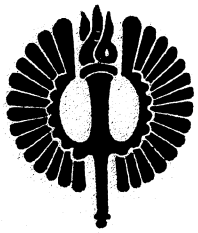


AN IMPLEMENTATION OF AN ALGORITHM FOR FINDING MINIMAL SPANNING TREES IN A EUCLIDEAN COORDINATE SPACE

Jyrki Katajainen
University of Turku
Department of mathematical sciences
Computer science
SF-20500 Finland

13.8.1981
Report D22

SARJAT:
A: Raportit
B: Käsikirjoitukset
C: Luentomonisteet
D: Käsikirjat

SERIES:
A: Reports
B: Manuscripts
C: Lecture Notes
D: Manuals

TURUN YLIOPISTO
MATEMAATTISTEN TIETEIDEN LAITOS
TIETOJENKÄSITTELYOPPI
20500 TURKU 50

UNIVERSITY OF TURKU
DEPARTMENT OF MATHEMATICAL SCIENCES
COMPUTER SCIENCE
SF-20500 TURKU 50, FINLAND

1. Introduction

In general form the minimal spanning tree problem can be stated as follows. The minimal spanning tree connects all the points of a connected undirected graph such that the sum of the costs of the edges which connects these points is minimal. We shall consider a very special minimal spanning tree problem where the graph is situated in a k-dimensional Eucledean space. That means the graph is complete so that the number of the edges in the graph is $n(n-1)/2$ and the cost of an edge is the distance between the end points.

This paper presents an implementation of an algorithm for finding a minimal spanning tree in a Euclidean space studied more deeply in [5]. The main ideas of the algorithm can be found from the algorithms of Cheriton and Tarjan [3] and Bentley and Friedman [2]. The purpose was to produce an algorithm which is simpler than but as efficient as that of Bentley and Friedman. It has been shown experimentally that the algorithm works in the expected $O(n \log^2 n)$ time when n is the number of the points in a graph. However, there is a point set for which the running time is $O(n^2)$ [5].

Our minimal spanning tree algorithm can be described without details of implementation as follows.

```
for i:=1 to n do T_i:={i}; (* The points are numbered 1 through n *)
F:={T_1,T_2,...,T_n}; (* F is a forest of subtrees of the graph *)
                      (* Note that T_i is a set of edges and points *)
for i:=1 to n-1 do
   begin
      select a tree T_k from the forest F;
      find the shortest edge (u,u') which connects the points uЄT_k and
           u'ЄT'ЄF-{T_k};
      merge the tree T_k into the tree T' with edge (u,u');
   end.
```

In the selection phase we always select from the forest F the tree for which the number of the points is minimal. To find the shortest distance (u,u') we are going to use a k-d tree like Bentley and Friedman.

If we study the algorithm carefully we find that the following data structures must be maintained: a) a k-d tree, b) a priority queue of the sizes of the trees in F, c) a structure for the points of the trees T_i and d) for the edges of the tree T_i. In addition we do not want to compute the distance outside the tree T_k for every point of the tree T_k. Thus we maintain e) a priority queue for every T_i. The priority means the same as the distance between the point and its closest point so far. When the minimal priority of

the tree T_k refers to a point in that tree we recompute the priority by means of a k-d tree and update the priority queue. This has to be repeated until the minimal priority refers outside the tree T_k. The priority queues must be implemented such that the MINIMUM, DELETEMINIMUM, INSERT and UNION instructions can be executed efficiently. Thus we have implemented the queues as 2-3 trees [1, pp. 145-157].

2. The data structures

2.1 k-d tree

Let us consider the k-d tree as in [4]. In Figure 1a there is a 2-d tree. If we suppose that the coordinate values of the points used to construct the tree are integers between 0 and 100, then Figure 1b shows how the space is split by the tree, for example all the points, of which the first coordinate value is between 60 and 100 and the second between 0 and 40, belong to the third bucket.

