ERRATA IW 121/79

The omission of $\mathrm{INFO}(\nu)[5]$, a boolean variable to be maintained at each node ν of the connected interval tree, necessitates the changes below.

page line

7 +1 $\mathrm{INFO}(\nu)[1:4] \rightarrow \mathrm{INFO}(\nu)[1:5]$

7 +18 Insert:

$\mathrm{INFO}(\nu)[5]$: a *boolean* which is true if each leaf interval of the subtree rooted at ν is covered by an alive line segment *which does not* cover all of the ν-interval, and is false otherwise.

7 +19 $\mathrm{INFO}(\nu)[3] = \rightarrow \mathrm{INFO}(\nu)[3] = \mathrm{INFO}(\nu)[5] =$

7 +26 $\mathrm{INFO}(\mathrm{root})[1] \neq 0 \rightarrow \mathrm{INFO}(\mathrm{root})[1] \neq 0$ or $\mathrm{INFO}(\mathrm{root})[5]$

8-9 add a 5-th element "f" to each 4-tuple in the figures.

11 +8:+15 change to:

```
Leftcond := $\mathrm{INFO}(\nu_\ell)[1] \geq 1$ or $\mathrm{INFO}(\nu_\ell)[5]$;
rightcond := $\mathrm{INFO}(\nu_r)[1] \geq 1$ or $\mathrm{INFO}(\nu_r)[5]$;
if leftcond and rightcond
then $\mathrm{INFO}(\nu)[4] := 0$; $\mathrm{INFO}(\nu)[5] :=$ true
else if leftcond then $\mathrm{INFO}(\nu)[4] := \mathrm{INFO}(\nu_r)[4]$
     else if rightcond then $\mathrm{INFO}(\nu)[4] := \mathrm{INFO}(\nu_\ell)[4]$
```

-6 $\mathrm{INFO}(\nu)[4] \rightarrow \mathrm{INFO}(\nu)[4:5]$

12 +9:+13 change to:

```
Leftcond := $\mathrm{INFO}(\nu_\ell)[1] \geq 1$ or $\mathrm{INFO}(\nu_\ell)[5]$;
rightcond := $\mathrm{INFO}(\nu_r)[1] \geq 1$ or $\mathrm{INFO}(\nu_r)[5]$;
if leftcond and rightcond then
else $\mathrm{INFO}(\nu)[5] :=$ false;
    if leftcond then $\mathrm{INFO}(\nu)[4] := \mathrm{INFO}(\nu_r)[4]$
    else if rightcond then $\mathrm{INFO}(\nu)[4] := \mathrm{INFO}(\nu_\ell)[4]$
```

-10 $\mathrm{INFO}(\nu)[2:4] \rightarrow \mathrm{INFO}(\nu)[2:5]$

-5 append: or $\mathrm{INFO}(\mathrm{root})[5]$

15 -3 and $\mathrm{INFO}(\nu)[3] \rightarrow$,$\mathrm{INFO}(\nu)[3]$ and $\mathrm{INFO}(\nu)[5]$.

stichting
mathematisch
centrum

MC

AFDELING INFORMATICA
(DEPARTMENT OF COMPUTER SCIENCE)

IW 121/79 NOVEMBER

P.M.B. VITÁNYI & D. WOOD

COMPUTING THE PERIMETER OF A SET OF RECTANGLES

Preprint

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

1980 Mathematics Subject Classification: 68B15, 68C25, 28.04,
28A75, 26B15, 51M25

ACM-Computer Reviews-Categories: 5.25, 5.39, 4.34, 3.71

Computing the perimeter of a set of rectangles*)

by

Paul M.B. Vitányi**) & Derick Wood***)

ABSTRACT

We describe an algorithm for computing the perimeter (sum of lengths of boundaries) of the area in the plane covered by a given set of n rectilinearly-oriented rectangles. With some modifications the algorithm also computes the measure (surface) of this area. For the latter task such an algorithm was available before. Our main thrust shall be a comparison of the worst-case performances of the algorithms under various computational assumptions. The results strengthen the conjecture that $\Theta(n)$ space and $\Theta(n \log n)$ time simultaneously is unattainable for the perimeter and measure problems when a realistic model of computation is assumed. The algorithms generalize easily to higher dimensions. Without substantially altering the time/storage requirements, the perimeter algorithm can be adapted to output the boundary of a set of intersecting rectangles (or intervals in d-space), which may be useful in computer graphics.

KEY WORDS & PHRASES: *Geometric complexity, algorithms and data structures, intersecting rectilinearly-oriented rectangles, intersecting intervals in* d*-space, perimeter, measure, boundary.*

---

*) The work of the second author supported in part by a Natural Sciences and Engineering Research Council of Canada Grant No. A 7700 and in part by a grant of the Netherlands Organization for the Advancement of Pure Research (Z.W.O.)

**) Mathematical Centre, 2e Boerhaavestraat 49,
Amsterdam, The Netherlands

***) Unit for Computer Science, McMaster University,
Hamilton, Ontario L8S 4K1, Canada

This report will be submitted for publication elsewhere.

1. INTRODUCTION

Consider the problem of computing the perimeter of a set of n rectilinearly-oriented rectangles in the plane. That is, we want to determine the total length of the boundaries delineating the total area covered by the rectangles. This is the 2-dimensional particularization of a problem which we shall call the *perimeter problem*. More formally, the perimeter problem is stated as follows. A *closed interval* (generalized rectangular parallelepiped) in d-space consists of all points $x = (x_1,x_2,\ldots,x_d)$ such that $\ell_i \leq x_i \leq u_i$ $(1 \leq i \leq d)$ for some fixed numbers $\ell_1,\ell_2,\ldots,\ell_d$ and $u_1,u_2.,,,u_d$ with $\ell_i < u_i$ for all i. Given n closed intervals in d-space, say $A_1,A_2,\ldots,A_n$, the perimeter problem asks for an efficient algorithm to compute the (d-1)-dimensional *measure* of the *boundary* of $\bigcup_{i=1}^{n} A_i$. The boundary of $\bigcup_{i=1}^{n} A_i$ is defined as the intersection of $\bigcup_{i=1}^{n} A_i$ and the closure of its complement in d-space. Therefore, the boundary will consist of pieces of (d-1)-dimensional hyperplanes, and the *perimeter* is the sum of the (d-1)-dimensional measures of these pieces. For d = 1 the perimeter problem consists in computing the number of *end points* in $\bigcup_{i=1}^{n} A_i$, i.e., twice the number of disjoint closed intervals making up $\bigcup_{i=1}^{n} A_i$.

