Suffix Array | Set 1 (Introduction)

We strongly recommend to read following post on suffix trees as a pre-requisite for this post.

Pattern Searching | Set 8 (Suffix Tree Introduction)

A suffix array is a sorted array of all suffixes of a given string. The definition is similar to Suffix Tree which is compressed trie of all suffixes of the given text. Any suffix tree based algorithm can be replaced with an algorithm that uses a suffix array enhanced with additional information and solves the same problem in the same time complexity (Source Wiki).
A suffix array can be constructed from Suffix tree by doing a DFS traversal of the suffix tree. In fact Suffix array and suffix tree both can be constructed from each other in linear time.
Advantages of suffix arrays over suffix trees include improved space requirements, simpler linear time construction algorithms (e.g., compared to Ukkonen's algorithm) and improved cache locality (Source: Wiki)

Example:

```
Let the given string be "banana".

0 banana                          5 a
1 anana     Sort the Suffixes     3 ana
2 nana      ---------------->     1 anana
3 ana        alphabetically       0 banana
4 na                              4 na
5 a                               2 nana

So the suffix array for "banana" is {5, 3, 1, 0, 4, 2}
```


Naive method to build Suffix Array

A simple method to construct suffix array is to make an array of all suffixes and then sort the array. Following is implementation of simple method.

```
// Naive algorithm for building suffix array of a given text
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

// Structure to store information of a suffix
struct suffix
{
    int index;
    char *suff;
};

// A comparison function used by sort() to compare two suffixes
int cmp(struct suffix a, struct suffix b)
{
    return strcmp(a.suff, b.suff) < 0? 1 : 0;
}

// This is the main function that takes a string 'txt' of size n as an
// argument, builds and return the suffix array for the given string
int *buildSuffixArray(char *txt, int n)
{
    // A structure to store suffixes and their indexes
    struct suffix suffixes[n];

    // Store suffixes and their indexes in an array of structures.
    // The structure is needed to sort the suffixes alphabatically
    // and maintain their old indexes while sorting
    for (int i = 0; i < n; i++)
    {
        suffixes[i].index = i;
        suffixes[i].suff = (txt+i);
    }

    // Sort the suffixes using the comparison function
    // defined above.
    sort(suffixes, suffixes+n, cmp);

    // Store indexes of all sorted suffixes in the suffix array
    int *suffixArr = new int[n];
    for (int i = 0; i < n; i++)
```


Popular Posts

```
        suffixArr[i] = suffixes[i].index;

    // Return the suffix array
    return  suffixArr;
}

// A utility function to print an array of given size
void printArr(int arr[], int n)
{
    for(int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// Driver program to test above functions
int main()
{
    char txt[] = "banana";
    int n = strlen(txt);
    int *suffixArr = buildSuffixArray(txt,  n);
    cout << "Following is suffix array for " << txt << endl;
    printArr(suffixArr, n);
    return 0;
}
```

Output:

```
Following is suffix array for banana
5 3 1 0 4 2
```

The time complexity of above method to build suffix array is $O(n^2Logn)$ if we consider a O(nLogn) algorithm used for sorting. The sorting step itself takes $O(n^2Logn)$ time as every comparison is a comparison of two strings and the comparison takes O(n) time.
There are many efficient algorithms to build suffix array. We will soon be covering them as separate posts.

Search a pattern using the built Suffix Array

To search a pattern in a text, we preprocess the text and build a suffix array of the text. Since we have a sorted array of all suffixes, Binary Search can be used to search. Following is the search function. Note that the function doesn't report all occurrences of pattern, it only report one of them.

```
// This code only contains search() and main. To make it a complete ru
// above code or see http://ideone.com/1Io9eN
```

```
// A suffix array based search function to search a given pattern
// 'pat' in given text 'txt' using suffix array suffArr[]
void search(char *pat, char *txt, int *suffArr, int n)
{
    int m = strlen(pat);  // get length of pattern, needed for strncmp

    // Do simple binary search for the pat in txt using the
    // built suffix array
    int l = 0, r = n-1;  // Initilize left and right indexes
    while (l <= r)
    {
        // See if 'pat' is prefix of middle suffix in suffix array
        int mid = l + (r - l)/2;
        int res = strncmp(pat, txt+suffArr[mid], m);

        // If match found at the middle, print it and return
        if (res == 0)
        {
            cout << "Pattern found at index " << suffArr[mid];
            return;
        }

        // Move to left half if pattern is alphabtically less than
        // the mid suffix
        if (res < 0) r = mid - 1;

        // Otherwise move to right half
        else l = mid + 1;
    }

    // We reach here if return statement in loop is not executed
    cout << "Pattern not found";
}
```

```
// Driver program to test above function
int main()
{
    char txt[] = "banana";  // text
    char pat[] = "nan";   // pattern to be searched in text

    // Build suffix array
    int n = strlen(txt);
    int *suffArr = buildSuffixArray(txt, n);
```


Recent Comments

```
    // search pat in txt using the built suffix array
    search(pat, txt, suffArr, n);

    return 0;
}
```

Output:

```
Pattern found at index 2
```

The time complexity of the above search function is O(mLogn). There are more efficient algorithms to search pattern once the suffix array is built. In fact there is a O(m) suffix array based algorithm to search a pattern. We will soon be discussing efficient algorithm for search.