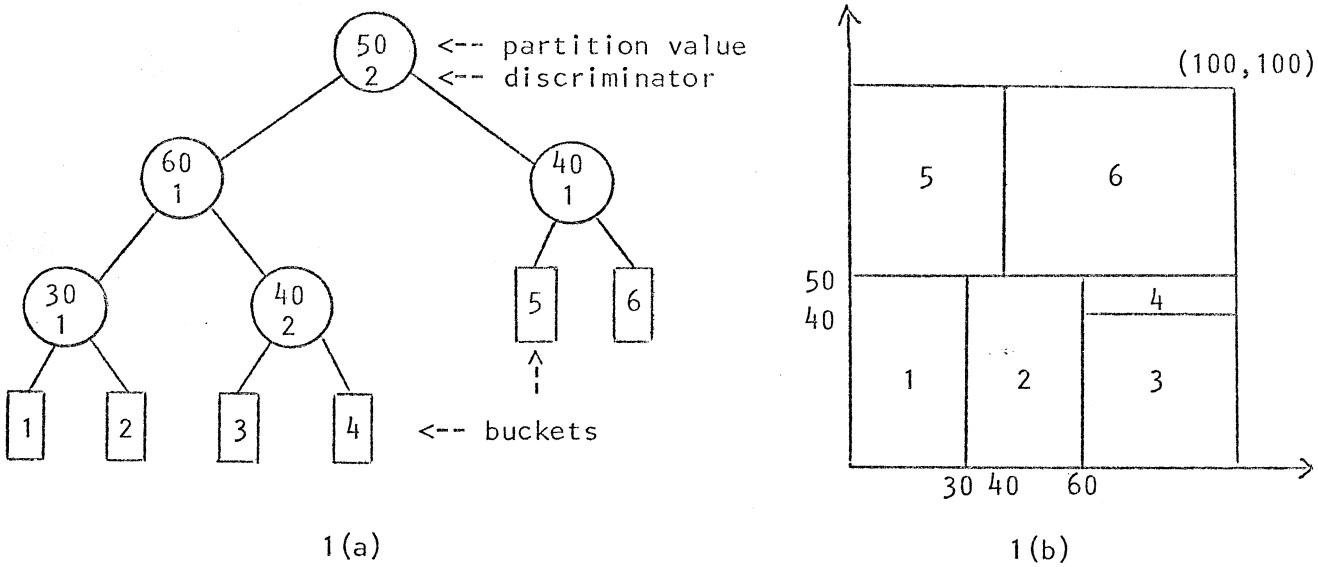


Fig. 1. (a) A 2-d tree and (b) the space is split by the 2-d tree.

The building procedure and the searching procedure for a k-d tree can be found directly in [4]. We only had to make the algorithms iterative because we program them with FORTRAN. In principle a k-d tree is build so that the discriminator of the root is the coordinate in which the range of the coordinate values is largest. As the partition value we choose the coordinate value whose index in the ascending order is $\max\{h,n-h\}$ where h is the integer of form 2^l closest to $\lceil n/2 \rceil$. The points having a discriminating coordinate value less than the partition value belong to the right subtree. The point the discriminating coordinate value of which is equal to the partition value can be found either in the left or in the right subtree.

The same procedure is continued recursively with both subtrees until the number of the points is less than some set bucket size. This produces an unbalanced, but complete, tree. In the building procedure we have used the program described in [1, p.102], which selects the i th value of a given point set.

The searching procedure is also easy to describe recursively. First we shall go to examine the subtree in which the query point lies. After that we shall examine the other subtree if it is possible that there is a point closer to the query point than the nearest point so far. Either at once or after examining the other subtree we shall test if it is at all possible that in the tree there will still be a point closer than the closest point so far. If not, we shall stop the search. Otherwise we shall have to continue. When we are searching the closest point we are not allowed to take into account the points which belong to the same tree T_k of the forest F.

The building procedure used makes it possible to implement a k-d tree as a heap. So we use two arrays KDISC and PARTI: in both the sons of the node i will be found at 2i and 2i+1 [1, pp. 87-88]. We have implemented the buckets as an array KBUCK. For example, if the bucket size is 8, the locations 1-8 of the array contain the numbers of the points in the first bucket and the locations 9-16 the numbers of the points in the second bucket and so on.

2.2 The priority queue of the sizes of the subtrees

We combine the trees T_i by using the minimal size selection rule of Cheriton and Tarjan [3]. In Figure 2 we see the arrays used to implement the priority queue of the sizes of the subtrees. In LIST(m) there is a pointer to the first tree T_i of size m. LINKS(i,1) consists of a pointer to the next tree T_j of the same size. LINKS(i,2) points back to the preceding tree. We use the two-way chaining, which facilitates the updating of the queue. In addition, we maintain one array IFSIZE for the sizes of the trees T_i.