For d = 2 the perimeter problem consists in finding the sum of the lengths of the linesegments which form the boundary of $\bigcup_{i=1}^{n} A_i$, where the A's are rectangles in the plane. We shall present an algorithm for solving the 2-dimensional perimeter problem and analyze its complexity. Under certain cost measures this algorithm is optimal. Virtually the same algorithm can be used to solve the case d = 1, and then yields running times of the same order of magnitude as does the case d = 2. Presumably, this is optimal for d = 1. Generalizations of the algorithm to the cases $d \geq 3$ do not seem to be optimal.

Problems like the perimeter problem belong to the general area of *geometric complexity*. A closely related problem was proposed by KLEE [1977] who called it the *measure problem*. The measure problem asks us to design efficient algorithms to compute the d-dimensional measure of the union of n intervals $A_1,A_2,\ldots,A_n$ in d-space, d = 1,2,... . It was soon found, that for the 1-dimensional case there was an algorithm with a worst case running time of $\Theta(n \log n)$, which was proved to be optimal.

BENTLEY [1977] designed a very efficient algorithm to solve the 2-dimensional case, viz. that of finding the total area in the plane covered by a set of rectangles. For this purpose he introduced a data-structure called the *segment tree*. The algorithm runs in $\Theta(n \log n)$ time, i.e., the same as the optimal algorithm for the 1-dimensional case, and is therefore optimal too. The generalization of this algorithm to the d-dimensional case runs in time $\mathcal{O}(n^{d-1} \log n)$, $d \geq 2$. The algorithm is also described in VAN LEEUWEN and WOOD [1979]. In the latter paper BENTLEY's [1977] result also is improved for dimensions greater or equal to 3, by the exhibition of an algorithm which solves the measure problem in d dimensions in time $\mathcal{O}(n^{d-1})$, $d \geq 3$. (This is achieved by using as underlying data structure not the segment tree, but the *quad tree* due to FINKEL and BENTLEY [1974].) Another application of the segment tree has been in algorithms for reporting all pairs of intersecting rectangles from a given set of rectangles in the plane , see e.g. BENTLEY and WOOD [1979].

In the solution to the perimeter problem we give in section 2, the segment tree will once again prove its value as underlying data structure for algorithms solving problems connected with sets of intersecting rectangles. For completeness sake, we show in section 3 how to modify the given algorithm to obtain a variant of BENTLEY's [1977] algorithm for solving the measure problem in 2-space.

Our main thrust shall be to analyze the performances of the algorithms, both for the perimeter and the measure problem, under various models of computation for the representation and manipulation of numbers. It shall be shown, that under reasonable assumptions, such as corresponding to actual computer implementations of the algorithms, $\mathcal{O}(n \log n)$ running time implies $\Omega(n \log n)$ space in the worst case for the measure problem, and seems impossible for the perimeter problem.

However, it will also appear that if we allow slightly more time, i.e. $\mathcal{O}(n \log^2 n)$, then both problems can be solved in $\Theta(n \log \log n)$ space in the worst case. On a RAM with uniform cost criterion both algorithms run in $\Theta(n \log n)$ time and $\Theta(n)$ space.

We conjecture that an algorithm to solve either problem in simultaneous $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ space does not exist under "reasonable" models of computation. What is reasonable here shall be argued in sections 4 and 5.

In section 5 we shall note that the generalization of the perimeter algorithm to d dimensions runs in time $\mathcal{O}(d\, n^{d-1} \log n)$ and that this may be improved to $\mathcal{O}(d\, n^{d-1})$ for $d \geq 3$. How optimal this is (for $d \geq 3$) we do not know. The presented algorithms for the 2-dimensional case are optimal under the uniform (constant) cost criterion.

Note that from the results one gets the feeling that the perimeter problem is more difficult than the measure problem, but the evidence is not conclusive.

Problems connected with large sets of intersecting rectangles arise in many applications. For instance, the determination of all pairs of intersecting rectangles in the plane is a problem arising in maintaining architectural data bases and forms a crucial step in design rule checking for Very Large Scale Integrated circuity (VLSI), see BENTLEY and WOOD [1979]. An application in computer graphics might be to determine the boundary of a set of rectangles. In section 5 we indicate how to adapt the perimeter algorithm to do so.

2. THE PERIMETER PROBLEM

Suppose we are given n rectilinearly-oriented rectangles $A_1, A_2, \ldots, A_n$ (in 2-space) represented as

$$A_i = [\ell_{i1} : u_{i1};\ \ell_{i2} : u_{i2}], \quad 1 \leq i \leq n,$$

where ℓ = "low" and u = "high" (or "upper"). For the area or *measure problem* we wish to compute the measure of the total area in the plane covered by the rectangles. For the *perimeter problem* we wish to compute the sum of the lenghts of the boundaries delineating this area. An approach to a solution of such problems is usually based on the scanning technique. By scanning the set of rectangles from left-to-right in, say, the x-direction, and keeping track of appropriate information about the rectangles currently being scanned, the perimeter or measure is accumulated. Such an approach only requires the scanning of the 2n endpoints of the rectangles, since these are the points in the scan at which changes occur. In Figure 1 seven rectangles are displayed together with the corresponding scan lines $s_1, s_2, \ldots, s_{14}$. To compute the perimeter of the rectangles we first compute the contribution in the x-direction and secondly in the y-direction.

