Maximal Segments of Bounded Variation and Bounded Descent

Wim H. Hesselink
Dept. of Computing Science, University of Groningen
P.O.Box 407, 9700 AK Groningen, The Netherlands
Email: w.h.hesselink@rug.nl

To Jan van Leeuwen, for his 65th birthday

1 Introduction

Arnold Meijster, who wanted to recognize lines of characters in old manuscripts, posed the following problem. Given an array of real numbers and an upper bound, determine the maximal segments with variation below the upper bound, in an efficient way. Here, the variation of a segment is the difference between the maximum and the minimum value on the segment.

It turns out that there is a linear-time algorithm for this problem, and that this algorithm uses the slightly unusual data structure of a two-sided queue. Actually, it uses two such two-sided queues. This suggested a second algorithm that only uses one two-sided queue to determine the maximal segments of bounded descent. We derive the first algorithm in section 2, and present the second algorithm in section 3.

To fix the notation, let x be the array, let N be the length of x, and let B be the upper bound, according to the declaration:

const $N \in \mathbb{N}$, $B \in \mathbb{R}$ $\{\ N \geq 1 \wedge B \geq 0\ \}$;
const $x \in$ **array**$[N]$ **of** $\mathbb{R}$.

A segment is a subset of $\mathbb{N}$ of the form $s(p,q) = \{j \mid p \leq j < q\}$ with p, $q \in \mathbb{N}$ and $0 \leq p < q \leq N$.

2 Maximal Segments of Bounded Variation

The *variation* of a segment S is the difference $\mathrm{var}_S(x) = \max_S(x) - \min_S(x)$, where $\max_S(x) = \mathrm{Max}\{x[j] \mid j \in S\}$, and $\min_S(x)$ is defined similarly. We define segment S to be *BV-maximal* if $\mathrm{var}_S(x) \leq B$, and $S = T$ for every segment $T \supseteq S$ with $\mathrm{var}_T(x) \leq B$. The aim is to determine all BV-maximal segments.

Note that arbitrarily many BV-maximal segments can overlap. For instance, let $r \in \mathbb{N}_+$, and let $x[i] = B \cdot i/r$ for all i. Then $s(k, k+r+1)$ is BV-maximal for all $k < N - r$, and generally $r+1$ of these segments overlap.

We propose to scan the array from left to right (from 0 to N), and to collect the BV-maximal segments along the way. We therefore define $BVS(n)$ to be the

set of the pairs (p, q) such that $s(p, q)$ is a BV-maximal segment of $s(0, n)$. The aim is to determine $BVS(N)$.

We thus introduce variables

$$\textbf{var } n : \mathbb{N} \ ;$$
$$\textbf{var } bs : \textbf{set of } \mathbb{N}^2 \ ;$$

and aim to establish the postcondition $bs = BVS(N)$. This suggests the loop invariant $n \leq N \ \wedge \ bs = BVS(n)$.

If $(p, q) \in BVS(n)$ and $p < q < n$, then $(p, q) \in BVS(n + 1)$. There is always precisely one p with $(p, n) \in BVS(n)$, but this pair need not be in $BVS(n + 1)$ because extension of $s(p, n)$ to the right may be possible. We don't want to remove pairs from bs. We therefore abandon the suggested invariant, and instead introduce an integer variable k, and postulate the loop invariant

J0: $\quad k < n \leq N \ \wedge \ \{(k, n)\} \cup bs = BVS(n) \ \wedge \ (k, n) \notin bs \ .$

This invariant is easily initialized. We thus get the following program to establish the postcondition $BVS(N) = bs$.

(0)
$k := 0 \ ; \ n := 1 \ ; \ bs := \emptyset \ ;$
while $n < N$ **do**
  if $(k, n + 1) \notin BVS(n + 1)$ **then**
    add (k, n) *to* bs ;
    determine k *with* $(k, n + 1) \in BVS(n + 1)$ **fi** ;
  $n := n + 1$;
od ;
add (k, n) **to** bs .

This program uses so-called active finalization because the postcondition aimed at is not implied by the conjunction of the invariant and the negation of the guard, but is established from the postcondition of the loop by a simple assignment.

We now need additional data to remove the set BVS from the body of the loop. In order to decide $(k, n + 1) \in BVS(n + 1)$, we have to compare $\max_S(x)$ and $\min_S(x)$ for $S = s(k, n + 1)$. If $(k, n + 1) \notin BVS(n + 1)$, however, we have to determine a new value for k, and for this we have to compare local maxima and minima of array x that are more to the right than the old k. For each of these local extrema, we need the rightmost occurrence.