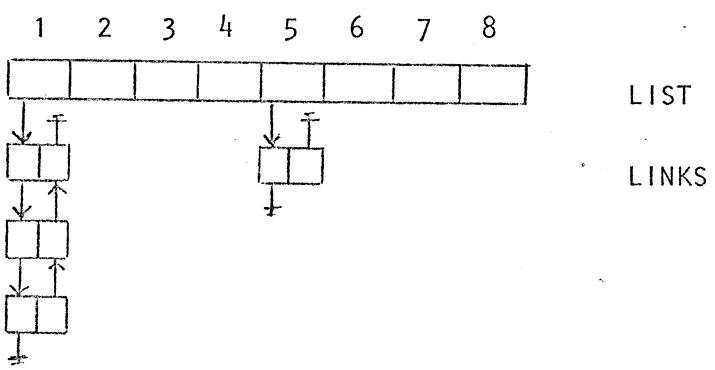
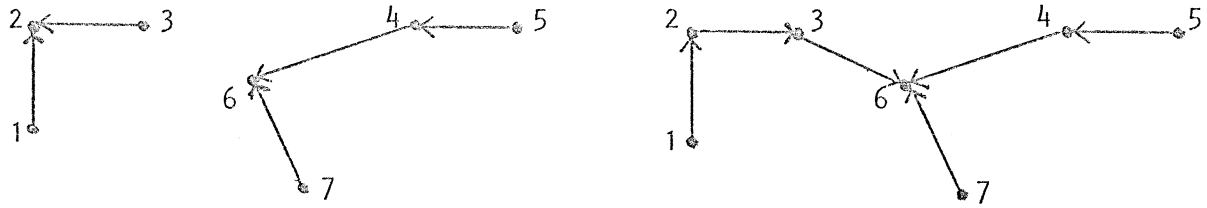


Fig. 2. The priority queue of the sizes of the subtrees when we have three subtrees of size one and one subtree of size five.

2.3 Structures for the trees T_i

The points of the subtrees are chained. The start address for the chain of the tree T_i is in IFIRST(i). The chaining goes on in NODES(j,3) for $j \leq n$. The main purpose of the matrix will be explained later.

The edges of the subtrees are implemented by the father links in an array MST. In Figure 3 we see how two subtrees are merged. So some links of the smaller subtree must be inverted during the merging.



3(a) 3(b)

Fig. 3. The merging of the edges of two subtrees, 3a before and 3b after the merging.

2.4 The priority queues

As mentioned above, the priority queues of the subtrees are 2-3 trees, which we have implemented using a matrix NODES and an array PRIOR. For any j NODES(j,1), NODES(j,2) and NODES(j,3) point to the sons of the node, NODES(j,4) points to the father of the node. In PRIOR(j) there is the priority of the node. The algorithms of the instructions MINIMUM, DELETEMINIMUM, INSERT and UNION are described in [1, pp. 148-157]. We only had to make the recursive algorithms iterative. The root of the priority queue of the tree T_i is found in NROOT(i).

Since the pointers of the leaves of a 2-3 tree are not needed, we have made use of those locations. For $j \leq n$ NODES(j,2) contains the number of the closest point for the point j, NODES(j,2) tells to which subtree the point j belongs and NODES(j,3) is the pointer for chaining the points of the subtree NODES(j,2).

3. Instructions

The program THEMST finds a minimal spanning tree for a point set in k-dimensional Euclidean space. The form of the program call is

```
CALL THEMST(n,n*2,⌈n/KBSIZE⌉-1,⌈log2⌈n/KBSIZE⌉⌉,k,ABSMAX,POINT,MST,
            CMST,KBUCK,KDISC,PARTI,KBSIZE,BLOW,BHIGH,ISTACK,TEMP,
            LIST,LINKS,IFSIZE,IFIRST,NODES,PRIOR,NROOT).
```

The points must be in the matrix POINT before calling the program. ABSMAX is the greatest absolute value for the coordinate of the points. The minimal spanning tree is given with father links in the array MST and the cost of the spanning tree is CMST. The value 8 has been observed to be good as the bucket size of the k-d tree if the number of the points is about one thousand [5]. However, if the point set is smaller, a greater value for the bucket size, for example 16, could give better running times. For a greater point set a smaller value for the bucket size could be better. In the main program the variable KBSIZE must be initialized and the third and fourth parameters for THEMST must be computed by using this KBSIZE value. The arrays BLOW, BHIGH, ISTACK and TEMP are temporaries needed during the k-d tree searching.

