Algorithm 91

BALANCING A BINARY TREE

A. Colin Day
Computer Centre
University College London
London.

Author's Note:

Ordered binary trees are very useful tools when performing a search. A datum is compared with the data at successive nodes of the tree. When used for such a purpose, the greatest efficiency is obtained when the tree is most regular, i.e. when all routes through the tree (from root to leaf) are approximately the same length. Regularising a tree in this way is here referred to as *balancing* the tree. Algorithms have been described for dynamically balancing a tree to which nodes are continually being added (Knuth, 1973). However, the algorithm described here balances a fixed tree, and so is of use when the search tree is available in its entirety at the outset of the operation.

The terminology used here differs in some respect from Knuth's, so the terms must be defined at the start. A *complete* tree is one whose routes are all the same length. An example is shown in **Fig. 1** of a complete tree of height 3, i.e. the length of every route through the tree is 3. A *balanced* tree is one for which no route differs in length from any other route by more than 1. Given a fixed number of nodes, the balanced tree is the tree of minimum height which can be constructed. A balanced tree is said to be of height n if its routes are of length n or $n - 1$. **Fig. 2** gives an example of a balanced tree of height 4. Note that a complete tree of height n may by this definition be considered a balanced tree of height n or a balanced tree of height $n + 1$.

The purpose of this algorithm is to produce a balanced binary tree from any input binary tree, using no extra workspace, and maintaining the order within the nodes of the tree. It can thus be used after the binary tree has been used to sort the data at the nodes into some (alphabetic or numeric) order. One important aspect of the algorithm is that the data forming the nodes of the tree are not moved around; only the pointers are changed.

The first step is to strip the nodes from the tree in order, keeping them chained together by means of their right pointers. The input tree must have positive pointers in the vectors ILPT and IRPT giving the left and right pointers respectively. It must also have negative pointers representing backtracks (as in Day, 1972). Stripping a tree in this way is well known and described, so no further description of the process will be given here. (Note that although the input tree is threaded by means of the negative backtrack pointers, the output tree produced by this algorithm is not threaded.)

The list of nodes chained by their right pointers may now be represented as in **Fig. 4**(*a*). This will be termed the *backbone*. The balancing algorithm works by applying to pairs of nodes in the backbone the transformation shown in **Fig. 3.** This transformation is accomplished in three steps:

1. Make the predecessor of B point to D
2. Make the right pointer of B point to C
3. Make the left pointer of D point to B.

Note that if A and C are both complete trees of height n, then B will become a complete tree of height $n + 1$.

This transformation will not affect the (alphabetic or numeric) order of the items within the tree. This order is revealed by stripping the tree, for each node taking first the left subtree, then the item at the node, then the right subtree. Stripping each side of Fig. 3 gives the order *ABCD*.

The operation of applying the transformation to adjacent pairs of nodes on the backbone is known as a *pass*. Several passes are made. After the first one, the nodes of the backbone point (via their left pointers) to complete trees of height 1, as in Fig. 4(*b*). After the second pass, the nodes of the backbone point to complete trees of height 2. After the nth pass, the nodes point to complete trees of level n.

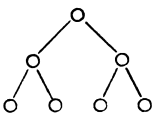

Fig. 1 A complete tree (height 3)

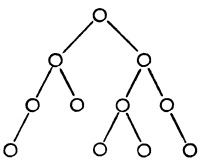

Fig. 2 A balanced tree (height 4)

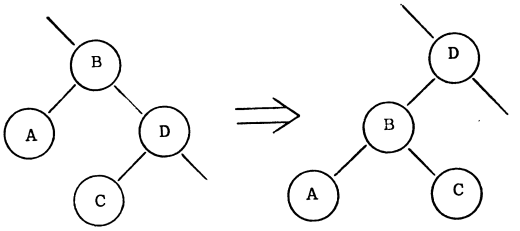

Fig. 3 The transformation

The crucial question for the algorithm is; How many times do we apply the transformation within each pass? This is controlled by adjusting the length of the backbone. The algorithm may be stated in words as follows:

1. Reduce the length of the backbone by 1
2. Divide the length of the backbone by 2 to find the number of transformations, m
3. If m is zero, exit; otherwise perform m transformations on the backbone
4. Return to 1.

Note that the division in step (2) rounds down if the length is not even. After the transformations have been made, the length of the backbone must be reduced by m to allow for the nodes which have been removed from it by the m transformations.

