Largest Subarray Sum
(Kadane's algorithm)

by

Ashap-ud-doula Bappy

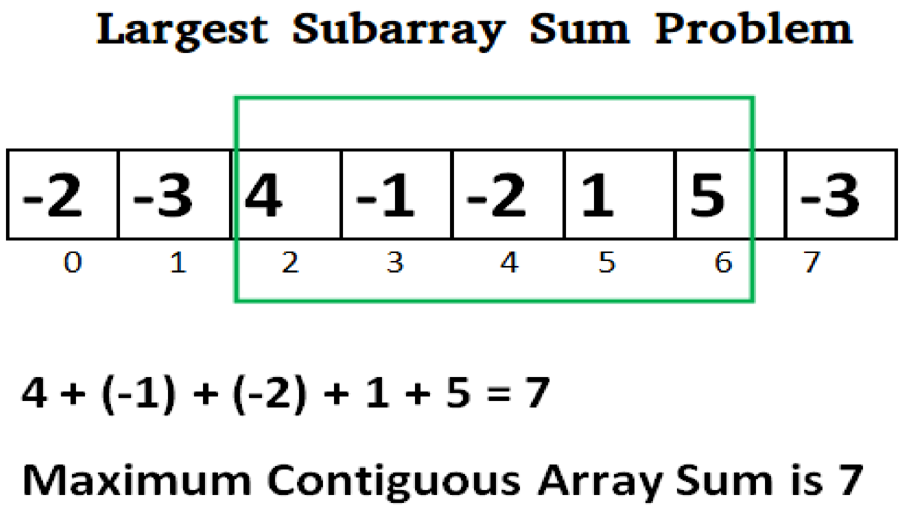
Introduction

Largest - stands for its own meaning.

Subarray - A subarray is a **contiguous** part of array. An array that is inside another array. For example, consider the array [1, 2, 3, 4], There are 10 non-empty sub-arrays. The subarrays are (1), (2), (3), (4), (1,2), (2,3), (3,4), (1,2,3), (2,3,4) and (1,2,3,4). In general, for an array/string of size n, there are **n*(n+1)/2** non-empty subarrays/substrings.

So, we are given a one-dimensional array, we have to find the sum of contiguous subarray within the array of numbers which has the largest sum.

For example,


Largest Subarray Sum Problem
-2
-3
4
-1
-2
1
5
-3
0
1
2
3
4
5
6
7
4 + (-1) + (-2) + 1 + 5 = 7
Maximum Contiguous Array Sum is 7

An O(n) approach

The idea is relatively easy. For this problem we will look for all **positive** (the sum of all the elements of this subarray will be positive) contiguous subarrays. Because whenever the subarray sum is negative, adding one more element to the subarray will only reduce the sum.

If we take the above array as example, the sum of the first two element is,

A [0] + A [1] = (-2) + (-3) = -5

Now if we add the **3rd** element which is **A [2] = 4** to the sum then the sum will be **(-5) + 4 = -1**

But if we start a new subarray from the 3rd element then the sum will be 4 which is greater than the previous sum which was obtained by taking the first three elements. So, it is better to start a new subarray from the 3rd element and ignore the sum of the first two elements.

See the code which prints the largest subarray sum:

```cpp
int main()
{
    //size of the array.
    int N = 8;

    int A[] = {-2, -3, 4, -1, -2, 1, 5, -3};

    int curSum = 0;
    int mxSum = INT_MIN;

    for(int i = 0; i < N; i++)
    {
        curSum += A[i]; // add current element to 'curSum'.

        // update the ‘mxSum’ to get the
        // maximum of all ‘curSum’ at every index.
        mxSum = max(curSum, mxSum);

        // start a new subarray from here,
        // because the previous sum was negative.
        if(curSum < 0)curSum = 0;

    }

    // output = Subarray sum from index (0 based index)
    // 2 to 6 = 4 + (-1) + (-2) + 1 + 5 = 7
    cout<< mxSum <<"\n";


    return 0;
}
```

Print indices of the subarray

Now you may want to print the indices of the subarray with maximum sum. The technique is quite easy. We just maintain the indices whenever we get the maximum sum and update the starting and ending indices.

See the code which prints indices of the subarray with the largest sum:

```
int main()
{
    //size of the array
    int N = 8;

    int A[] = {-2, -3, 4, -1, -2, 1, 5, -3};

    int curSum = 0;
    int mxSum = INT_MIN;
    int start = 0, end = 0, s = 0;

    for(int i = 0; i < N; i++)
    {
       // add current element to 'curSum'.
       curSum += A[i];

       // update the ‘mxSum’ to get the
       // maximum of all ‘curSum’ at every index.
       if(mxSum < curSum)
       {
          mxSum = curSum;
          start = s; // update the 'start' index with 's'.
          end = i;   // update the 'end' index with the current index 'i'.
       }

       // start a new subarray from here,
       // because the previous sum was negative.
       if(curSum < 0)
       {
          curSum = 0;

          // since curSum is negative,
          // start a new subarray from the next index 'i+1'.
          s = i+1;

       }
    }

    // output = Subarray sum from index (0 based index)
    // 2 to 6 = 4 + (-1) + (-2) + 1 + 5 = 7
    cout<< mxSum <<"\n";

    return 0;
}
```

Programming exercises

1. UVa 00787
2. UVa 10684
3. UVa 10755