Applications of Suffix Array

Suffix array is an extremely useful data structure, it can be used for a wide range of problems. Following are some famous problems where Suffix array can be used.

1) Pattern Searching

2) Finding the longest repeated substring

3) Finding the longest common substring

4) Finding the longest palindrome in a string

See this for more problems where Suffix arrays can be used.

This post is a simple introduction. There is a lot to cover in Suffix arrays. We have discussed a O(nLogn) algorithm for Suffix Array construction here. We will soon be discussing more efficient suffix array algorithms.

References:

http://www.stanford.edu/class/cs97si/suffix-array.pdf

http://en.wikipedia.org/wiki/Suffix_array

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Related Tpoics:

- Remove minimum elements from either side such that 2*min becomes more than max
- Divide and Conquer | Set 6 (Search in a Row-wise and Column-wise Sorted 2D Array)
- Bucket Sort
- Kth smallest element in a row-wise and column-wise sorted 2D array | Set 1
- Find the number of zeroes
- Find if there is a subarray with 0 sum
- Divide and Conquer | Set 5 (Strassen's Matrix Multiplication)
- Count all possible groups of size 2 or 3 that have sum as multiple of 3

Writing code in comment? Please use **ideone.com** and share the link here.

14 Comments GeeksforGeeks

Sort by Newest

ram • 8 days ago

This uses hell of memory. isnt it ?

 • Reply • Share ›

kaushik Lele • 23 days ago

I have followed "bucket" or "map" methodology to developed code.
e.g. for string "banab"
Starting character is 'b' so bucket of 'b' with value "anab" ('b' is node value)
Then next substring is "anab". So bucket of 'a' with value "nab" ('a' is node valu

Similarly another bucket 'n' for value "anab"

Now next string is "ab". We already have bucket for 'a' with value "anab".

So we will create subbuckets

Bucket 'a' -> sub-bucket 'n' with value "ab" (starting "a" and "n" are from bucke
Bucket 'a' -> sub-bucket 'b' with value null (starting "a" from bucket)

If there was substring "anan..". Then we would have repeated steps
Bucket 'a' -> sub-bucket 'n' -> sub-sub-bucket 'a' and so on

Refer my code for word "banana". It prints value as expected.
5 a
3 ana

see more

 • Reply • Share ›

sambhavsharma • 3 months ago

What if we search for "bana" or "ban"? I doesn't work for these patterns?

• Reply • Share ›

anonymous sambhavsharma • a month ago

we are using strncmp(text, pattern, pattern_length) that why it matche

Reply • Share ›

Kartik sambhavsharma • 3 months ago

Please take a closer look. The program works fine for these cases als

Reply • Share ›

sambhavsharma Kartik • 3 months ago

I think mid = l + (r-1)/2 might not be the right condition,
It should be (l+r)/2

Reply • Share ›

Kartik sambhavsharma • 3 months ago

Please see http://googleresearch.blogspot...

Reply • Share ›

sambhavsharma Kartik • 3 months ago

This code doesn't work for the current condition.
for value "bana".
and the document says one possible fix could be:

int mid = low + ((high - low) / 2);

Reply • Share ›

Saurabh Verma • 3 months ago

how to write code in comment box??

Reply • Share ›

Kartik Saurabh Verma • 3 months ago

Looks like code in disqus comment system doesn't work very well. Usi

link seems to be a good idea.

• Reply • Share ›

CREATORS93 . • 3 months ago

this is great

• Reply • Share ›

geek • 3 months ago

There is a problem in the implementation of pattern search function search. TI
should not be used, because it search for exact match. It is possible that patte
element, so instead of strcmp I would suggest using of Find() function. For exa
your implementation fails and returns pattern not found although it is present, I

1 • Reply • Share ›

GeeksforGeeks ➚ geek • 3 months ago

Geeks, thanks for pointing this out. We have updated the code to use s
http://www.cplusplus.com/refer...

• Reply • Share ›

Thewar • 4 months ago

Most awaited post :)

2 • Reply • Share ›

Subscribe Add Disqus to your site