Suffix Array | Set 1 (Introduction) - GeeksforGeeks

Source: https://www.geeksforgeeks.org/suffix-array-set-1-introduction/

Suffix Array | Set 1 (Introduction) Last Updated : 23 Jul, 2025 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. CPP // 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 alphabetically // 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 ++) 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 ; } Java //importing the required packages import java.util.ArrayList ; import java.util.Arrays ; class suffix_array { public static void main (String [] args) throws Exception { String word = "banana" ; String arr1 [] = new String [word . length ()] ; String arr2 [] = new String [word . length ()] ; ArrayList < Integer > suffix_index = new ArrayList < Integer > (); int suffix_array [] = new int [word . length ()] ; for (int i = 0 ; i < word . length (); i ++) { arr1 [i] = word . substring (i); } arr2 = arr1 . clone (); Arrays . sort (arr1); for (String i : arr1) { String piece = i ; int index = new suffix_array (). index (arr2 , piece); suffix_index . add (index); } for (int i = 0 ; i < suffix_array . length ; i ++) { suffix_array [i] = suffix_index . get (i); } System . out . println ("following is the suffix array for banana"); for (int i : suffix_array) { System . out . print (i + " "); } } //simple function to return the index of item from array arr[] int index (String arr [] , String item) { for (int i = 0 ; i < arr . length ; i ++) { if (item == arr [i]) return i ; } return - 1 ; } } Python # Naive algorithm for building suffix array of a given text import sys class Suffix : def __init__ (self , index , suff): self . index = index self . suff = suff # A comparison function used by sort() to compare two suffixes def cmp (a , b): return

(a . suff < b . suff) - (a . suff > b . suff) # 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 def build_suffix_array (txt , n): # A structure to store suffixes and their indexes suffixes = [Suffix (i , txt [i :]) for i in range (n)] # Sort the suffixes using the comparison function # defined above. suffixes . sort (key = cmp) # Store indexes of all sorted suffixes in the suffix array suffix_arr = [suffixes [i] . index for i in range (n)] # Return the suffix array return suffix_arr # A utility function to print an array of given size def print_arr (arr): for i in range (len (arr)): print (arr [i], end = " ") print () # Driver program to test above functions def main (): txt = "banana" n = len (txt) suffix_arr = build_suffix_array (txt , n) print ("Following is suffix array for" , txt) print_arr (suffix_arr) if __name__ == "__main__" : main () C# // Naive algorithm for building suffix array of a given text using System ; using System.Linq ; class SuffixArray { private readonly string _text ; private readonly int [] _suffixArray ; private struct Suffix { public int Index { get ; set ; } public string Suff { get ; set ; } } private static int CompareSuffixes (Suffix a , Suffix b) { return string . Compare (a . Suff , b . Suff , StringComparison . Ordinal); } public SuffixArray (string text) { _text = text ; var n = _text . Length ; var suffixes = new Suffix [n]; for (var i = 0 ; i < n ; i ++) { suffixes [i] = new Suffix { Index = i , Suff = _text . Substring (i) }; } Array . Sort (suffixes , CompareSuffixes); _suffixArray = new int [n]; for (var i = 0 ; i < n ; i ++) { _suffixArray [i] = suffixes [i]. Index ; } } public int [] GetSuffixArray () { return _suffixArray ; } } class Program { static void Main (string [] args) { var txt = "banana" ; var suffixArray = new SuffixArray (txt); Console . WriteLine ("Following is suffix array for " + txt); Console . WriteLine (string . Join (" " , suffixArray . GetSuffixArray (). Select (x => x . ToString ()))); } } // This code is contributed by snehalsalokhe JavaScript // Naive algorithm for building suffix array of a given text // A comparison function used by sort() to compare two suffixes function comp (a , b) { const name1 = a . suff ; const name2 = b . suff ; let comparison = 0 ; if (name1 > name2) { comparison = 1 ; } else if (name1 < name2) { comparison = - 1 ; } return comparison ; } // 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 function buildSuffixArray (txt , n) { // A Array to store suffixes and their indexes var suffixes = []; // Store suffixes and their indexes in an array of objects. // The object is needed to sort the suffixes alphabetically // and maintain their old indexes while sorting for (var i = 0 ; i < n ; i ++) { var suffix = {}; suffix . index = i ; suffix . suff = txt . substr (i); suffixes . push (suffix); } // Sort the suffixes using the comparison function defined above. suffixes . sort (comp); // Store indexes of all sorted suffixes in the suffix array var suffixArr = []; for (var i = 0 ; i < n ; i ++) suffixArr [i] = suffixes [i]. index ; // Return the suffix array return suffixArr ; } // A utility function to print an array of given size function printArr (arr , n) { for (var i = 0 ; i < n ; i ++) { console . log (arr [i]); } } // Driver program to test above functions var txt = "banana" ; var n = txt . length ; var suffixArr = buildSuffixArray (txt , n); console . log ("Following is suffix array for " + txt); printArr (suffixArr , n); Output Following is suffix array for banana 5 3 1 0 4 2 Time Complexity: O(n*k*Logn). if we consider a O(nLogn)) algorithm used for sorting. The sorting step itself takes O(n*k*Logn) time as every comparison is a comparison of two strings and the comparison takes O(K) time where K is max length of string in given array. Auxiliary Space: O(n) 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. CPP // This code only contains search() and main. To make it a complete running // above code or see // 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 ; // Initialize 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 alphabetically 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); // search pat in txt using the built suffix array search (pat , txt , suffArr , n); return 0 ; } Java import java.io.* ; public class GFG { // A suffix array based search function to search a given pattern // 'pat' in given text 'txt' using suffix array suffArr[] static void search (String pat , String txt , int [] suffArr , int n) { // Get the length of the pattern int m = pat . length (); // Initialize left and right indexes int l = 0 ; int r = n - 1 ; // Do simple binary search for the pat in

```
txt using the built suffix array while ( l <= r ) { // Find the middle index of the current subarray int mid = l
+ ( r - l ) / 2 ; // Get the substring of txt starting from suffArr[mid] and of length m String res = txt .
substring ( suffArr [ mid ] , suffArr [ mid ] + m ); // If the substring is equal to the pattern if ( res . equals (
pat )) { // Print the index and return System . out . println ( "Pattern found at index " + suffArr [ mid ] );
return ; } // If the substring is less than the pattern if ( res . compareTo ( pat ) < 0 ) { // Move to the right
half of the subarray l = mid + 1 ; } else { // Move to the left half of the subarray r = mid - 1 ; } } // If the
pattern is not found System . out . println ( "Pattern not found" ); } static int [] buildSuffixArray ( String txt
, int n ) { // Create a list of all suffixes String [] suffixes = new String [ n ] ; for ( int i = 0 ; i < n ; i ++ ) {
suffixes [ i ] = txt . substring ( i , n ); } // Sort the suffixes java . util . Arrays . sort ( suffixes ); // Create the
suffix array int [] suffArr = new int [ n ] ; for ( int i = 0 ; i < n ; i ++ ) { suffArr [ i ] = txt . indexOf ( suffixes [ i
] ); } return suffArr ; } // Driver program to test above function public static void main ( String [] args ) {
String txt = "banana" ; // text String pat = "nan" ; // pattern to be searched in text // Build suffix array int n
= txt . length (); int [] suffArr = buildSuffixArray ( txt , n ); // search pat in txt using the built suffix array
search ( pat , txt , suffArr , n ); } } //This code is contributed by shivamsharma215 Python import sys # A
suffix array based search function to search a given pattern # 'pat' in given text 'txt' using suffix array
suffArr[] def search ( pat , txt , suffArr , n ): # Get the length of the pattern m = len ( pat ) # Initialize left
and right indexes l = 0 r = n - 1 # Do simple binary search for the pat in txt using the built suffix array
while l <= r : # Find the middle index of the current subarray mid = l + ( r - l ) // 2 # Get the substring of
txt starting from suffArr[mid] and of length m res = txt [ suffArr [ mid ]: suffArr [ mid ] + m ] # If the
substring is equal to the pattern if res == pat : # Print the index and return print ( "Pattern found at
index" , suffArr [ mid ]) return # If the substring is less than the pattern if res < pat : # Move to the right
half of the subarray l = mid + 1 else : # Move to the left half of the subarray r = mid - 1 # If the pattern is
not found print ( "Pattern not found" ) def buildSuffixArray ( txt , n ): # Create a list of all suffixes suffixes
= [ txt [ i :] for i in range ( n )] # Sort the suffixes suffixes . sort () # Create the suffix array suffArr = [ txt .
