Homework #2

Due date: 3/29 lab

Maximum Contiguous Subsequence

Given a sequence of signed numbers, determine the maximum sum found in any contiguous subsequence.
Also, identify such a maximum contiguous subsequence (mcs).

Comments

1 The maximum sum is at least 0, since an empty sequence is a contiguous subsequence and its sum is 0.
In particular, if all numbers are negative, an empty sequence is an mcs of the sequence.
2 In case there are more than one mcs's, find any one of them.
3 This is Problem 2008-13 of Collegiate Programming Exam.

Example

```
int a[13]
={38,-62,47,-33,28,13,-18,-46,8,21,12,-53,25};
```
`a[2..5]` is an mcs of `a` whose sum is 55.

```
double b[10]
={3.1,-4.1,5.9,2.6,-5.3,5.8,9.7,-9.3,-2.3,8.4};
```
`b[2..6]` is an mcs of `b` whose sum is 18.7.

```
short c[8]={1,2,3,-100,3,3,-20,6};
```
`c[0..2]`, `c[4..5]`, and `c[7..7]` are mcs's of `c` whose sums are all 6.

```
int d[9]={-1,-2,-3,-4,-5,-6,-7,-8,-9};
```
An empty sequence is an mcs of `d` whose sum is 0.

An empty sequence may be represented by any array in which the low-end index is greater than the high-end index, e.g.
`d[1..0]`, `d[4..3]`, `d[0..-1]`.

Digression – pair and tuple

For the purpose of this homework, let's first briefly introduce STL **pair** and **tuple** class templates. The **pair** class template is old, whereas the **tuple** class template is new in C++11.

Basically, a **pair** object contains exactly 2 elements, but a **tuple** object may contain n elements, for $n \geq 1$.

As far as this homework is concerned, the example below contains enough information for the use of these two classes.

Example

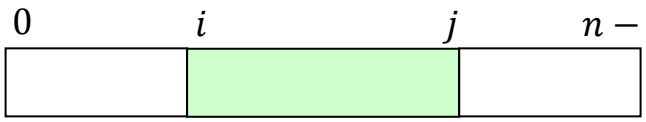
```
#include <iostream>
#include <utility>      // for pair
#include <tuple>        // for tuple
using namespace std;
int main()
{
   pair<int,char> a;
   tuple<int,char,double> b;
   a=make_pair(2,'a');                    // 1
   b=make_tuple(3,'b',4.5);               // 1
   cout << a.first << a.second;           // 2
   cout << get<0>(a) << get<1>(a);        // 3
   cout << get<0>(b) << get<1>(b) << get<2>(b); // 4
   get<1>(b)='c';                         // 5
   cout << get<0>(b) << get<1>(b) << get<2>(b);
}
```

Comments

1 **make_pair** creates a pair. **make_tuple** creates a tuple. Both are STL function templates.
2 **first** and **second** are two data members of **pair**.
3 Use tuple-like notation (see 4) to access the 2 elements **first** and **second** of pair **a**.
4 Access the 3 elements of tuple **b**, where **get** is an STL function template.
5 Modify the 1st element of tuple **b**.

Algorithm A – $O(n^2)$ enumeration algorithm

Find the maximum among the sums of all the subsequences **a[**$i..j$**]**



This algorithm is inefficient, since there are

$$\binom{n}{2} + n = \frac{n(n-1)}{2} + n = O(n^2) \quad \because 0 \le i \le j \le n$$

subsequences.

The following function implements this algorithm.

Input An array **a** of **n** elements

Output The out-mode parameter **maxsum** receives the sum of an mcs.The function value is a pair of array indices that identify an mcs. More precisely, let **p** be the returned pair, then **a[p.first..p.second]** is an mcs.
In case that an empty sequence is an mcs, the returned pair **p** is arbitrarily set to **p.first=1** and **p.second=0**

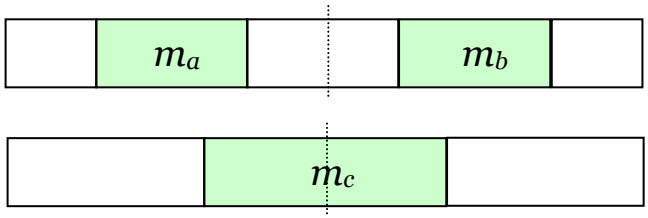
```
template<typename T>
pair<int,int> mcs(T* a,int n,T& maxsum)
{
   int left=1,right=0;
   maxsum=T(0);
   for (int i=0;i<n;i++) {
      T sum=T(0);
      for (int j=i;j<n;j++) {
         sum+=a[j];        // compute the sums incrementally
         if (sum>maxsum) {    // *
            maxsum=sum; left=i; right=j;
         }
      }
   }
   return make_pair(left,right);
}
```

Q: What would happen if in the starred line > is replaced by >=?
A: Find another mcs, if any.

As far as Collegiate Programming Exam is concerned, you may submit the preceding code. However, as a computer expert, you shall do better.

This homework asks you to implement two better algorithms, as described below.

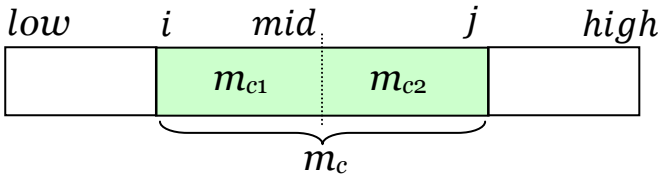
Algorithm B – $O(n \log n)$ divide-and-conquer algorithm



Let
$m_a =$ the maximum sum of the left-half subarray
$m_b =$ the maximum sum of the right-half subarray
$m_c =$ the maximum sum crossing the midpoint
Then,
max(m_a, m_b, m_c) is maximum sum of the whole array.