In order to demonstrate that the algorithm does in fact produce a balanced tree, the method of mathematical induction will be used. The symbols shown in **Fig. 5** will be used for complete trees and balanced trees of height n. After n passes have been made, let us suppose that the situation may be represented as in **Fig. 6.** The dotted line shows the limit of the backbone. According to step (1) of the algorithm, we reduce the length of the backbone by 1, giving us **Fig. 7**(*a*). It will be obvious that this is exactly the same as Fig. 7(*b*). We now make one more pass (the $n + 1$th). If the length of the backbone is even, then the resulting situation is as in **Fig. 8**(*a*). If it was odd, then one node will not be paired off and will not enter into the transformation, so the result will be as in Fig. 8(*b*). However, a complete tree of height n or a complete tree of height $n + 1$ are both equivalent to a balanced tree of height $n + 1$, since the latter may have routes through it of length n or $n + 1$. Therefore both 8(*a*) and 8(*b*) are equivalent to **Fig. 9,** which is homomorphous with Fig. 6.

Two matters still remain. It must be shown that at the lowest level we have a state of affairs which can be represented by Fig. 6, and the exit condition must be considered. The first of these is trivial. If n is 0, and if the backbone covers every node, then Fig. 6 is identical to Fig. 4(*a*).

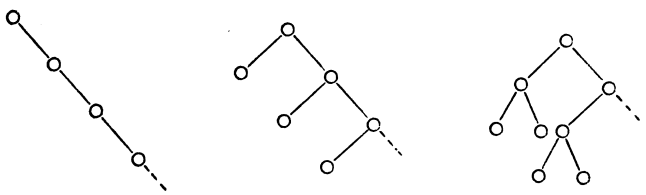

Fig. 4 (*a*) Before first pass (*b*) After first pass (*c*) After second pass

Figure 5 (*a*) Symbol for complete tree of height *n* (*b*) Symbol for balanced tree of height *n*

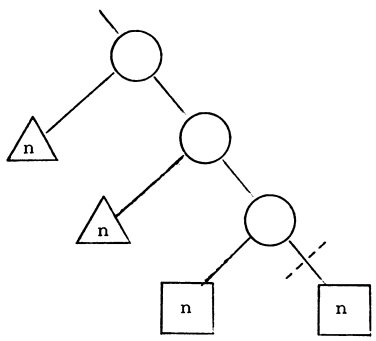

Fig. 6

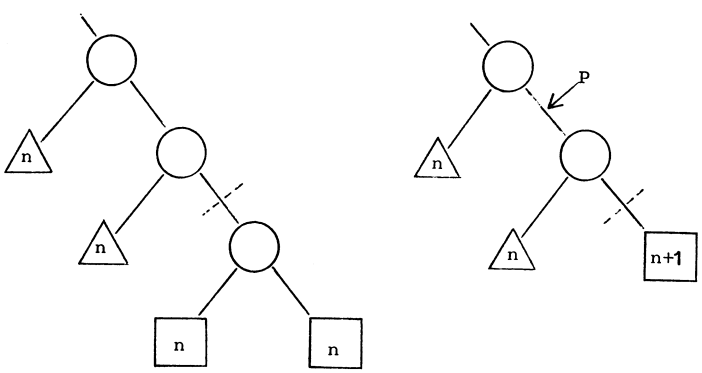

Fig. 7

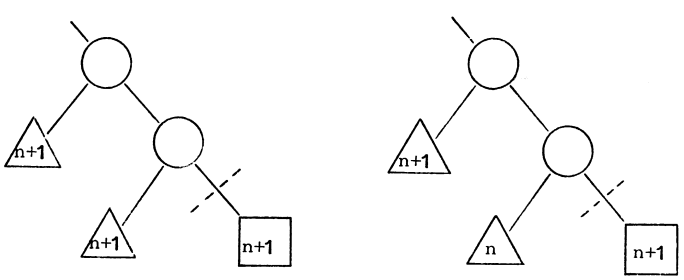

Fig. 8

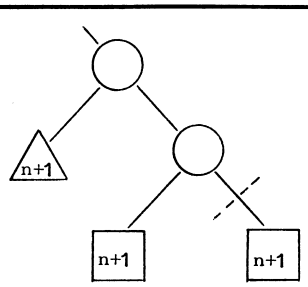

Fig. 9

The exit condition arises when m is 0. This happens when the length of the backbone is 1, and is represented by Fig. 7(*b*) if the backbone begins at point P, and everything above and to the left of P does not exist. In this case it is obvious that we are left with a balanced tree of height $n + 2$, and therefore the algorithm terminates only when the tree is balanced.

