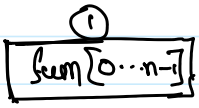
Segment Trees

Wednesday, 20 January 2021 3:06 PM

Fenwick tree	Segment tree
find sum of given range	find sum of given range
find product of given range	find product of range
	find min of given range
	find max of given range
TC - O(logn)	TC - O(logn)
SC - O(n)	SC - O(4n)

→ Segment trees store the ranges of sum (or) product (or) min (or) max. like this —

a) It starts with index 1 and at index 1 it store sum for all values of array i.e., 0 to n-1



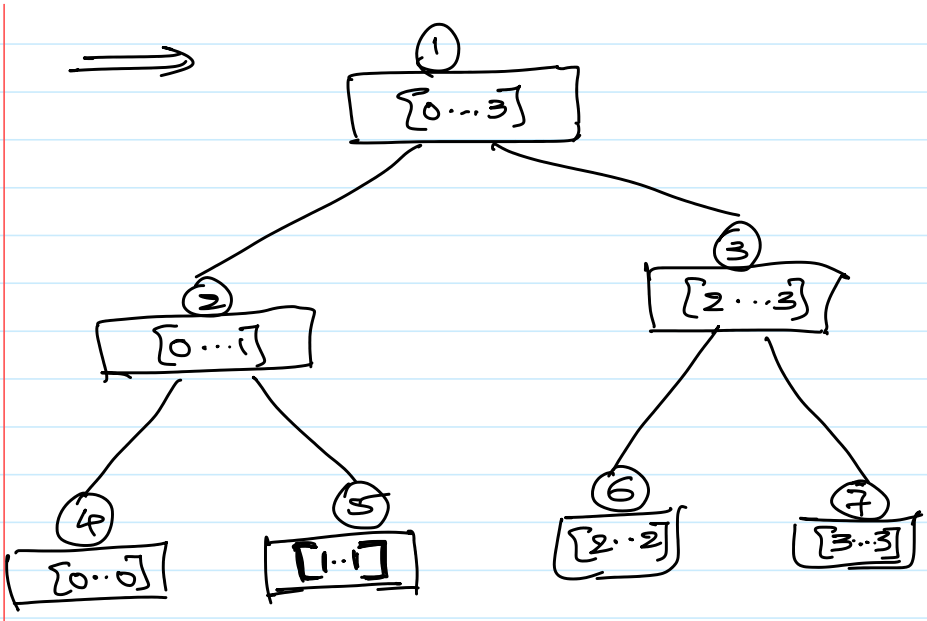
(∵ If we have index i in array then, its children are at 2*i , 2*i+1)

b) Now it will store the children sums i.e., $[0 \ldots \frac{n-1}{2}]$ and $[\frac{n-1}{2}+1 , n-1]$ at indices 1*2 , 1*2+1 respectively.

(∵ left = 0 ; right = n-1 ⟹ middle = $\frac{0+n-1}{2} = \frac{n-1}{2}$)

Eg:- Let, n = 4

⟹



Building a segment tree :- (O(n log n))

→ We start with index v = 1, and store the sum from tl = 0 to tr = n-1 in index v = 1

→ After that we store at indices 2*v, 2*v+1 from range tl to tm and tm+1 to tr.

$$\left(tm = \frac{tl + tr}{2}\right)$$

→ If tl == tr, that means the range contains only 1 value

⟹ segment[v] = arr[tl].

Build code :-

```
void build(int a[], int v, int tl, int tr) {
    if (tl == tr) {
        t[v] = a[tl];
    } else {
        int tm = (tl + tr) / 2;
        build(a, v*2, tl, tm);
        build(a, v*2+1, tm+1, tr);
        t[v] = t[v*2] + t[v*2+1];
    }
}
```

→ this is for sum. For product we can initialize all the values of t[] with 1 and $t[v] = t[v*2] * t[v*2+1]$

→ for minimum -

$t[v] = Math.min(t[v*2], t[v*2+1])$

→ for maximum -

$t[v] = Math.max(t[v*2], t[v*2+1])$

Update in segment tree is

→ We start from 1st index. We are given a new value and position to update.

→ We can observe that always left subtree range is tl to tm and right subtree range is tm+1 to tr.