In the main program enough space must be reserved for the data areas of the program THEMST. With the following sample program we find the cost of the minimal spanning tree for a randomly generated point set.

```
      DIMENSION POINT(68,10)
      DIMENSION KBUCK(68),KDISC(4),PARTI(4)
      DIMENSION BLOW(10),BHIGH(10),ISTACK(0/3),TEMP(0/3)
      DIMENSION LIST(68),LINKS(68,2)
      DIMENSION IFSIZE(68),IFIRST(68)
      DIMENSION NODES(136,4),PRIOR(136),NROOT(68)
      DIMENSION MST(68)
C-- RANDOM NUMBERS BETWEEN 0 AND 1000.0 ARE USED FOR CONSTRUCTING THE POINT S
      DO 10 I=1,68
      DO 10 J=1,10
10    POINT(I,J)=1000.0*RAN(Z)

      CALL THEMST
     *(68,136,4,3,10,1000.0,POINT,MST,CMST,KBUCK,KDISC,PARTI,16,BLOW,
     *BHIGH,ISTACK,TEMP,LIST,LINKS,IFSIZE,IFIRST,NODES,PRIOR,NROOT)

      TYPE *,CMST
      CALL EXIT
      END
```

4. Conclusions

We have to admit that our program is comparatively long and it needs a lot of memory. The length of the arrays KBUCK, LIST, IFSIZE, IFIRST, NROOT and MST must be n, the length of KDISC and PARTI is $\lceil n/KBSIZE \rceil - 1$ and the length of PRIOR is 2n. POINT is an n×k matrix and LINKS n×2 and NODES 2n×4. The total memory required is therefore at most (k+3)n real numbers and 17n integers.

In the program THEMST there must be subroutines for building a k-d tree (BUILD), searching by means of a k-d tree (SEARCH), merging two 2-3 trees (UNION), deleting the minimum and inserting into a 2-3 tree (NEXT), merging two point sets (MERGE), merging two edge sets (EDGE), selecting the subtree of minimal size (MIN) and updating the queue of the sizes of the

subtrees (UPDATE). The whole program is in the Appendix. In all the program takes about 22 kilos of words in a DEC-10 computer.

References

[1] Alfred Aho, John Hopcroft and Jeffrey Ullman, "The Design and Analysis of Computer Algorithms", Addison-Wesley, 1974.

[2] Jon Louis Bentley and Jerome H. Friedman, "Fast Algorithms for Constructing Minimal Spanning Trees in Coordinate Spaces", IEEE Trans. on Computers, Vol.C-27, No. 2, February 1978.

[3] David Cheriton and Robert Endre Tarjan, "Finding Minimal Spanning Trees", SIAM J. Computing, Vol. 5, No. 4, December 1976.

[4] Jerome H. Friedman, Jon Louis Bentley and Raphael Ari Finkel, "An Algorithm for Finding Best Matches in Logarithmic Expected Time", ACM Transactions on Mathematical Software, Vol. 3, No.3, September 1977.