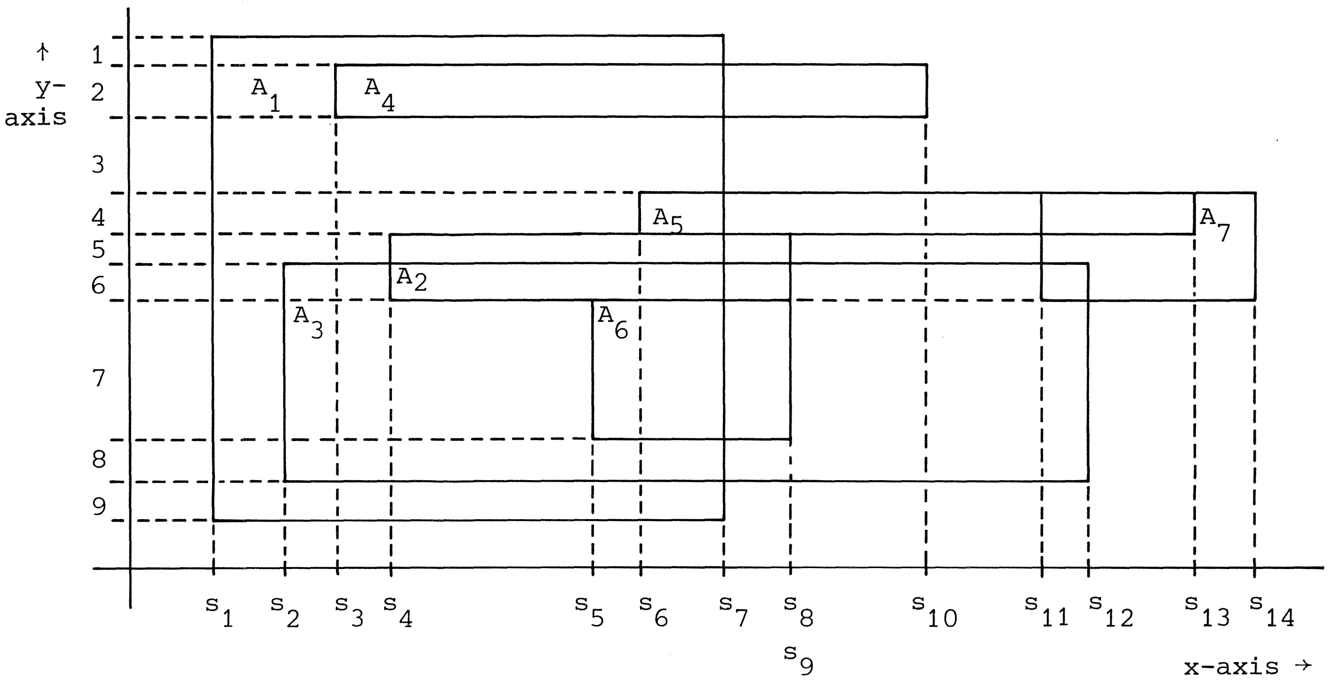


Figure 1.

The contribution P_x in the x-direction is the sum of the 2n-1 partial contributions in the x-direction defined by the 2n scan lines. These, in turn, are simply twice the number of "maximal connected intervals" at the first one of a pair of consecutive scan lines multiplied by the distance in between these scan lines. More precisely:

DEFINITION 1: I is a *maximal connected interval*, or *m.c.i.*, at scan line s if:

(i) I is a connected interval of s;

(ii) I is contained by $\bigcup_{i=1}^{n} A_i$;

(iii) I is maximal, i.e., I is not contained by any superinterval I' for which (i) and (ii) hold;

(iv) There is a number $\varepsilon > 0$ such that (i)-(iii) also hold with s substituted by s_δ, a line parallel to s but δ further away from the origin, for each δ, $0 \le \delta \le \varepsilon$.

Condition (iv) is needed to ensure that upper boundaries of rectangles at s do not contribute to m.c.i.'s at s.

Let $d_1^{(i)}, d_2^{(i)}, \ldots, d_{n_i}^{(i)}$ be the m.c.i.'s at scan line s_i for $1 \leq i \leq 2n$. Then the perimeter of the n rectangles is given by $P_x + P_y$ where P_x is the portion of the perimeter parallel to the x-axis and P_y is the portion of the perimeter parallel to the y-axis. We can compute P_x as

$$(1) \qquad P_x = 2 \sum_{i=1}^{2n-1} n_i (s_{i+1} - s_i).$$

P_y is computed similarly. The surface of the area (or the measure) defined by the n rectangles is given by

$$(2) \qquad M = \sum_{i=1}^{2n-1} (s_{i+1} - s_i)(\sum_{j=1}^{n_i} \text{length } (d_j^{(i)})),$$

where length (d) is the length (or 1-dimensional measure) of d. Note that when two scan lines s_i and s_{i+1} are coincident then $s_{i+1} - s_i = 0$ and hence no contribution to the perimeter or measure is involved at step i.

Thus, to compute the perimeter of the set of rectangles we carry out two scans of the figure, while to compute the measure we scan the figure but once. (2) demonstrates that, to compute the two-dimensional measure, we need the one-dimensional measure in the y-direction at each scan line (perpendicular to the x-axis.) This fact forms also the basis for the algorithm to compute the measure as given by BENTLEY [1977] (see also VAN LEEUWEN and WOOD [1979]).

With each scan line s_i we need to associate

(i) its x-coordinate (tacitly assumed to be identical with s_i);

(ii) whether it corresponds to a left or right (lower or higher) end of a rectangle; and

(iii) the interval or line segment in the y-direction the rectangle concerned defines.

To be able to scan the figure in a left-to-right manner we first need to sort the scan lines by their x-coordinates. To compute P_x as given by (1) we then move through the scan lines in the sorted order while computing at step i of this procedure the number of maximal connected intervals at scan line s_i. This we are able to do by using a suitable modification of the segment tree (BENTLEY [1977], BENTLEY and WOOD [1979]) which we shall call the *connected interval tree.* The major distinction between the segment tree and the connected interval tree is that identification of the inserted and deleted line segments is not required in the latter case while it is required in the former case.

Let us now describe the connected interval tree, its construction and manipulation. In Figure 1 7 rectangles are displayed which give rise to 9 line fragments in the y-direction, according to the division of the covered part of the y-axis by the y-coordinates of the 7 rectangles. Thus, the projection of each rectangle on the y-axis is made up of a contiguous subset of these fragments. For example, the projection of A_3 on the y-axis is made up of the line fragments numbered 6, 7 and 8. We now construct a minimal height binary tree with as many leaves as there are fragments in the y-direction induced by the rectangles. In our example we construct a minimal binary tree with 9 leaves representing the fragments 1-9 in left-to-right order as displayed in Figure 2: the *skeletal* connected interval tree.

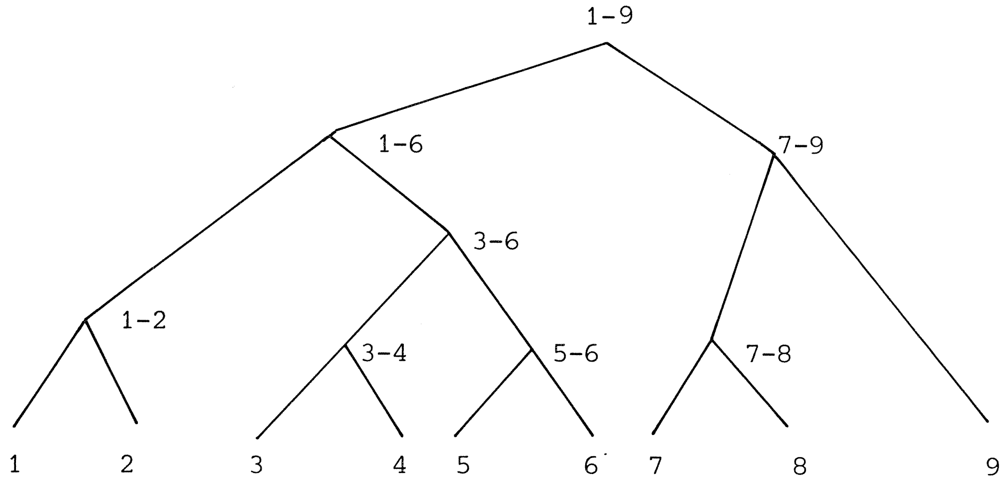


Figure 2. The skeletal connected interval tree

The internal nodes of the tree represent the total interval spanned by their sons; hence for a node ν we write ν-*interval* to mean the interval represented by ν. A line segment I *covers* a ν-interval if the interval represented by ν is contained in I. At each node ν in the tree we place four additional pieces of information given by INFO(ν) [1:4]. The different fields of INFO(ν) are updated at each scan line s_i so as to contain the following information at the end of the update.