⟹ if pos <= tm, then we call the left subtree recursively.
else, we call the right subtree

else, we call the right subtree recursively.

Update Code:—

```
void update(int v, int tl, int tr, int pos, int new_val) {
    if (tl == tr) {
        t[v] = new_val;
    } else {
        int tm = (tl + tr) / 2;
        if (pos <= tm)
            update(v*2, tl, tm, pos, new_val);
        else
            update(v*2+1, tm+1, tr, pos, new_val);
        t[v] = t[v*2] + t[v*2+1];
    }
}
```

→ this is for sum. For product we can initialize all the values of t[] with 1 and $t[v] = t[v*2] * t[v*2+1]$

→ for minimum –

$t[v] = Math.min(t[v*2], t[v*2+1])$

→ for maximum –

$t[v] = Math.max(t[v*2], t[v*2+1])$

Query in segment tree :—

→ We are given the range in which we have to return the answer. they are l and r.

→ If $l > r$, that is a invalid call we return 0;

we return 0;

→ If l == tl and r == tr, we can see that we have arrived at desired result

⟹ return t[v];

→ Else,

$tm = (tl + tr)/2$;

a) We know that left subtree has maximum index upto tm. Even if r > tm we can retrieve only upto tm in the left subtree So, we take l to min(r, tm).

b) We know that left subtree starts with tm+1. If l > tm+1, we only need the sum from l. So, we take from max(l, tm+1) to r.

Query code :-

```
int sum(int v, int tl, int tr, int l, int r) {
    if (l > r)
        return 0;
    if (l == tl && r == tr) {
        return t[v];
    }
    int tm = (tl + tr) / 2;
    return sum(v*2, tl, tm, l, min(r, tm))
           + sum(v*2+1, tm+1, tr, max(l, tm+1), r);
}
```

→ this is for sum.

→ for product —

if l > r, we return 1;

and generally we

return product(v*2, tl, tm, l, min(r, tm))
* product(v*2+1, tm+1, tr, max(tm+1, l), r)

→ for minimum —

if l > r, we return Integer.MAX_VALUE;

(∵ Because when finding min, Max value doesn't alter the result)

generally —

return min(findMin(), findMin());

→ for maximum —

if l > r, return Integer.MIN_VALUE;

generally —

return max(findMax(), findMax());

Program 1

Thursday, January 21, 2021 1:31 PM

Problem Statement:

Malika taught a new fun time program practice for Engineering Students.
As a part of this she has given set of numbers, and asked the students
to find the minimum number between indices S1 and S2 (S1<=S2), inclusive.

Now it's your task to implement the Solution class:

segmentTree(int[] nums, int n):

- build the Segment tree with the integer array nums[].
- n is size of the array.

int findMinimum(int n, int S1, int S2):

- n is size of the array.
- S1, S2 are indices

Returns the minimum value in the subarray nums[S1, S2]
(i.e., nums[S1] + nums[S1 + 1], ..., nums[S2]).

Input Format:

Line-1: An integer n, size of the array nums[] (set of numbers).
where 1<= n <= 50000
Line-2: Two integers S1 and S2, index positions
where 0 <= S1 <= S2 < n
and 1<= nums[i] <= 99999.

Output Format:

An integer, sum of integers between indices(s1, s2).

Sample Input-1:

10
2 9

Sample Output-1:

10208

NOTE: First 10 values of the input are:
66905 11444 18252 54299 10208 59466 17861 24128 31974 69081

Code:

```
import java.util.*;

class Solution
{
   int t[];


   void buildTree(int v, int[] nums, int tl, int tr)
   {
      if(tl==tr)
         t[v]=nums[tl];
      else
      {
         int tm=(tl+tr)/2;
         buildTree(2*v,nums,tl,tm);
         buildTree(2*v+1,nums,tm+1,tr);
         t[v]=Math.min(t[2*v],t[2*v+1]);
      }
   }

   // method to implement segment tree
        void segmentTree(int nums[], int n)
        {
           t=new int[4*n];
      buildTree(1,nums,0,n-1);
        }

   int getMin(int v, int tl, int tr, int l, int r)
   {
      if(l>r)
         return Integer.MAX_VALUE;
      if(l==tl && r==tr)
         return t[v];
      int tm=(tl+tr)/2;
      return Math.min(getMin(2*v,tl,tm,l,Math.min(tm,r)),getMin(2*v+1,tm+1,tr,Math.max(l,tm+1),r));
   }

   // method to find the minimum value in the range
        int findMinimum(int n, int s1, int s2)
        {
      return getMin(1,0,n-1,s1,s2);
```

```
    }

}
```

