share ›