INFO(ν)[1]: an *integer* which equals the number of times the ν-interval is completely covered by an *alive* line segment, that is, the left (lower) border of a rectangle which has been inserted in the tree at the present or a previous scan line, and not yet been deleted from the tree as the result of meeting a right (upper) border at the present or an earlier scan line. (Cf. also VAN LEEUWEN and WOOD [1979]).

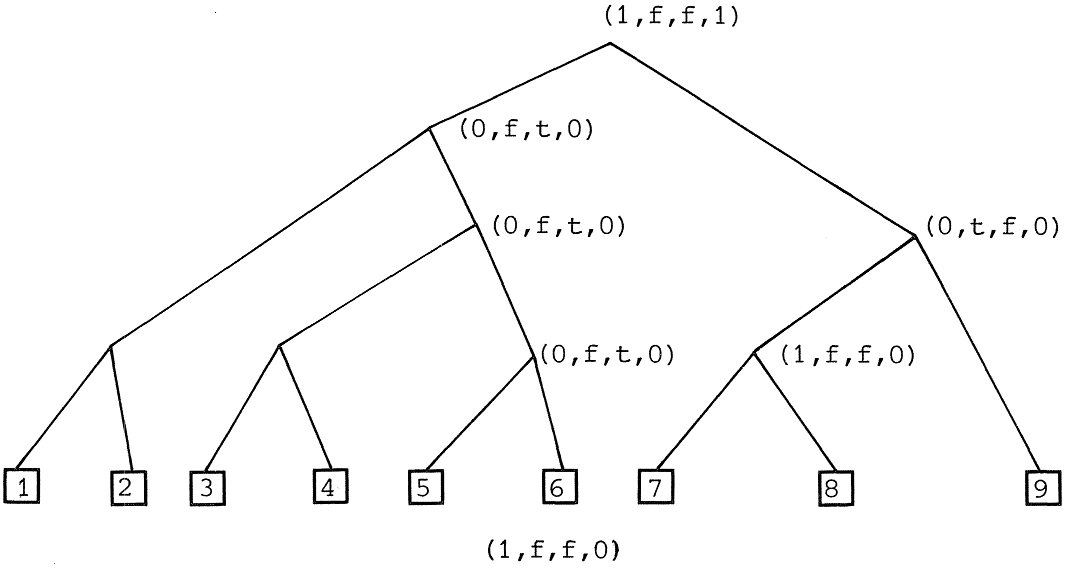
INFO(ν)[2]: a *boolean* which is *true* if the leftmost leaf-interval of the subtree rooted at ν is covered by an alive line segment *which does not* cover all of the ν-interval, and *false* otherwise.

INFO(ν)[3]: a *boolean* as INFO(ν)[2] but about rightmost leaf-intervals.

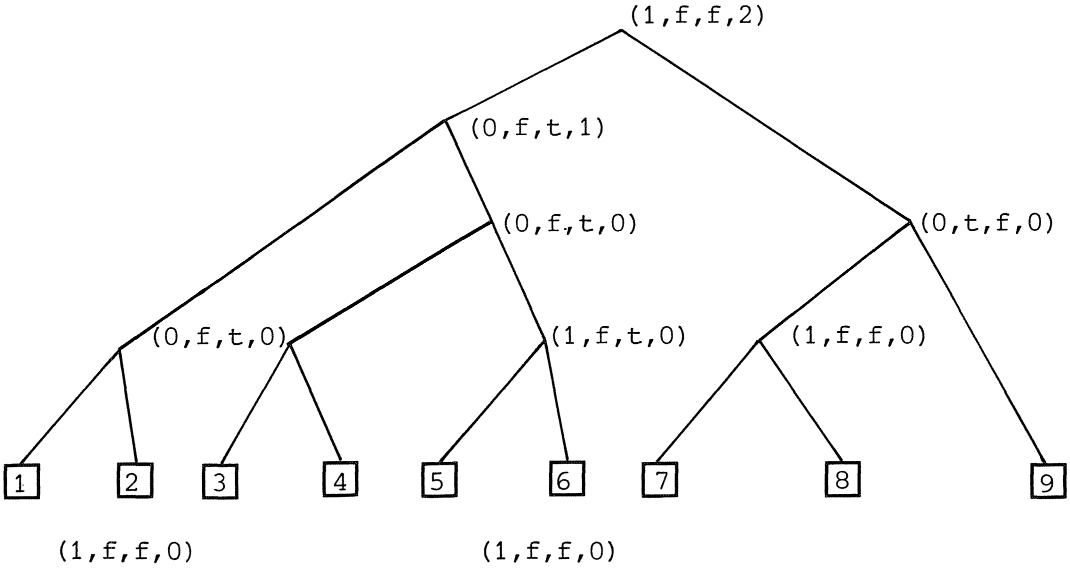
INFO(ν)[4]: an *integer* which equals the number of maximal connected intervals, contained in the ν-interval, which do not contain the leftmost leaf-interval nor the rightmost leaf-interval of the subtree rooted at ν.

Initially, INFO(ν)[1] = INFO(ν)[4] = 0 and INFO(ν)[2] = INFO(ν)[3] = *false* for all nodes ν of the connected interval tree T. At each scan line s_i, $1 \leq i \leq 2n$, we either insert a line segment (viz. the lower border of a rectangle) or delete a line segment (viz. the upper border of a rectangle) from the connected interval tree. Subsequent to the updating which is involved, INFO(root) contains the information needed to determine the number of maximal connected intervals at s_i. Specifically, the number of m.c.i.'s equals 1 if INFO(root)[1] $\neq$ 0 and equals INFO(root)[2] + INFO(root)[3] + INFO(root)[4] otherwise (where we assume *true* $\equiv$ 1 and *false* $\equiv$ 0.). For example, at scan lines s_2, s_4 and s_6 of Figure 1 the connected interval tree will be as shown in Figure 3 a,b and c, were only the significant INFO values are displayed. Since INFO(root)[1] $\neq$ 0 in all three cases there is only 1 m.c.i. as expected. However, if A_1 is deleted from the set of rectangles and from the trees in Figures 1 and 3 respectively, then at s_2 there is one m.c.i. since INFO(root)[2] + INFO(root)[3] + INFO(root)[4] = 1; at s_4 there are 2 m.c.i.'s and at s_6 again 2 m.c.i.'s.

after s_2:



after s_4:



after s_6:

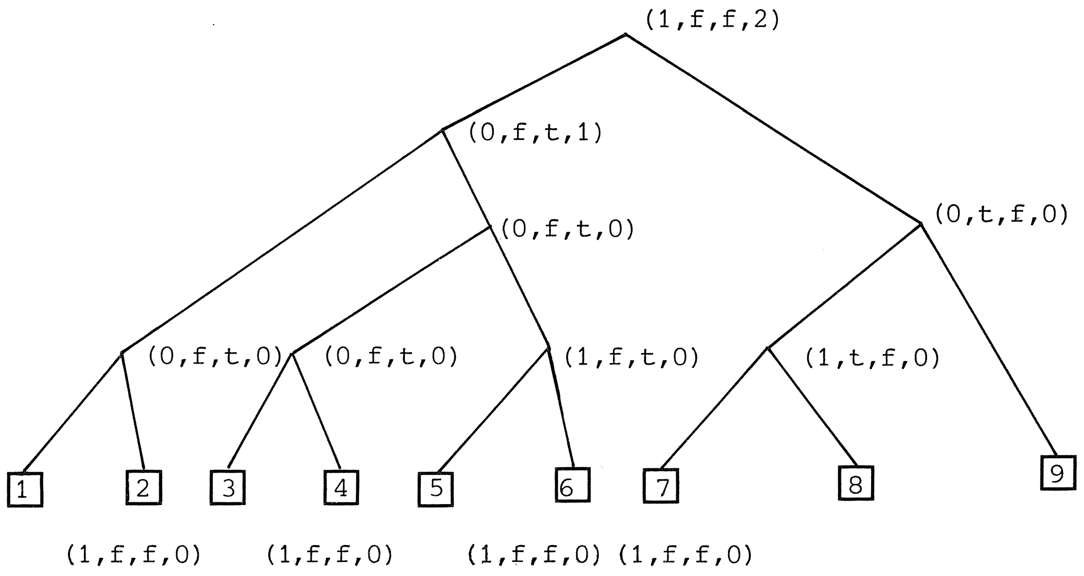


Figure 3.

Having introduced the connected interval tree, we now describe the insertion and deletion procedures for the line segments. Note that the structure of the tree remains unchanged during these operations, only the information contained in the tree is modified.

For convenience sake let I denote the line segment to be inserted or deleted, and let $I(\nu)$ denote the ν-interval for each node ν in the tree T. Furthermore, let ν_ℓ, ν_r and ν_f denote the leftson, rightson and father of a node ν; and let $LEFT(I(\nu))$ and $RIGHT(I(\nu))$ denote the leftmost and rightmost leaf-intervals, respectively, of the subtree rooted at ν.

Basically, the insertion and deletion in the connected interval tree follows the strategy used in the segment tree of BENTLEY [1977] and BENTLEY and WOOD [1979].

On insertion of I into the connected interval tree T, we first visit the root. At each node ν visited during insertion one of the following four conditions hold.

(i) $I(\nu) \subseteq I$.

(ii) $I(\nu) \nsubseteq I$ *and* $\mathrm{LEFT}(I(\nu)) \subseteq I$.

(iii) $I(\nu) \nsubseteq I$ *and* $\mathrm{RIGHT}(I(\nu)) \subseteq I$.

(iv) $I(\nu) \nsubseteq I$ *and* $\mathrm{LEFT}(I(\nu)) \nsubseteq I$ *and* $\mathrm{RIGHT}(I(\nu)) \nsubseteq I$.

Note that conditions (i)-(iv) exclude each other. Condition (i) will increment $\mathrm{INFO}(\nu)[1]$ by 1. Condition (ii) causes $\mathrm{INFO}(\nu)[2]$ to be set to *true*. Condition (iii) causes $\mathrm{INFO}(\nu)[3]$ to be set to *true*. Condition (iv) may cause a change in $\mathrm{INFO}(\nu)[4]$ to be discussed below.

On visiting a node ν, the decision to visit any of the sons of ν depends on whether

(a) $I(\nu) \subseteq I$.

(b) $I(\nu) \nsubseteq I$ *and* $I(\nu_\ell) \cap I \neq \emptyset$.

(c) $I(\nu \nsubseteq I$ *and* $I(\nu_r) \cap I \neq \emptyset$.

In case (a) neither of ν's sons is visited. In case (b) ν_ℓ is visited and in case (c) ν_r is visited. Note that both (b) and (c) may hold, causing a visit to both sons of ν. However, it is not difficult to see (and shown in BENTLEY [1977] and BENTLEY and WOOD [1979]) that at each level of the tree at most 4 nodes may be visited. Since there are at most $2n-1$ leaves in the tree, and the tree is of minimal height by construction, on each insertion of a line segment into T $\mathcal{O}(\log n)$ nodes are visited. During deletion of I from T exactly the same strategy is followed as during insertion. The only differences lie in the way in which the $\mathrm{INFO}(\nu)$ associated with the visited nodes ν is updated. Hence also during the deletion of a line segment from T $\mathcal{O}(\log n)$ nodes are visited. Both insertion and deletion in T have a downward and an upward phase. We now present the insertion and deletion algorithms for connected interval trees.

```
Insert(I,ν); I is the interval to be inserted, ν is the root node of the
             connected interval tree;
begin
¢ Downward phase ¢

  if $I(\nu) \subseteq I$ then INFO(ν)[1] := INFO(ν)[1] + 1
  else
  if $I(\nu_\ell) \cap I \neq \emptyset$
```

```
  then if $LEFT(I(\nu)) \subseteq I$ then $INFO(\nu)[2]$ := true fi;
       Insert $(I,\nu_\ell)$ fi;
  if $I(\nu_r) \cap I \neq \emptyset$
  then if $RIGHT(I(\nu)) \subseteq I$ then $INFO(\nu)[3]$ := true fi;
       Insert $(I,\nu_r)$ fi
  fi;

¢ Upward phase ¢

  if $INFO(\nu_\ell)[1] \geq 1$ and $INFO(\nu_r)[1] \geq 1$
  then $INFO(\nu)[4] := 0$
  else
      if $INFO(\nu_\ell)[1] \geq 1$
      then $INFO(\nu)[4] := INFO(\nu_r)[4]$
      else
          if $INFO(\nu_r)[1] \geq 1$
          then $INFO(\nu)[4] := INFO(\nu_\ell)[4]$
          else
              if $INFO(\nu_\ell)[3]$ or $INFO(\nu_r)[2]$
              then $INFO(\nu)[4] := INFO(\nu_\ell)[4] + INFO(\nu_r)[4] + 1$
              else $INFO(\nu)[4] := INFO(\nu_\ell)[4] + INFO(\nu_r)[4]$
              fi
          fi
      fi
  fi
end of insertion.
```

Note that during the upward phase of the insertion algorithm $INFO(\nu)[4]$ is re-computed for all nodes of T on the insertion paths. During the downward phase $INFO(\nu)[1:3]$ is updated for all nodes of T on the insertion paths.