Test Cases:

```
case =1
input =100
4 87
output =312
case =2
input =1000
78 951
output =537
case =3
input =10000
92 8765
output =113
case =4
input =25000
1145 18754
output =109
case =5
input =30000
12321 29876
output =110
case =6
input =35000
21321 34876
output =102
case =7
input =49999
567 43786
output =101
case =8
input =49999
1 575
output =312
```

Program 2

Thursday, January 21, 2021 1:35 PM

→ General method is to iterate over entire vocab for each word and check for given conditions. (My code)

→ We can also use a hashmap with keys as acronyms and value as the string with that acronym.

a) If we encounter is first time we make a key, value in hashmap.

b) Else, we check if previous value for key is same as the value we are about to insert. If yes, we leave it. If NO, we set the value a "" (empty string)

→ Finally, we return true only if the key is not present (or) if it's present the value is same as the word. Else return false.

Problem Statement:

Ananth interested in creating the acronyms for any word.
The definition of acronym is another word with a concatenation of its first letter,
the number of letters between the first and last letter, and its last letter.

If a word has only two characters, then it is an acronym of itself.

Examples:
- Acronym of 'dog' is 'd1g'.
- Acronym of 'another' is 'a5r'.
- Acronym of 'ab' is 'ab'.

You are given a list of vocabulary, and another list of words.
Your task is to check is word with the vocabulary and
return "true" if atleast one of the following rules satisfied:
1. acronym of the word should not match with any of the acronyms of vocabulary
2. if acronym of the word matches with any of the acronyms of vocabulary
'the word' and matching words in vocabulary should be same.
Otherwise, return 'false'.

Input Format:

Line-1: Space separated strings, vocabulary[]
Line-2: Space separated strings, words[]

Output Format:

Print N boolean values, where N = words.length.

Sample Input-1:
---
cool bell cool coir move more mike
cool char move

Sample Output-1:
---
true false false

My Code:

```
import java.util.*;
class Acronyms
{
   public static void main(String args[])
   {
      Scanner sc=new Scanner(System.in);
      String vocab[]=sc.nextLine().split(" ");
      String words[]=sc.nextLine().split(" ");
      int len=vocab.length;
```

```
        String acw="",acv="";
        int wl=0,vl=0;
        int count=0;
        boolean flag=false;
        for(String word:words)
        {
          flag=false;
          count=0;
          wl=word.length();
          if(wl==2)
            acw=word;
          else
            acw=""+word.charAt(0)+(wl-2)+word.charAt(wl-1);

          for(String voc:vocab)
          {
            vl=voc.length();
            if(vl==2)
              acv=voc;
            else
              acv=""+voc.charAt(0)+(vl-2)+voc.charAt(vl-1);

            if(!acv.equals(acw))
            {
              count++;
            }
            else if(acv.equals(acw) && word.equals(voc))
            {
              flag=true;
            }
            else
            {
              flag=false;
              break;
            }
          }
          if(count==len)
          {
            flag=true;
          }
          System.out.print(flag+" ");
        }

      }
}
```

Optimized Code (Using Hashmap):

```
import java.util.*;
public class ValidWordAbbr {
    HashMap<String, String> map;
    public ValidWordAbbr(String[] dictionary) {
        map = new HashMap<String, String>();
        for(String str:dictionary){
            String key = getKey(str);
            // If there is more than one string belong to the same key
            // then the key will be invalid, we set the value to ""
            if(map.containsKey(key) && !map.get(key).equals(str))
                map.put(key, "");
            else
                map.put(key, str);
        }
    }
    public boolean isUnique(String word) {
        String key = getKey(word);
        return !map.containsKey(key)||map.get(key).equals(word);
    }
    private String getKey(String str){
        if(str.length()<=2) return str;
        return str.charAt(0)+Integer.toString(str.length()-2)+str.charAt(str.leng
th()-1);
    }

    public static void main(String args[])
    {
        Scanner sc=new Scanner(System.in);
        String dictionary[]=sc.nextLine().split(" ");
        String words[]=sc.nextLine().split(" ");
        ValidWordAbbr vwa=new ValidWordAbbr(dictionary);
        for(String word:words)
            System.out.print(vwa.isUnique(word)+" ");
    }
}
```