References

DAY, A. C. (1972). *Fortran Techniques, with special reference to non-numerical applications*, London: Cambridge University Press, pp. 76-81.

KNUTH, D. E. (1973). *The Art of Computer Programming, Vol. 3. Sorting and Searching*, Reading: Addison Wesley, pp. 451-468.


```
      SUBROUTINE BALNCE(ILPT,IRPT,N,IROOT)
C
C         BALANCES A TREE OF N ITEMS.
C            ILPT(N) - LEFT POINTERS
C            IRPT(N) - RIGHT POINTERS
C            IROOT - ROOT OF TREE (CHANGED BY SUBROUTINE)
C
      DIMENSION ILPT(N), IRPT(N)
C
C         STRIP ITEMS FROM TREE AS LIST
C
      IF (N .LE. 1) RETURN
      ISTRT = 0
      J = IROOT
      GO TO 20
C FOLLOW LEFT POINTERS
   10 J = ILPT(J)
   20 IF (ILPT(J) .GT. 0) GO TO 10
C THIS ITEM IS TO BE STRIPPED
      IF (ISTRT .GT. 0) GO TO 30
C FIRST ITEM - KEEP TRACK OF START
      ISTRT = J
      GO TO 40
C NOT FIRST - CHAIN TOGETHER
   30 IRPT(LAST) = J
   40 ILPT(J) = 0
      LAST = J
C TEST FOR BACKTRACK, END OR RIGHT BRANCH
      J = IRPT(J)
      IF (J) 50, 60, 20
C BACKTRACK
   50 J = -J
      GO TO 30
C
C         FORM BALANCED TREE
C
C SET LENGTH OF BACKBONE
   60 NBACK = N - 1
C FIND NO. OF TRANSFORMATIONS
   70 M = NBACK / 2
      IF (M .LE. 0) RETURN
C INITIALISE FOR LOOP
      J = IROOT
      L = J
C MOVE ON ROOT IN ANTICIPATION
      IROOT = IRPT(IROOT)
C PERFORM TRANSFORMATIONS
      DO 80 I = 1, M
      K = IRPT(J)
      IRPT(L) = K
      IRPT(J) = ILPT(K)
      ILPT(K) = J
      L = K
   80 J = IRPT(K)
C AMEND LENGTH OF BACKBONE
      NBACK = NBACK - M - 1
      GO TO 70
      END
```

Algorithm 92

THE DRAWING OF DASHED LINES

G. Berry,
MRC Pneumoconiosis Unit,
Penarth, S. Glamorgan, CF6 1XW

Author's note:

When producing figures on a graph plotter it is frequently required to be able to draw lines in dashed form instead of as unbroken lines. This not only eases the interpretation of the plotter output but also allows the plotter to be used to produce figures which are up to publication standard, or may easily be brought up to this standard by

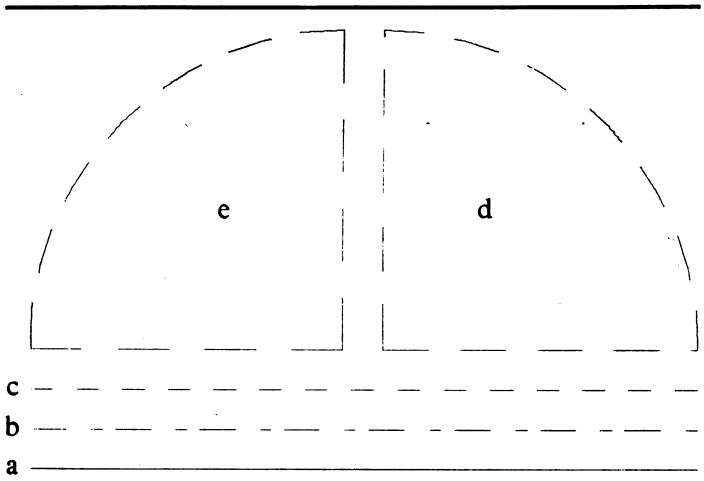

Fig. 1 Examples of output from *GPDASH*. (*a*) Continuous line. (*b*) Line 8·5 inches long on original plot with specified segment lengths 0·45, 0·3, 0·15, 0·15 inches (these lengths were multiplied by 8·5/8·25 to give complete segment at end). (*c*) As (*b*) with specified segment lengths all 0·3 inch (multiplied by 8·5/8·7). (*d*) Quadrant and radii of circle of radius 4 inches drawn starting from centre in one sequence, specified segment lengths 0·6, 0·4, 0·6, 0·4 inches. (*e*) As (*d*) except that each radius and the arc were drawn by separate calls.

touching up, with a consequent saving of effort and increase in accuracy.