For deletion we have a similar algorithm, which we now give:

```
Delete $(I,\nu)$; I is the interval to be deleted; $\nu$ is the root of the con-
                  nected interval tree;
```

```
begin ¢ Downward phase ¢
    if I($\nu$) $\subseteq$ I then INFO($\nu$)[1] := INFO($\nu$)[1] - 1
    else if I($\nu_\ell$) $\cap$ I $\neq \emptyset$ then Delete (I,$\nu_\ell$) fi;
         if I($\nu_r$) $\cap$ I $\neq \emptyset$ then Delete (I,$\nu_r$) fi
    fi;
    ¢ Upward phase ¢
    INFO($\nu$)[2] := INFO($\nu_\ell$)[2] or (INFO($\nu_\ell$)[1] $\geq$ 1);
    INFO($\nu$)[3] := INFO($\nu_r$)[3] or (INFO($\nu_r$)[1] $\geq$ 1);
    if INFO($\nu_\ell$)[1] $\geq$ 1 and INFO($\nu_r$)[1] $\geq$ 1
    then INFO($\nu$)[4] := 0
    else if INFO($\nu_\ell$)[1] $\geq$ 1
         then INFO($\nu$)[4] := INFO($\nu_r$)[4]
         else if INFO($\nu_r$)[1] $\geq$ 1
              then INFO($\nu$)[4] := INFO($\nu_\ell$)[4]
              else if INFO($\nu_\ell$)[3] or INFO($\nu_r$)[2]
                   then INFO($\nu$)[4] := INFO($\nu_\ell$)[4] + INFO($\nu_r$)[4] + 1
                   else INFO($\nu$)[4] := INFO($\nu_\ell$)[4] + INFO($\nu_r$)[4]
                   fi
              fi
         fi
    fi
end delete.
```

Here during the downward phase INFO(ν)[1] is updated and during the upward phase INFO(ν)[2:4].

For both insertion and deletion the number of nodes visited is $\mathcal{O}(\log n)$.

The number of maximal connected intervals at a scan line s_i is contained in INFO(root) subsequent to the insertion or deletion at this scan line. To obtain this number we do a query:

```
Query(T) := if INFO(root)[1] $\neq$ 0
            then 1
            else INFO(root)[4] + INFO(root)[3] +
                 INFO(root)[2]
            fi ¢ we take true $\equiv$ 1 and false $\equiv$ 0 ¢
```

The algorithm is correct if subsequent to each insertion or deletion the INFO(ν) for each node ν in T is correct. This is easily ascertained to be so.

3. THE MEASURE PROBLEM

To determine the measure is easier than to determine the perimeter. We use the same connected interval tree skeleton with different additional information at each node ν: INFO(ν)[1:3].

INFO(ν)[1]: as before.
INFO(ν)[2]: a *real* equal to the length of the ν-interval.
INFO(ν)[3]: a *real* equal to the sum of the lengths of the maximal connected intervals, covered by inserted and not yet deleted line segments, contained in the ν-interval.

INFO(ν)[2] is determined at the time of setting up the connected interval tree. INFO(ν)[1] and INFO(ν)[3] are computed during insertion and deletion, quite similar to INFO(ν)[1] and INFO(ν)[4] in the algorithm for the perimeter problem. INFO(ν)[3] can be determined by the following piece of program (in an upward phase):

```
if INFO($\nu_\ell$)[1] ≥ 1 and INFO($\nu_r$)[1] ≥ 1
then INFO($\nu$)[3] := INFO($\nu_\ell$)[2] + INFO($\nu_r$)[2]
else if INFO($\nu_\ell$)[1] ≥ 1
     then INFO($\nu$)[3] := INFO($\nu_\ell$)[2] + INFO($\nu_r$)[3]
     else if INFO($\nu_r$)[1] ≥ 1
          then INFO($\nu$)[3] := INFO($\nu_\ell$)[3] + INFO($\nu_r$)[2]
          else INFO($\nu$)[3] := INFO($\nu_\ell$)[3] + INFO($\nu_r$)[3]
          fi
     fi
fi
```

Note that INFO(leaf)[3] = 0. Finally, for the measure we have

```
QUERY(T) := if INFO(root)[1] > 0 then INFO(root)[2]
                                  else INFO(root)[3]
            fi
```

4. TIME AND SPACE REQUIREMENTS

Using the connected interval tree and its associated updating algorithms we can compute the perimeter of a set of rectangles based on formula (1) and the measure of a set of rectangles based on formula (2). Thus, in both cases the time taken for n rectangles can be expressed as

$$(3) \qquad \mathrm{TIME}(n) = \mathcal{O}(n \log n + n + \mathrm{UPDATE}(n) + \mathrm{QUERY}(n)).$$

The endpoints of the rectangles need to be sorted (in both the x- and the y-directions for the perimeter problem), taking $\Theta(n \log n)$ time; the skeletal connected interval tree can be constructed in $\Theta(n)$ time; and at each scan line there is a deletion or insertion taking a total of UPDATE(n) time and a query taking a total of QUERY(n) time. Similarly, the space needed for n rectangles can be expressed as:

$$(4) \qquad \mathrm{SPACE}(n) = \mathcal{O}(n + \mathrm{INF}(n)),$$

since the n rectangles and the skeletal connected interval tree, that is without $\mathrm{INFO}(\nu)$ at each node ν, require $\Theta(n)$ space. INF(n) denotes the space required to store the $\mathrm{INFO}(\nu)$'s for n rectangles.

Thus we see that the actual space/time costs depend to a large extend on the costs required to update and store $\mathrm{INFO}(\nu)$ or, more precisely, on how much we charge for reals, boolean and integers and the manipulations performed on them as far as relevant in this setting.

Booleans: we may clearly charge a constant amount in space for storage and in time for manipulation.

Reals : Obviously here everything depends on the precision we require. However, it seems reasonable to charge a constant amount in space for storage and in time for manipulation by keeping reals in $\Theta(1)$ words.

Integers: we will consider three measures:

(I) Constant cost for storage and time for manipulation.