Test Cases:

```
case =1
input =deer door cake card
dear cart cane make
output =false true false true
case =2
input =cool bell cool coir move more mike
cool char move
output =true false false
case =3
input =deer door cake card cool bell cool coir move more mike
cool char move dear cart cane make
output =true false false false true false false
case =4
input =deer door cake card cool bell cool coir move more mike monk masi mari mure
 musc mact monk
mark rock move more nike monk cool char move dear cart cane make
output =false true false false true true true false false false true false false
case =5
input =avarice adaptor aviator admirer however hurdler hurrier
axoneme avraice adaptor aviator howevee hurdler
output =false false false false true false
case =6
input =monk masi mark mure musc mact monk
mark rock move more nike monk
output =false true false false true false
```

Program 3

Wednesday, 20 January 2021 6:43 PM

abc de
ade

	0	1	2	3	4
0	1	0	0	0	0
1	0	1			
2	0				

→ Porterity means Longest Common Subsequence

Recursive approach :-

→ Base condition :-

We can take the string till their sizes becomes '0'.

⟹
```
if(n==0 || m==0)
    return 0;
```

Since, there is no common LCS between them.

→ Main Code :-

→ Let's say we have two strings x and y.

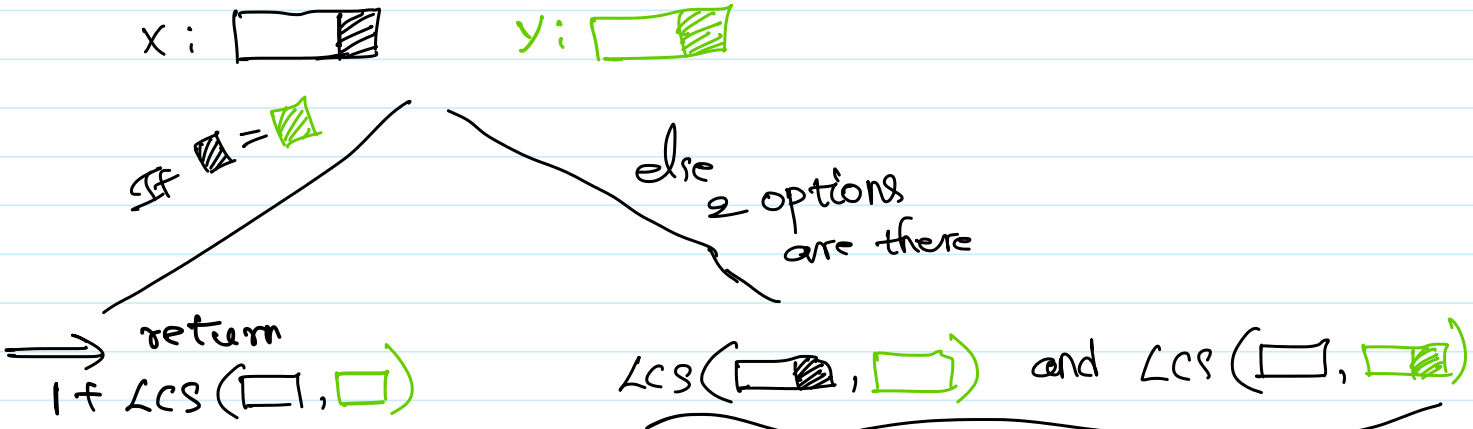
→ We are computing the LCS from end to start of string.

→ Now if we observe at any point x[end] and y[end] can be equal

(or) not equal.

→ If they are equal, then it will contribute to the LCS. So, we'll return (1+LCS(xend-1, yend-1)) because we remove the end of both.

→ If they are not equal, then there's a possibility that we'll find an LCS with LCS(xend, yend-1) (or) LCS(xend-1, yend).