We therefore introduce

$$\textbf{var } p, q, r, s : \mathbb{N} \ ;$$
$$\textbf{var } mx, mn : \textbf{array}[N] \textbf{ of } \mathbb{N} \ .$$

The elements of $mx[p : q]$ serve as the indices of the rightmost local maxima of x on the segment $s(k, n)$ according to the invariant

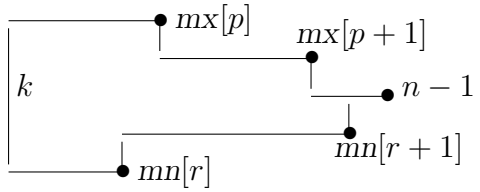
$$
\begin{array}{ll}
\textit{J1:} & p \le q < n \ \wedge \ k \le mx[p] \ \wedge \ mx[q] = n-1 \\
& \wedge \ (\forall\, i : p \le i < q \ \Rightarrow \ mx[i] < mx[i+1]) \\
& \wedge \ (\forall\, j : k \le j < n \ \Rightarrow \ x[mx[p]] \ge x[j]) \\
& \wedge \ (\forall\, i, j : p \le i < q \ \wedge \ mx[i] < j < n \\
& \qquad \Rightarrow \ x[mx[i]] > x[j] \ \wedge \ x[mx[i+1]] \ge x[j]) \ .
\end{array}
$$

Here, the first two lines express that $mx[p : q]$ is a subsequence of the ordered sequence of elements of $s(k, n)$ that ends in $n-1$. The third line says that $mx[p]$ is an index where the maximum of x on $s(k, n)$ is attained. The fourth line expresses that each $mx[i]$ is the rightmost index of a local maximum for a new stretch.

Similarly, the elements of $mn[r : s]$ serve as the indices of the rightmost local minima of x on the segment $s(k, n)$ according to the invariant

$$
\begin{array}{ll}
\textit{J2:} & r \le s < n \ \wedge \ k \le mn[r] \ \wedge \ mn[s] = n-1 \\
& \wedge \ (\forall\, i : r \le i < s \ \Rightarrow \ mn[i] < mn[i+1]) \\
& \wedge \ (\forall\, j : k \le j < n \ \Rightarrow \ x[mn[r]] \le x[j]) \\
& \wedge \ (\forall\, i, j : r \le i < s \ \wedge \ mn[i] < j < n \\
& \qquad \Rightarrow \ x[mn[i]] < x[j] \ \wedge \ x[mn[i+1]] \le x[j]) \ .
\end{array}
$$

In the diagram we sketch the meaning of these two invariants as bounding the graph of $x[k : n-1]$ by rectangles. The bullets represent points $(j, x[j])$ with $j = mx[i]$ and $j = mn[i]$. The rightmost bullet has $j = n-1$.



The invariants *J1* and *J2* are easily initialized by

$$
\begin{array}{l}
p := q := r := s := 0 \ ; \\
mx[0] := mn[0] := 0 \ .
\end{array}
$$

Now assume that $J1 \wedge J2$ holds at the start of the loop body. The incrementation $n := n+1$ asks for inclusion of n into the sequences mx and mn, and accommodation of $x[n]$ in the inequalities. We therefore propose commands *rightMx* and *rightMn* that satisfy the Hoare triples

$$
\begin{array}{llll}
\{\ J1\ \} & \textit{rightMx} & \{\ J1[n := n+1]\ \} & , \\
\{\ J2\ \} & \textit{rightMn} & \{\ J2[n := n+1]\ \} & .
\end{array}
$$

In either case, first the superfluous part at the righthand side of the sequence mx or mn is removed. Subsequently the number n is added to the sequence.

rightMx:
while $p \leq q \ \wedge \ x[mx[q]] \leq x[n]$ **do** $q := q - 1$ **od** ;
$q := q + 1$; $mx[q] := n$.

rightMn:
while $r \leq s \ \wedge \ x[mn[s]] \geq x[n]$ **do** $s := s - 1$ **od** ;
$s := s + 1$; $mn[s] := n$.

After *rightMx* and *rightMn*, the extrema of x on $s(k, n+1)$ are at the indices $mx[p]$ and $mn[r]$. We therefore have

$$(k, n+1) \in BVS(n+1) \quad \equiv \quad mx[p] \leq mn[r] + B \ .$$

If $(k, n+1) \notin BVS(n+1)$, we have to modify k. In this case, $x[n]$ is one of the two extrema because $(k, n) \in BVS(n)$. Therefore, $p = q$ or $r = s$. Now, from the left, the maxima above $x[n] + B$ have to be discarded, as well as the minima below $x[n] - B$. Moreover, index k should be moved to the right of the rightmost discarded extremum. This is done in the loops:

leftMx:
while $x[mx[p]] > x[n] + B$ **do**
$k := mx[p] + 1$; $p := p + 1$;
od .

leftMn:
while $x[mn[r]] < x[n] - B$ **do**
$k := mn[r] + 1$; $r := r + 1$;
od .