In this FORTRAN algorithm, *GPDASH*, the term *dashed line* is used to denote a line consisting of four segments repeated in sequence. These segments are a dash of length d_1, a blank of length b_1, a dash of length d_2 and a blank of length b_2. The simplest form of dashed line may be produced by making these four lengths identical. Secondly, all the dashes could be of the same length with the blanks having a different length. Thirdly, a *double dashed line* may be produced by putting $d_1 \neq d_2$ and/or $b_1 \neq b_2$. An unbroken line may be drawn by putting any one of the four lengths as zero. Thus *GPDASH* allows many possibilities enabling quite complicated figures to be produced clearly.

The algorithm joins a sequence of points by straight lines and in order to give a curve, approximated by a series of short straight lines, the appearance of having been drawn in one sweep rather than piecemeal, unfinished dashes or blanks at the end of a join are finished at the beginning of the next join.

GPDASH has no provision for interpolation (other than linear) between successive points so that if it is being used to draw a curve the data points provided must be sufficiently close to produce a visually acceptable curve. This is in contrast to *QUARC* (McConalogue, 1971) in which points are joined by arcs having given slopes at their end points. A dashed curve may be drawn by *QUARC* but only in the simplest form, i.e. $d_1 = b_1 = d_2 = b_2$.

In order to ensure that the last point of the sequence is visible the lengths of the dashes and blanks are modified so that the sequence of lines finishes with a complete dash. This is achieved by calculating the total length of the sequence of lines at entry, and then either multiplying or dividing all the dash- and blank- lengths by a constant. This constant is chosen to be as close to unity as possible. Apart from the first and last points, points may occur within blanks. If the curve is continuous and has a continuous derivative this is as would be required, but if there is a discontinuity of slope at some point then this may be emphasised by splitting the sequence of points into two parts, the first part ending, and the second part starting, at the point of discontinuity.

GPDASH calls an auxiliary algorithm, *PENTO*(*K*, *X*, *Y*). This algorithm must be supplied so that a call causes the pen to move in a straight line from its current position to the point (*X*, *Y*), the pen being raised up above the paper if $K = 1$ or down in contact with the paper if $K = 2$. Before the first call of *GPDASH* an origin and axes should have been established. Throughout, all co-ordinates should be supplied in data units, and the scale of the graph is defined by the scale factors *XS* and *YS*.

Some of the possible types of dashed line which may be produced by the algorithm are illustrated in **Fig. 1.** In the lower part a continuous line and two types of dashed line are drawn. Above are two quadrants of a circle with radii, where the arcs are approximated by 13 equally spaced points. The quadrant and radii on the right were drawn as one sequence of 15 points, whereas on the left the two radii and the arc were drawn by three separate calls of the algorithm. The latter shows the change in direction at the end of the radii more clearly.

Acknowledgement

This algorithm was developed on the Cardiff Joint Computing Centre 4-70 and I am grateful to Dr. J. C. Baldwin, the Centre's Director, for allowing me facilities.