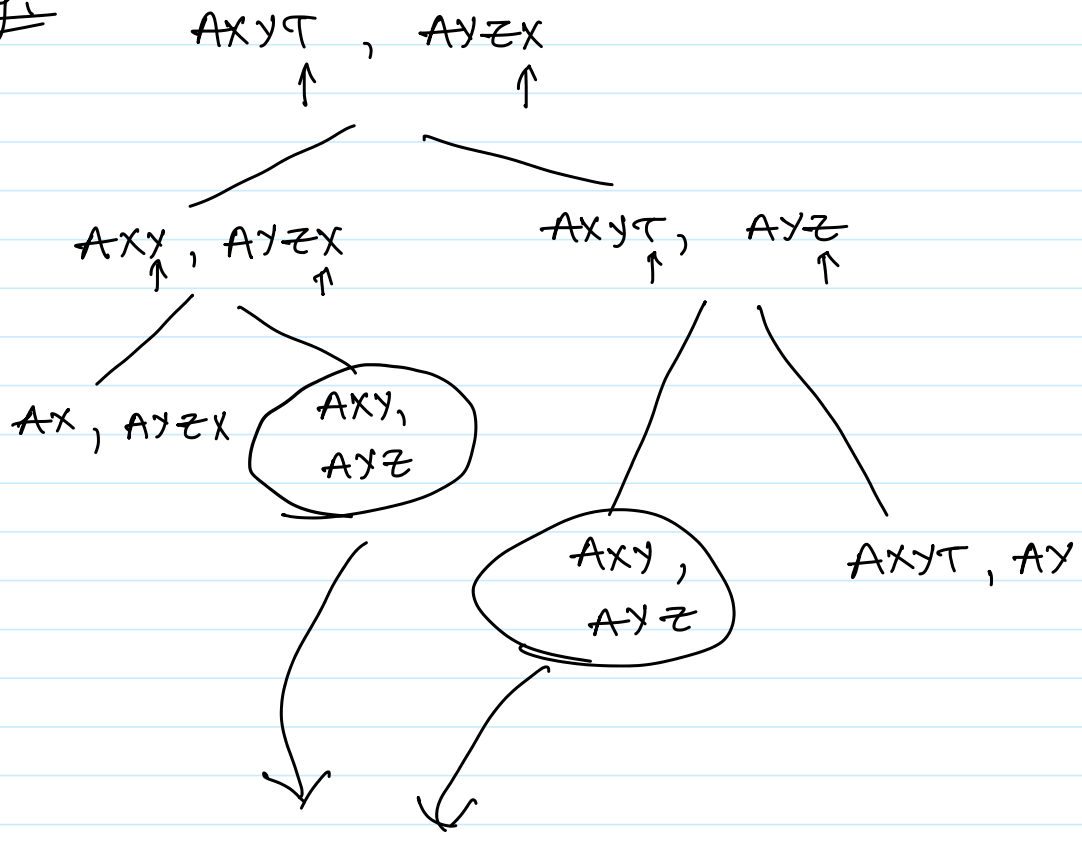
Recursion code:

```
if (xlen==0 || ylen==0)
        return 0;

if (x[xlen-1] == y[ylen-1])
        return 1+LCS(xlen-1, ylen-1);

else
        return max(LCS(xlen-1, ylen),
                   LCS(xlen, ylen-1));
```

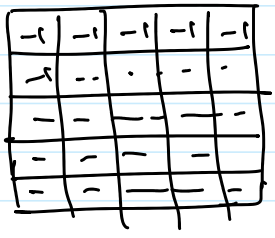
```
        LCS(xlen, ylen-1));
```

Recursion with memoization :-

Eg :-



→ We can see that there are repeating sub problems.
So, we'll use a table to store these values.
We can use a hashmap or/ matrix to store these.



m+1 x n+1 matrix

m+1, n+1 because we have to store from length = m to 0.

m+1, n+1 because we have to store from length = m to 0.
and length = n to 0.