index ( suffix ) for suffix in suffixes ] return suffArr # Driver program to test above function def main ():
txt = "banana" # text pat = "nan" # pattern to be searched in text # Build suffix array n = len ( txt ) suffArr
= buildSuffixArray ( txt , n ) # search pat in txt using the built suffix array search ( pat , txt , suffArr , n )
return 0 if __name__ == '__main__' : sys . exit ( main ()) # This code is contributed by Vikram_Shirsat
C# using System ; class GFG { // A suffix array based search function to search a // given pattern 'pat'
in given text 'txt' using suffix // array suffArr[] static void Search ( string pat , string txt , int [] suffArr , int
n ) { // Get the length of the pattern int m = pat . Length ; // Initialize left and right indexes int l = 0 ; int r =
n - 1 ; // Do simple binary search for the pat in txt using // the built suffix array while ( l <= r ) { // Find the
middle index of the current subarray int mid = l + ( r - l ) / 2 ; // Get the substring of txt starting from //
suffArr[mid] and of length m string res = txt . Substring ( suffArr [ mid ], m ); // If the substring is equal to
the pattern if ( res . Equals ( pat )) { // Print the index and return Console . WriteLine ( "Pattern found at
index " + suffArr [ mid ]); return ; } // If the substring is less than the pattern if ( res . CompareTo ( pat ) <
0 ) { // Move to the right half of the subarray l = mid + 1 ; } else { // Move to the left half of the subarray r
= mid - 1 ; } } // If the pattern is not found Console . WriteLine ( "Pattern not found" ); } static int []
BuildSuffixArray ( string txt , int n ) { // Create a list of all suffixes string [] suffixes = new string [ n ]; for (
int i = 0 ; i < n ; i ++ ) { suffixes [ i ] = txt . Substring ( i , n - i ); } // Sort the suffixes Array . Sort ( suffixes
); // Create the suffix array int [] suffArr = new int [ n ]; for ( int i = 0 ; i < n ; i ++ ) { suffArr [ i ] = txt .
IndexOf ( suffixes [ i ]); } return suffArr ; } // Driver program to test above function static void Main (
string [] args ) { string txt = "banana" ; // text string pat = "nan" ; // pattern to be searched in text // Build
suffix array int n = txt . Length ; int [] suffArr = BuildSuffixArray ( txt , n ); // search pat in txt using the
built suffix array Search ( pat , txt , suffArr , n ); } } // This code is contributed by prajwal kandekar
JavaScript // A suffix array based search function to search a given pattern // 'pat' in given text 'txt' using
suffix array suffArr[] function search ( pat , txt , suffArr , n ) { var m = pat . length ; // get length of pattern
// Do simple binary search for the pat in txt using the // built suffix array var l = 0 ; var r = n - 1 ; //
Initialize left and right indexes while ( l <= r ) { // See if 'pat' is prefix of middle suffix in suffix array var
mid = l + Math . floor (( r - l ) / 2 ); var c = txt . substring ( suffArr [ mid ]); var res = 0 ; for ( var i = 0 ; i <
m ; i ++ ) { if ( pat [ i ] == c [ i ]) { continue ; } else if ( pat [ i ] < c [ i ]) { res = - 1 ; break ; } else { res = 1 ;
break ; } } // If match found at the middle, print it and return if ( res == 0 ) { console . log ( "Pattern found
at index " + suffArr [ mid ]); return ; } // Move to left half if pattern is alphabetically 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 console . log ( "Pattern not found" ); } function comp ( a , b ) {
const name1 = a . suff ; const name2 = b . suff ; let comparison = 0 ; if ( name1 > name2 ) { comparison
= 1 ; } else if ( name1 < name2 ) { comparison = - 1 ; } return comparison ; } function buildSuffixArray (
```

txt , n) { // A Array to store suffixes and their indexes var suffixes = []; // Store suffixes and their indexes in an array of objects. // The object is needed to sort the suffixes alphabetically // and maintain their old indexes while sorting for (var i = 0 ; i < n ; i ++) { var suffix = {}; suffix . index = i ; suffix . suff = txt . substr (i); suffixes . push (suffix); } // Sort the suffixes using the comparison function defined above. suffixes . sort (comp); // Store indexes of all sorted suffixes in the suffix array var suffixArr = []; for (var i = 0 ; i < n ; i ++) suffixArr [i] = suffixes [i]. index ; // Return the suffix array return suffixArr ; } var txt = "banana" ; // text var pat = "nan" ; // pattern to be searched in text // Build suffix array var n = txt . length ; var suffArr = buildSuffixArray (txt , n); // search pat in txt using the built suffix array search (pat , txt , suffArr , n); Output: Pattern found at index 2 Time Complexity: O(mlogn) Auxiliary Space: O(m+n) 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: https://web.stanford.edu/class/cs97si/suffix-array.pdf https://en.wikipedia.org/wiki/Suffix_array Comment Article Tags: Article Tags: Pattern Searching Advanced Data Structure DSA Suffix-Array