(II) Logarithmic cost for storage and time for manipulation;

we can for example keep an m-bit integer $a_1a_2\ldots a_m$ as a linked list of length m/r:

$$\boxed{a_1a_2\ldots a_r} \longrightarrow \boxed{a_{r+1}a_{r+2}\ldots a_{2r}} \longrightarrow \ldots \longrightarrow \boxed{\quad \ldots a_m}.$$

(III) The cost resulting of storing integers in unary notation, for example as a linked list or counter:

$$\boxed{1} \longrightarrow \boxed{1} \longrightarrow \ldots \quad \text{or} \quad \longrightarrow \begin{array}{|c|}\hline 1\\ \hline 1\\ \hline \vdots\\ \hline 1\\ \hline\end{array}$$

In the following we shall analyze the costs, resulting from the assumed constant costs for booleans and reals together with each of the costs (I), (II) and (III) for integers, in time and space for both the perimeter - and the measure algorithm. Unless stated otherwise, all estimates for running time and storage cost are tacitly assumed to be for the *worst case.*

I. Constant cost criterion

Each update requires $\Theta(\log n)$, viz. the number of visited nodes, and each query requires $\Theta(1)$. Hence we immediately have $\text{UPDATE}(n) = \Theta(n \log n)$, $\text{QUERY}(n) = \Theta(n)$; and furthermore $\text{INF}(n) = \Theta(n)$; for both the perimeter and measure algorithms. Hence for both algorithms we have under the constant cost criterion that

(5) $$\text{TIME}_{\text{const.}}(n) = \Theta(n \log n)$$

and

(6) $$\text{SPACE}_{\text{const.}}(n) = \Theta(n).$$

II. Logarithmic cost criterion

First note that the contribution of $\text{INFO}(\nu)[2]$ and $\text{INFO}(\nu)[3]$ is $\Theta(1)$ for each ν in T for both the perimeter and the measure connected interval trees.

There are at most n insertions in the connected interval tree. Therefore, for each node ν in T, INFO(ν)[1] needs to count up to n, which costs $\mathcal{O}(\log n)$ space under the logarithmic cost. Since there are $\Theta(n)$ nodes in the tree, all in all it seems to take $\mathcal{O}(n \log n)$ space to maintain INFO(.)[1] for the total tree. However, this estimate is much to crude as we now show.

Since the connected interval tree T has at most 2n-1 leaves, each insertion visits at most 2 log 2n nodes. (I.e., twice the height of the tree.) At each of these visited nodes ν, INFO(ν)[1] might be incremented with 1 during an insertion. Hence the total count of the 4n nodes in the tree, with respect to INFO(.)[1] satisfies

$$(7) \qquad \sum_{\nu\in T} \text{INFO}(\nu)[1] \le 2n \log 2n$$

since there are n insertions. If all insertions visit exactly the same nodes in T, then T must consist of only the root and the space used by INFO(root)[1] is $\mathcal{O}(\log n)$. We obtain an upper bound on the space used by the INFO(.)[1]'s as follows. The amount of space used by the combined INFO(ν)[1]'s is maximized if they all count to the same maximum, i.e. (log 2n)/2. This takes $\Theta(\log\log n)$ space for each INFO(ν)[1]. Hence

$$(8) \qquad \sum_{\nu\in T} \text{length}(\text{INFO}(\nu)[1]) \le \Theta(n \log\log n).$$

Secondly, we need to bound the space used by the INFO(.)[4]'s. Clearly this is maximized when the number of connected intervals is maximized. This occurs when the leaf-intervals 1,3,5,...,2n-1 are covered and 2,4,...,2n-2 are not. Consider a node ν at level i in the tree T. Then INFO(ν)[4] is approximately equal to $2^{\log n - i}$. (Hence, e.g. INFO(root)[4] = n, INFO(leaf)[4] = 0 and INFO(father of leaf)[4] = 1).

Now since there are 2^i nodes ν at level i in T, and there are log n levels in T, we obtain:

$$(9) \qquad \sum_{\nu\in T} \text{length}(\text{INFO}(\nu)[4])$$

$$= \sum_{i=0}^{\log n} \sum_{\substack{\nu \text{ in level} \\ i \text{ of } T}} \text{length}(\text{INFO}(\nu)[4])$$

$$= \Theta(\textstyle\sum_{i=0}^{\log n} 2^i(\log n - i))$$

$$= \Theta(2^{\log n} \textstyle\sum_{j=0}^{\log n} j.2^{-j}), \quad \text{by } j = \log n - i,$$

$$= \Theta(n \textstyle\sum_{j=0}^{\log n} j.2^{-j}).$$

Since for $2^j > j^3$, e.g. for $j \geq 10$, we have that $j.2^{-j} < j^{-2}$, and since $\Sigma_{j=1}^{\infty} j^{-2}$ converges, we obtain

$$(10) \qquad \textstyle\sum_{j=0}^{\log n} j.2^{-j} < \sum_{j=0}^{\infty} j.2^{-j}$$

$$< \textstyle\sum_{j=0}^{9} j.2^{-j} + \sum_{j=10}^{\infty} j^{-2}$$

$$= c,$$

for some constant c. Hence from (9) and (10) we find that

$$(11) \qquad \sum_{\nu \in T} \text{length}(\text{INFO}(\nu)[4]) = \Theta(n).$$

From (8) and (11) and the previous remarks we find that the worst case space requirements under the logarithmic cost for both the perimeter and measure algorithm are

$$(12) \qquad \text{SPACE}_{\text{log.}}(n) = \Theta(n \log \log n).$$

Time requirements

We now analyze TIME(n), that is, UPDATE(n) and QUERY(n). First consider the contribution from INFO(.)[1]. We saw above, that the space used is maximized when each node contains a count of $\Theta(\log n)$. However, time taken is maximized when $\Theta(\log n)$ nodes contain a count of $\Theta(n)$, that is, when $\Theta(n)$ identical line segments I are inserted which occasion the visit of $\Theta(\log n)$ nodes. In this case a further insertion or deletion of I takes $\Theta(\log^2 n)$ time, yielding a total contribution of $\Theta(n \log^2 n)$ for the manipulation time of INFO(.)[1], as opposed to $\Theta(n \log \log n)$ time in the former case. By merging $\frac{1}{2}n$ rectangles consuming maximal space and $\frac{1}{2}n$ rectangles consuming maximal time, as indicated above, we reach simultaneously the worst-case space and time requirements. The time bound given is indeed worst-case since there are at most $\Theta(n)$ insertions/deletions and each insertion/deletion necessitates the visiting of $\mathcal{O}(\log n)$ nodes, each of which requires at most $\mathcal{O}(\log n)$ time for the updating of INFO(.)[1]. Hence, to maintain INFO(.)[1] during n insertions/deletions might take a worst-case time of order

$$(13) \qquad \Theta(n \log^2 n).$$

Now consider the contribution of INFO(.)[4]. Based on the space analysis of INFO(.)[4] we observe that the worst situation which can occur is when we need to add two values of INFO(.)[4] one of which is maximal. Since there are at most four nodes visited on each level during an insertion or deletion, and length (INFO(ν)[4]) $\le \log n - i$ for a node ν at level i of the tree, we obtain a contribution of