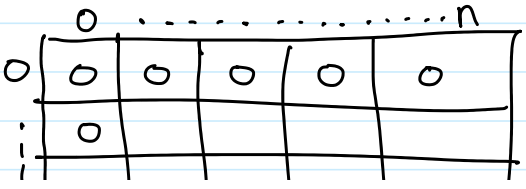
⟹ t[m+1][n+1]; Initialize all with -1

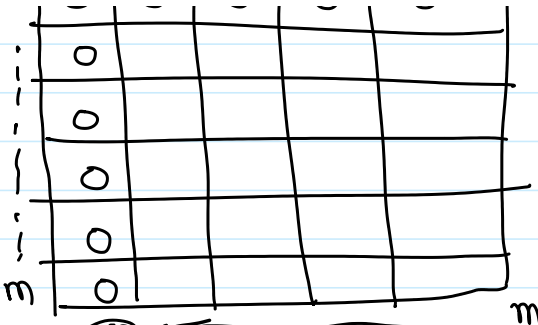
Recursion memoization Code :-

```
int LCS (X, Y, m, n)
{
    if (m == 0 || n == 0)
        return 0;

    if (t[m][n] != -1)
        return t[m][n];

    if (X[m-1] == Y[n-1])
        return t[m][n] = 1 + LCS (X, Y, m-1, n-1);

    else
        return t[m][n] = max (LCS (X, Y, m-1, n),
                              LCS (X, Y, m, n-1));
```

Top down DP approach :—

Following the above recursion we can see that —





m+1 x n+1 matrix

here, i, j is not index, it's the length

→ We initialized first row and first column because in recursion base condition we wrote

if (m==0 || n==0) return 0;

So, first column is n=0 and first row is m=0.

DP Code :-

→ Now, i from 1 to m, j from 1 to n

```
if ( X[i-1] == Y[j-1] )
        mat[i][j] = 1 + mat[i-1][j-1];
else
    mat[i][j] = max(mat[i-1][j],
                    mat[i][j-1])
```

(∵ same conditions as recursion)

```
return mat[m][n];
```

Problem Statement:

Vihaan is given a pair of words likely word1 and word2,
he is asked to create a method which returns the numbers of characters in a word
formed from long lasting frequent posteriority.

Posteriority is the word formed from the original word with few characters removed
without modifying the corresponding order of the left over characters.

Find the longest common posteriority of two words.
Return 0 if no common posteriority.

Input Format:

Two space separated strings S1, S2.

Output Format:

Print an integer, the length of longest common prosperity.

Sample Input-1:
---
abcde ace

Sample Output-1:

3

Sample Input-2:
---
acd bef

Sample Output-2:

0

Code:

```
import java.util.*;
class Posteriority
{
   public static void main(String args[])
   {
      Scanner sc=new Scanner(System.in);
      String word1=sc.next();
      String word2=sc.next();

      int l1=word1.length();
      int l2=word2.length();
      int dp[][]=new int[l1+1][l2+1];

      for(int i=0;i<l1+1;i++)
      {
         dp[i][0]=0;
      }
      for(int j=0;j<l2+1;j++)
      {
         dp[0][j]=0;
      }

      for(int i=1;i<l1+1;i++)
      {
         for(int j=1;j<l2+1;j++)
```

```
            {
              if(word1.charAt(i-1)==word2.charAt(j-1))
              {
                dp[i][j]=1+dp[i-1][j-1];
              }
              else
              {
                dp[i][j]=Math.max(dp[i-1][j],dp[i][j-1]);
              }
            }
          }

          System.out.println(dp[l1][l2]);
        }
      }
```

Test Cases:

```
case =1
input =abcdefghijklmno iamthebeststudentincollege
output =7
case =2
input =abcdefghijklmnopqrstuvwxyz hjkdhowiqjfmaknfeiuwpowrvnkljgopopqwjdknfjbdhjf
ewg
output =10
case =3
input =cucwxladpycawcvqeoshgmcmpxxvhmtszwxtlkomkdvdcaytzv tyxdbygmehbqeobnxydakrh
wztxmdacapiasnfifhlfsutgldhpglikzxbhgtfcjgmedqj
output =17
case =4
input =idckldaiieilfbdaeflhehfadghaldeikehkadeehejklglebf jcbaffgegcjflhgihfgifli
flhjcfbagcbkkjagfffkakfclhgbahdfakeihebala
output =20
case =5
input =abcde adobe
output =3
case =6
input =abcdefghijklmnopqrstuvwxyz zyxwvutsrqponmlkjihgfedcba
output =1
case =7
input =nagarjunaisahero cementfactorynamedasnagarjuncements
output =9
case =8
input =abcdefghijklm nopqrstuvwxyz
output =0
```