Hint

Compute m_c as $m_{c1} + m_{c2}$, where
m_{c1} = the maximum sum of subarrays of the form **a[**$i..mid$**]**
m_{c2} = the maximum sum of subarrays of the form **a[**$mid + 1..j$**]**



Q: Why is $m_c = m_{c1} + m_{c2}$?

Q: How to compute m_{c1}?
A: A loop suffices, since there are only $mid - low + 2$ subarrays, including an empty subarray, of the form **a[**$i..mid$**].**

You are asked to implement algorithm B in two ways:

```
// Version B1 – array as pointer
template<typename T>
pair<int,int> mcs(T* a,int n,T& maxsum);

// Version B2 – array as array
template<typename T,int n>
pair<int,int> mcs(T (&a)[n],T& maxsum);
```

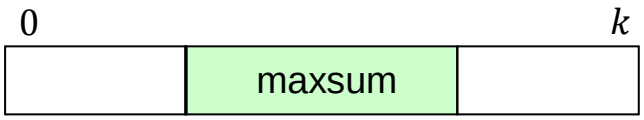
Hints

1 These two versions have similar code, as they implement the same algorithm.
2 Version B is a metaprogram. So, you need a more specialized function template to terminate the recursion.
3 Use **`reinterpret_cast`**
4 Run version B in VC++. Do NOT run it in GNU C++ (or Dev C++).

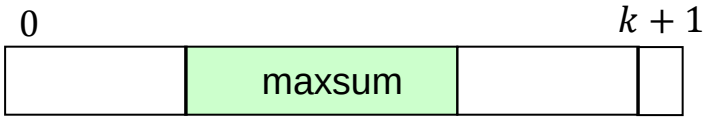
Algorithm C – $O(n)$ scanning algorithm

Given an array **`a[`**$0..n-1$**`]`**, scan its elements from left to right. After scanning **`a[`**$0..k$**`]`**, let

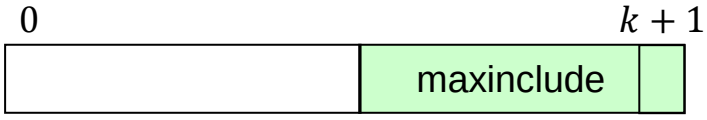
maxsum $=$ the maximum sum of **`a[`**$0..k$**`]`**



Now, consider the next element **`a[`**$k+1$**`]`**
A maximum sum subsequence of **`a[`**$0..k+1$**`]`** either excludes **`a[`**$k+1$**`]`** (whose sum is still maxsum)

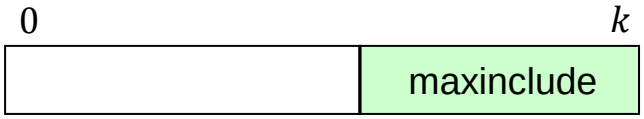


or includes **`a[`**$k+1$**`]`** (whose sum is maxinclude, as shown below)



Hint

Do NOT compute maxinclude of **a[**$0..k+1$**]** from scratch. Instead, compute it from maxinclude of **a[**$0..k$**]**



Write the following function template to implement algorithm C.

// Version C
```
template<typename T>
tuple<int,int,typename iterator_traits<T>::value_type>
mcs(T a,int n);
```

Let **p** be the tuple returned by this function, then
a[get<0>(p)..get<1>(p)] is an mcs, and
get<2>(p) is the maximum sum.

Comment

Given
```
int a[13],maxsum;
```
the call
mcs(a,13)
invokes an instance of version C, as desired. (Why?)

But, the call
mcs(a,maxsum)
is ambiguous, as it could be a call to an instance of version B2 or version C. (Why?)

Since version B2 is expected in this case, we may resolve this ambiguity by writing
mcs<int>(a,maxsum). (Why?)
This indeed will invoke an instance of version B2 in both VC++ and GNU C++.

Surprisingly, if we are given
```
double b[10],maxsum;
```
and do exactly the same thing by calling
mcs<double>(a,maxsum),

the overload resolution fails to select version B2 in VC++. In fact, it doesn't compile at all in VC++.
On the other hand, GNU C++ still correctly selects version B2. But, version B2 doen't work in GNU C++.

Now that both compilers have bugs, what shall we do?

Well, we may give up overloading! 

So, let's sadly rename version C as

```
template<typename T>
tuple<int,int,typename iterator_traits<T>::value_type>
mcs_scan(T a,int n);
```

Sample test

Suffice it to run the sample test given in file hw2test.cpp.

Sample output

```
Testing algorithm B1
55 : a[2..5] is an mcs.
18.7 : b[2..6] is an mcs.
6 : c[4..5] is an mcs.
0 : An empty sequence is an mcs.

Testing algorithm B2: Metaprogram
55 : a[2..5] is an mcs.
18.7 : b[2..6] is an mcs.
6 : c[4..5] is an mcs.
0 : An empty sequence is an mcs.

Testing algorithm C
55 : a[2..5] is an mcs.
18.7 : b[2..6] is an mcs.
6 : c[0..2] is an mcs.
0 : An empty sequence is an mcs.
```