At this point, we have $mx[q] = n = mn[s]$, so that the loops necessarily terminate because $B \geq 0$. Moreover, one of the loops stops immediately because of $p = q$ or $r = s$.

The complete program to determine the BV-maximal segments becomes

(1)
$k := 0$; $n := 1$; $bs := \emptyset$;
$p := q := r := s := 0$;
$mx[0] := mn[0] := 0$;
while $n < N$ **do**
rightMx ; *rightMn* ;
if $x[mx[p]] > x[mn[r]] + B$ **then**
add (k, n) *to* bs ;
leftMx ; *leftMn* **fi** ;
$n := n + 1$;
od ;
add (k, n) **to** bs .

The complexity of this program is linear in N. This is not completely obvious because of the four inner loops in the body of the outer loop. It is shown by means of the variant function $vf = 3(N-n)+(q-p)+(s-r)$. Indeed, $vf \geq 0$ at the start of each of the loops. We have initially $vf = 3(N-1)$. The value of vf decreases in every step of any of the inner loops. It also decreases with 1 in every execution of the body of the outer loop, outside of the inner loops, because of the assignments $n := n+1$, $q := q+1$, $s := s+1$. Therefore, the time complexity of the program is $\mathcal{O}(N)$.

The body of the main loop in (1) can be replaced by the following code:

if $x[mx[p]] > x[n] + B$ **then**
  add (k, n) *to* bs ;
  $q := q + 1$; $mx[q] := n$; *leftMx* ;
  $s := r$; $mn[r] := n$;
elsif $x[mn[r]] < x[n] - B$ **then**
  add (k, n) *to* bs ;
  $s := s + 1$; $mn[s] := n$; *leftMn* ;
  $q := p$; $mx[p] := n$;
else
  rightMx ; *rightMn*
fi ;
$n := n + 1$.

This is slightly more efficient than (1), because in the first two branches it makes one of the queues empty in a single step, while the same is done in a loop in (1).

3 Bounded Descent

The *descent* of a segment S of array x is defined as

$$\mathrm{desc}_S(x) = \mathrm{Max}\{x[i] - x[j] \mid i, j \in S : i < j\} \ .$$

The descent of S is negative if and only if x is increasing on the segment. In this case, however, the descent equals the maximum of the differences $x[i] - x[i+1]$ over the segment, so that special methods apply. On the other hand, above we have used the assumption $B \geq 0$ implicitly several times. We therefore retain this assumption.

Segment S is said to be of *descent bounded by* B if $\mathrm{desc}_S(x) \leq B$, or equivalently if $x[i] \leq x[j]+B$ for all $i, j \in S$ with $i < j$. Segment S is called *BD-maximal* if $\mathrm{desc}_S(x) \leq B$ and if $S = T$ for every segment $T \supseteq S$ with $\mathrm{desc}_T(x) \leq B$.

We now describe an algorithm to determine the set of all BD-maximal segments of array x. This algorithm closely resembles the algorithm for the BV-maximal segments.

Just as before, we define $BDS(n)$ to be the set of BD-maximal segments of $s(0,n)$. We can retain invariant *J0* and algorithm (0), when we replace *BVS* by *BDS*. In view of the inequalities $x[i] \leq x[j] + B$ for $i < j$, we need only to remember the (local) *maxima* of the past. We therefore only introduce the auxiliary variables p, q, mx, and the invariant *J1*. The resulting algorithm to determine the BD-maximal segments is

(2)
$k := 0$; $n := 1$; $bs := \emptyset$;
$p := q := 0$; $mx[0] := 0$;
while $n < N$ **do**
  rightMx ;
  if $x[mx[p]] > x[n] + B$ **then**
    add (k,n) *to* bs ;
    leftMx **fi** ;
  $n := n + 1$;
od ;
add (k,n) **to** bs .

Of course, the complexity of (2) is again linear.

For the sake of testing this algorithm, we needed an algorithm to determine the descent of a segment $s(k,n)$, given by

$$\mathrm{desc}_{s(k,n)}(x) = \mathrm{Max}\{x[i] - x[j] \mid i,j : k \leq i < j < n\} \ .$$

It is a good exercise for first-year students computer science to design a program to compute this in linear time.

In conclusion: I did not know these algorithms but I cannot claim they are new. It is a nice occasion to write them down as a birthday present for Jan van Leeuwen.