Maximum Sum Path in Two Arrays

Given two sorted arrays such the arrays may have some common elements. Find the sum of the maximum sum path to reach from beginning of any array to end of any of the two arrays. We can switch from one array to another array only at common elements.

Expected time complexity is O(m+n) where m is the number of elements in ar1[] and n is the number of elements in ar2[].

Examples:

```
Input:  ar1[] = {2, 3, 7, 10, 12}, ar2[] = {1, 5, 7, 8}
Output: 35
35 is sum of 1 + 5 + 7 + 10 + 12.
We start from first element of arr2 which is 1, then we
move to 5, then 7.  From 7, we switch to ar1 (7 is common)
and traverse 10 and 12.

Input:  ar1[] = {10, 12}, ar2 = {5, 7, 9}
Output: 22
22 is sum of 10 and 12.
Since there is no common element, we need to take all
elements from the array with more sum.

Input:  ar1[] = {2, 3, 7, 10, 12, 15, 30, 34}
        ar2[] = {1, 5, 7, 8, 10, 15, 16, 19}
Output: 122
122 is sum of 1, 5, 7, 8, 10, 12, 15, 30, 34
```

We strongly recommend to minimize the browser and try this yourself first.

The idea is to do something similar to merge process of merge sort. We need to calculate sums of elements between all common points for both arrays. Whenever we see a common point, we compare the two sums and add the maximum of two to the result. Following are detailed steps.

1) Initialize result as 0. Also initialize two variables sum1 and sum2 as 0. Here sum1 and sum2 are used to store sum of element in ar1[] and ar2[] respectively. These sums are between two common points.

2) Now run a loop to traverse elements of both arrays. While traversing compare current elements of ar1[] and ar2[].

2.a) If current element of ar1[] is smaller than current element of ar2[], then update sum1, else if current element of ar2[] is smaller, then update sum2.

2.b) If current element of ar1[] and ar2[] are same, then take the maximum of sum1 and sum2 and add it to the result. Also add the common element to the result.

Following is C++ implementation of above approach.

C++

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

// Utility function to find maximum of two integers
int max(int x, int y) { return (x > y)? x : y; }

// This function returns the sum of elements on maximum path
// from beginning to end
int maxPathSum(int ar1[], int ar2[], int m, int n)
{
    // initialize indexes for ar1[] and ar2[]
    int i = 0, j = 0;

    // Initialize result and current sum through ar1[] and ar2[].
    int  result = 0, sum1 = 0, sum2 = 0;

    // Below 3 loops are similar to merge in merge sort
    while (i < m && j < n)
    {
        // Add elements of ar1[] to sum1
        if (ar1[i] < ar2[j])
            sum1 += ar1[i++];

        // Add elements of ar2[] to sum2
        else if (ar1[i] > ar2[j])
            sum2 += ar2[j++];

        else  // we reached a common point
        {
            // Take the maximum of two sums and add to result
            result += max(sum1, sum2);

            // Update sum1 and sum2 for elements after this
            // intersection point
            sum1 = 0, sum2 = 0;

            // Keep updating result while there are more common
            // elements
            while (i < m &&  j < n && ar1[i] == ar2[j])
            {
                result = result + ar1[i++];
                j++;
            }
        }
    }

    // Add remaining elements of ar1[]
```

```
    while (i < m)
        sum1  +=  ar1[i++];

    // Add remaining elements of ar2[]
    while (j < n)
        sum2 +=  ar2[j++];

    // Add maximum of two sums of remaining elements
    result +=  max(sum1, sum2);

    return result;
}

// Driver program to test above function
int main()
{
    int ar1[]  = {2, 3, 7, 10, 12, 15, 30, 34};
    int ar2[] =  {1, 5, 7, 8, 10, 15, 16, 19};
    int m = sizeof(ar1)/sizeof(ar1[0]);
    int n = sizeof(ar2)/sizeof(ar2[0]);
    cout << "Maximum sum path is "
         << maxPathSum(ar1, ar2, m, n);
    return 0;
}
```

Run on IDE

Python

```
# Python program to find maximum sum path

# This function returns the sum of elements on maximum path from
# beginning to end
def maxPathSum(ar1,ar2 , m , n):

    # initialize indexes for ar1[] and ar2[]
    i, j = 0, 0

    # Initialize result and current sum through ar1[] and ar2[]
    result, sum1, sum2= 0, 0, 0

    # Below 3 loops are similar to merge in merge sort
    while (i < m and j < n):

        # Add elements of ar1[] to sum1
        if ar1[i] < ar2[j]:
            sum1 += ar1[i]
            i+=1

        # Add elements of ar2[] to sum1
        elif ar1[i] > ar2[j]:
            sum2 += ar2[j]
            j+=1

        else:   # we reached a common point

            # Take the maximum of two sums and add to result
            result+= max(sum1,sum2)

            # Update sum1 and sum2 for elements after this intersection point
            sum1, sum2 = 0, 0
```

```
            # Keep updating result while there are more common elements
            while (i < m and j < n and ar1[i]==ar2[j]):
                result += ar1[i]
                i+=1
                j+=1

    # Add remaining elements of ar1[]
    while i < m:
        sum1 += ar1[i]
        i+=1
    # Add remaining elements of b[]
    while j < n:
        sum2 += ar2[j]
        j+=1

    # Add maximum of two sums of remaining elements
    result += max(sum1,sum2)

    return result

# Driver function
ar1 = [2, 3, 7, 10, 12, 15, 30, 34]
ar2 = [1, 5, 7, 8, 10, 15, 16, 19]
m = len(ar1)
n = len(ar2)
print "Maximum sum path is", maxPathSum(ar1, ar2, m, n)

# This code is contributed by __Devesh Agrawal__
```

Run on IDE

Output:

```
Maximum sum path is 122
```

Time complexity: In every iteration of while loops, we process an element from either of the two arrays. There are total m + n elements. Therefore, time complexity is O(m+n).

This article is contributed by **Piyush Gupta**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

64 Comments Category: Arrays

Related Posts:

- Longest Span with same Sum in two Binary arrays
- Count Inversions of size three in a give array
- Find the subarray with least average
- Count triplets with sum smaller than a given value
- Find zeroes to be flipped so that number of consecutive 1's is maximized
- Reorder an array according to given indexes
- Find maximum value of Sum(i*arr[i]) with only rotations on given array allowed
- Find maximum average subarray of k length

(Login to Rate and Mark)

2.8 Average Difficulty : **2.8/5.0**
Based on **5** vote(s)

Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.