$$(14) \qquad \mathcal{O}(\textstyle\sum_{i=0}^{\log n} (\log n - i))$$

$$= \mathcal{O}(\textstyle\sum_{j=0}^{\log n} j) = \mathcal{O}(\log^2 n).$$

Therefore, to maintain INFO(.)[4] during n insertions or deletions might take time of order

$$(15) \qquad \Theta(n \log^2 n).$$

Hence, by (13) and (15), we obtain that for both the perimeter and measure algorithm we have a worst-case UPDATE(n) = $\Theta(n \log^2 n)$. Since each query for the perimeter algorithm takes $\mathcal{O}(\log n)$ time ($\Theta(\log n)$ time in the worst-case) we have QUERY(n) = $\Theta(n \log n)$ for the perimeter algorithm. Because of the constant cost for reals we have QUERY(n) = $\Theta(n)$ for the measure algorithm. Therefore, under the logarithmic cost criterion, we have for both the perimeter and the measure algorithms a worst-case running time of:

$$(16) \qquad \mathrm{TIME}_{\log.}(n) = \Theta(n \log^2 n).$$

III. Unary cost criterion

Space requirements

By (7) we need all in all $\Theta(n \log n)$ space to store all the INFO(.)[1]'s. From (9) we compute the space needed for the INFO(.)[4]'s, namely:

$$(17) \qquad \sum_{\nu \in T} \text{INFO}(\nu)[4]$$

$$= \Theta\left(\sum_{i=0}^{\log n} 2^i . 2^{\log n - i}\right)$$

$$= \Theta(n \log n).$$

Hence INF(n) = $\Theta(n \log n)$ for both perimeter and measure algorithms and therefore, for both,

$$(18) \qquad \text{SPACE}_{\text{unary}}(n) = \Theta(n \log n).$$

Time requirements

Incrementing and decrementing an integer in unary notation is a constant cost operation, and so is checking for 0. Hence INFO(.)[1]'s contribution to UPDATE(n) is $\Theta(n \log n)$ for n insertion/deletions which visit $\Theta(\log n)$ nodes apiece. The contribution of the INFO(.)[4]'s is simply the cost in time of catenating two integers in unary notation. Clearly this can be carried out in constant time, say in time equal c. Hence the total of the contribution of the INFO(.)[4]'s is given by

$$\Theta\left(n \sum_{i=0}^{\log n} c\right) = \Theta(n \log n)$$

for n insertions/deletions. Thus, for both the perimeter and the measure algorithm we find under the unary cost criterion:

$$\text{UPDATE}(n) = \Theta(n \log n).$$

A difference between the measure and the perimeter algorithm comes in with the query. Whereas a query in the measure algorithm extracts a real from INFO(root), at constant cost $\Theta(1)$, a query in the perimeter algorithm extracts an integer written in unary from INFO(root) at cost $\mathcal{O}(n)$. Therefore we have for the measure algorithm that

$$QUERY(n) = \Theta(n)$$

while for the perimeter algorithm

$$QUERY(n) = \Theta(n^2),$$

since there may be $\Theta(n)$ connected intervals in the worst case.

Combining the above we obtain:

(19) $$TIME_{unary}(n) = \Theta(n \log n) \text{ for the measure problem}$$

and

(20) $$TIME_{unary}(n) = \Theta(n^2) \text{ for the perimeter problem.}$$

5. CONCLUDING REMARKS

We have presented an algorithm for the perimeter problem and analyzed its time-space requirements (for the worst-case) under three different cost measures for storing and manipulating integers. These are summarized in the following table together with a similar analysis of the related algorithm for the measure problem.

	PERIMETER		MEASURE	
	space	time	space	time
Constant cost	$\Theta(n)$	$\Theta(n \log n)$	$\Theta(n)$	$\Theta(n \log n)$
Logarithmic cost	$\Theta(n \log \log n)$	$\Theta(n \log^2 n)$	$\Theta(n \log \log n)$	$\Theta(n \log^2 n)$
Unary cost	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n \log n)$

Figure 4. Table

An analysis of the solution for the measure problem as in BENTLEY [1977] or VAN LEEUWEN and WOOD [1979] yields the same results.

The only distinction between the performances of the algorithms for the perimeter and for the measure problem occurs when using unary notation for the integers. This reflects the fact that the one-dimensional measure of the connected intervals is stored as a real, while their number is stored as an integer.

The results strengthen the conjecture that $\Theta(n)$ space *and* $\Theta(n \log n)$ time are unattainable as the worst-case performance of algorithms, for both the perimeter and measure problems, under realistic cost assumptions.

Note that it is realistic to assume a constant cost for storage and manipulation of reals, but that it is unrealistic to assume a constant cost for the storage and manipulation of integers in the discussed algorithms. This is so, because in many applications (like the ones sketched in the introduction) we may deal with a very great number of rectangles in a very limited area. A computation of the perimeter or the measure according to our algorithms *requires* exact bookkeeping of integers: it is easy to construct examples which give completely different answers otherwise. The algorithms in question *approximate* the values of the perimeter and measure insofar as the real values of the rectangle coordinates in the plane are precise.

The *d-dimensional* perimeter problems can be solved by a divide-and-conquer technique in $\mathcal{O}(dn^{d-1} \log n)$ time, $d \geq 2$, under the constant cost criterion. This can probably be improved to an $\mathcal{O}(dn^{d-1})$ time algorithm for $d \geq 3$ by techniques similar to those in VAN LEEUWEN and WOOD [1979].

It is easy to see that the algorithm as presented in section 2 can be changed so as to output the *boundary* of $\bigcup_{i=1}^{n} A_i$ too. This algorithm for computing the boundary of a set of intersecting rectangles may be of use to computer graphics and would have the same time/storage requirements as the perimeter algorithm under the unary cost criterion. Hint: maintain the real end points of the m.c.i.'s concerned in INFO(ν)[2:4] in the perimeter algorithm, instead of their presence and/or number.

REFERENCES

BENTLEY, J.L., [1977], *Algorithms for Klee's rectangle problems,* unpublished notes, Carnegie-Mellon Univ., 1977.

BENTLEY, J.L., & D. WOOD, [1979], *An optimal worst-case algorithm for reporting intersections of rectangles,* Techn. Rept. 79-CS-13, McMaster University, Unit for Computer Science, 1979.

FINKEL, R.A. & J.L. BENTLEY, [1974], *Quad trees, A data structure for retrieval on composite keys,* Acta Informatica 4 (1974) 1-9.

KLEE, V., [1977], *Can the measure of* $\cup_{i=1}^{n} [a_i, b_i]$ *be computed in less than* $\mathcal{O}(n \log n)$ *steps?*, American Mathematical Monthly 84 (1977) 284-285.

VAN LEEUWEN, J. & D. WOOD, [1979], *The measure problem for intersecting ranges in d-space,* manuscript in preparation, Univ. of Utrecht, Vakgroep Informatica.