Reference

McConalogue, D. J. (1971). Algorithm 66: An automatic french-curve procedure for use with an incremental plotter, *The Computer Journal*, Vol. 14, No. 2, pp. 207-209.

```
C-------------------------------------------------------------
C     SUBROUTINE GPDASH
C
C     PURPOSE
C       FOR USE WITH PLOTTER TO JOIN A SEQUENCE OF POINTS BY A
C       DASHED LINE, CONSISTING OF FOUR SEGMENTS, DASH1, BLANK1,
C       DASH2, BLANK2, REPEATED AS NECESSARY.
C
C     USAGE
C       CALL GPDASH(N, X, Y, XS, YS, D1, B1, D2, B2)
C
C     DESCRIPTION OF PARAMETERS
C       N - NUMBER OF POINTS TO BE JOINED.
C       X - INPUT VECTOR OF ABSCISSAE OF POINTS (IN DATA UNITS).
C       Y - INPUT VECTOR OF ORDINATES OF POINTS.
C       XS - ABSCISSA SCALE FACTOR (PLOT LENGTH PER DATA UNIT).
C       YS - ORDINATE SCALE FACTOR.
C       D1 - LENGTH OF FIRST DASH (IN PLOT LENGTH UNITS).
C       B1 - LENGTH OF FIRST BLANK.
C       D2 - LENGTH OF SECOND DASH.
C       B2 - LENGTH OF SECOND BLANK.
C
C     REMARKS
C         SUBROUTINE PENTO(K,X,Y) MUST BE SUPPLIED TO TAKE
C       ACTION:  IF K=1 MOVE PEN TO (X,Y), PEN UP IN MOTION
C                IF K=2 MOVE PEN TO (X,Y), PEN DOWN IN MOTION.
C         EPS SHOULD BE SET AS MINIMUM STEP LENGTH OF PLOTTER.
C         IF ANY OF D1,B1,D2,B2 IS LESS THAN EPS AN UNBROKEN
C       LINE WILL BE PRODUCED.
C         IF N IS LESS THAN 2 OR EITHER XS OR YS IS ZERO OR
C       NEGATIVE THE ALGORITHM WILL TAKE NO ACTION.
C
C     METHOD
C       THE SEQUENCE OF POINTS ARE JOINED BY STRAIGHT LINES,
C       WITH DASHES OR BLANKS UNFINISHED AT THE END OF A JOIN
C       FINISHED AT THE BEGINNING OF THE NEXT JOIN.  THE LENGTHS
C       OF THE DASHES AND BLANKS ARE SCALED SO THAT THE SEQUENCE
C       FINISHES WITH A COMPLETE DASH.
C
C-------------------------------------------------------------
C
      SUBROUTINE GPDASH(N,X,Y,XS,YS,D1,B1,D2,B2)
      DIMENSION X(N),Y(N),DL(4)
      DATA EPS /0.01/
C
C     SETS INITIAL CONDITIONS
C
      IF(N.LE.1.OR.XS.LE.0.0.OR.YS.LE.0.0) GO TO 140
      L=2
      J=1
      CALL PENTO(1,X(1),Y(1))
      IF(B1.LT.EPS.OR.B2.LT.EPS.OR.D1.LT.EPS.OR.D2.LT.EPS) GO TO 70
      M=1
      XSS=XS*XS
      YSS=YS*YS
C
C     CALCULATES TOTAL LENGTH OF DASHED LINE
C
      D=D1+B1+D2+B2
      W=0.0
   10 I=J
   20 IF(J.EQ.N) GO TO 30
      J=J+1
      U=X(J)-X(I)
      V=Y(J)-Y(I)
      U=SQRT(U*U*XSS+V*V*YSS)
      IF(U.LT.EPS.AND.J.NE.N) GO TO 20
      W=W+U
      GO TO 10
C
C     ADJUSTS LENGTHS OF DASHES AND BLANKS SO THAT THE LAST
C     JOIN WILL FINISH WITH A COMPLETE DASH.
C
```

```
 30 IF(W.LT.EPS) GO TO 130
    J=W/D
    V=FLOAT(J)*D
    U=W-V
    IF(U.LE.D1) GO TO 50
    IF(U.GT.(D-B2)) GO TO 40
    V=V+D1
    Z=B1+D2
    GO TO 60
 40 V=V+D
 50 V=V-B2
    Z=B2+D1
 60 U=(V-W)*(V+W)+V*Z
    IF(U.LT.0.0) V=V+Z
    W=W/V
    DL(1)=D1*W
    DL(2)=B1*W
    DL(3)=D2*W
    DL(4)=B2*W
    D=DL(1)
    J=1
    GO TO 80
 70 DL(1)=0.0

C
C     ENTERS PLOTTING SECTION.
C
    80 I=J
    90 IF(J.EQ.N) GO TO 130
       J=J+1
       IF(DL(1).LT.EPS) GO TO 120
       U=X(J)-X(I)
       V=Y(J)-Y(I)
       W=SQRT(U*U*XSS+V*V*YSS)
       IF(W.LT.EPS) GO TO 90
       DINTX=U/W
       DINTY=V/W
       W=W*W
C
C      CALCULATE INCREMENTS TO COMPLETE DASH OR BLANK NOT
C      FINISHED IN LAST JOIN.
C
       DX=DINTX*D
       DY=DINTY*D
  100  IF((DX*DX*XSS+DY*DY*YSS).GE.W) GO TO 110
       XX=X(I)+DX
       YY=Y(I)+DY
       CALL PENTO(L,XX,YY)
       L=3-L
       M=M+1
       IF(M.EQ.5) M=1
       DX=DX+DINTX*DL(M)
       DY=DY+DINTY*DL(M)
       GO TO 100
C
C      CALCULATES LENGTH OF UNFINISHED DASH OR BLANK.
C
  110  U=X(I)-X(J)+DX
       V=Y(I)-Y(J)+DY
       D=SQRT(U*U*XSS+V*V*YSS)
C
C      COMPLETES PRESENT JOIN.
C
  120  CALL PENTO(L,X(J),Y(J))
       GO TO 80
  130  CALL PENTO(2,X(N),Y(N))
  140  RETURN
       END
```

Algorithms supplement—Statement of Policy

A contribution to the Supplement may consist of an Algorithm, a Note on a previous algorithm, or an item under the heading of Correspondence.

Because the aim is to facilitate the interchange of algorithms, these should normally be submitted in one of the standard high level programming languages, namely ALGOL 60 (1), ALGOL 68 (2), FORTRAN (3), COBOL (4). In this case the algorithms must conform to the appropriate standard. If algorithms are submitted in other programming languages, the reference document for that language must be stated.

Algorithms must be self-contained. This means that an algorithm must consist of one or more complete segments, and that an algorithm must not use any non-local identifiers other than standard function names. COMMON areas are permitted in FORTRAN, but their use must be clearly described.

The algorithm must be written for publication in the appropriate reference language, and preceded by an appropriate Author's Note. It must be submitted in duplicate and be typewritten double-spaced. Where material is to appear in bold face it should be underlined in black. Where the appropriate character does not exist on a typewriter, it should be inserted neatly by hand in black and not be replaced by a similar composite character (e.g. $\leqslant$ should not be inserted as $\leq$).

An algorithm must be accompanied by a computer printout of a driver program testing it (possibly against test data) and producing test results. The machine, compiler and operating system used should be indicated. A computer readable copy of the algorithm, the test driver and any test data will be requested later, but should not be sent in the first instance. The Author's Note should include the theory of the method, with references, and also explain any tests used to verify the algorithm.

The algorithm must be syntactically correct, produce the results claimed and use computer resources as efficiently as possible. Constructions whose results may depend on the compiler used should be avoided (e.g. $y := x + f(x)$ where $f(b)$ is a function which alters the value of b). Comments should be used wherever appropriate to clarify the logic. Cases of failure should be clearly anticipated and handled. Approximate numerical constants must be given with as much accuracy as is appropriate. Numerical labels should be in ascending order.

Every effort is made to see that published algorithms are completely reliable. In particular all algorithms are submitted to independent referees and extensively checked. However, Notes or Correspondence which point out defects in or suggest improvements to previously published algorithms are welcomed. To help in preventing printing mistakes, galley proofs will be sent to authors where possible.

Whilst every effort is made to publish correct algorithms, no liability is assumed by any contributor, the Editor or *The Computer Journal* in connection therewith.

The copyright of all published algorithms remains with *The Computer Journal*. Nevertheless the reproduction of algorithms is explicitly permitted without charge provided that where the algorithm is used in another publication, reference is made to the author and to *The Computer Journal*.

In the event of the formation of a National Algorithm Library, all algorithms which have appeared in *The Computer Journal* will be made available to this Library.

References

1. PROGRAMMING LANGUAGE ALGOL, ISO/R/1538.
2. REVISED REPORT ON THE ALGORITHMIC LANGUAGE ALGOL 68. (1976). Edited by A. van Wijngaarden *et al*, Springer-Verlag.
3. PROGRAMMING LANGUAGE FORTRAN, ISO/R/1539.
4. PROGRAMMING LANGUAGE COBOL, ISO/R/1989.

Documents 1, 3 and 4 above may be obtained from: British Standards Institution, Sales Branch, 101 Pentonville Road, London N1.

Editors: P. A. Samet and A. C. Day, Computer Centre, University College London, 19 Gordon Street, London WC1H 0AH. Tel: 01-387 0858.