[5] Olli Nevalainen, Jarmo Ernvall and Jyrki Katajainen, "Finding Minimal Spanning Trees in a Euclidean Coordinate Space", BIT 21, 1981, pp. 46-54.

```
C- THIS PROGRAM DETERMINES A MINIMAL SPANNING TREE FOR A POINT SET
C- OF K-DIMENSIONAL EUCLIDEAN SPACE.
C- POINTS ARE GIVEN IN THE ARRAY POINT AND THE NUMBER OF POINTS IS N.

      SUBROUTINE THEMST
     *(N,NX2,NPER,LOGN,K,ABSMAX,POINT,MST,CMST,KBUCK,KDISC,PARTI,KBSIZE,
     *BLOW,BHIGH,ISTACK,TEMP,LIST,LINKS,IFSIZE,IFIRST,NODES,PRIOR,NROOT)

      DIMENSION POINT(N,K)
      DIMENSION KBUCK(N),KDISC(NPER),PARTI(NPER)
      DIMENSION BLOW(K),BHIGH(K),ISTACK(0/LOGN),TEMP(0/LOGN)
      DIMENSION LIST(N),LINKS(N,2)
      DIMENSION IFSIZE(N),IFIRST(N)
      DIMENSION NODES(NX2,4),PRIOR(NX2),NROOT(N)
      DIMENSION MST(N)

C- BUILD AN OPTIMIZED K-D TREE FOR THE POINT SET

      CALL BUILD(N,NX2,NPER,K,POINT,KBUCK,KDISC,PARTI,KBSIZE,PRIOR)

C- INITIALIZE OTHER DATASTRUCTURES NEEDED

      CALL INIT(N,NX2,IFSIZE,IFIRST,NROOT,NODES,PRIOR,LIST,LINKS,
     *          MST,CMST)

C- FORM THE MINIMAL SPANNING TREE

      NOW=0
      NFREE=N+1
      DO 1 I=1,N-1

C- SELECT A SUBTREE WITH MINIMAL SIZE

      CALL MIN(NOW,LITTLE,N,LIST,LINKS)

C- FIND THE CLOSEST SUBTREE OF THE SUBTREE LITTLE

      IX=NEXT(LITTLE,N,NX2,NPER,LOGN,K,ABSMAX,POINT,NODES,PRIOR,NROOT,
     *        KBUCK,KDISC,PARTI,KBSIZE,BLOW,BHIGH,ISTACK,TEMP)
      IY=NODES(IX,1)
      NEIRST=NODES(IY,2)

C- MERGE THE PRIORITY QUEUE OF LITTLE WITH THE PRIORITY QUEUE OF NEIRST

      CALL UNION(LITTLE,NEIRST,NFREE,N,NX2,NODES,PRIOR,NROOT)

C- UPDATE THE PRIORITY QUEUE OF THE SIZES OF THE SUBTREES

      CALL UPDATE(LITTLE,NEIRST,N,LIST,LINKS,IFSIZE)

C- MERGE THE SUBTREE LITTLE INTO THE SUBTREE NEIRST WITH EDGE (IX,IY)

      CALL MERGE(IX,IY,N,NX2,IFSIZE,IFIRST,NODES)

C- UPDATE THE MST AND CMST

      CMST=CMST+PRIOR(IX)
      CALL EDGE(IX,IY,N,MST)
1     CONTINUE
      RETURN
      END
```

```
C- THIS PROGRAM FORMS AN OPTIMIZED K-D TREE LIKE A HEAP

      SUBROUTINE BUILD(N,NX2,NPER,K,POINT,KBUCK,KDISC,PARTI,KBSIZE,S)
      DIMENSION POINT(N,K)
      DIMENSION KBUCK(N),KDISC(NPER),PARTI(NPER)
      DIMENSION S(NX2)

      DO 10 I=1,N
10    KBUCK(I)=I
      KDISC(1)=NPER+1
      I=0
      J=0
20    IF(J.GE.NPER) RETURN
      J=J+1
      IF(J.NE.2**I) GOTO 30
      LAST=0
      I=I+1
30    LFIRST=LAST+1
      LAST=MINO(N,LAST+KDISC(J)*KBSIZE)

      MEDIUM=INT(KDISC(J)/2.0)
      IF(KDISC(J)/2.0-FLOAT(MEDIUM).NE.0) MEDIUM=MEDIUM+1
      IA=INT(ALOG(FLOAT(MEDIUM))/ALOG(2.0))
      IH=2**IA
      IF(2*IH-MEDIUM.LT.MEDIUM-IH) IH=2*IH
      IA=MAXO(IH,KDISC(J)-IH)
      IF(2*J.LE.NPER) KDISC(2*J)=IA
      IF(2*J+1.LE.NPER) KDISC(2*J+1)=KDISC(J)-IA

      SPR=0.0
      DO 40 IK=1,K
      IKTEMP=IK
      SD=SPREAD(IKTEMP,LFIRST,LAST,KBUCK,N,K,POINT)
      IF(SD.LT.SPR) GOTO 40
      SPR=SD
      ID=IK
40    CONTINUE

      KDISC(J)=ID
      PARTI(J)=REMOVE(IA*KBSIZE,ID,LFIRST,LAST,N,NX2,K,KBUCK,POINT,S)
      GOTO 20
      END


C- DETERMINE THE SPREAD OF POINTS IN DIMENSION J BETWEEN LFIRST AND LAST

      FUNCTION SPREAD(J,LFIRST,LAST,KBUCK,N,K,POINT)
      DIMENSION KBUCK(N),POINT(N,K)

      SMALL=POINT(KBUCK(LFIRST),J)
      BIG=POINT(KBUCK(LFIRST),J)
      DO 10 I=LFIRST+1,LAST
      IF(POINT(KBUCK(I),J).LT.SMALL) SMALL=POINT(KBUCK(I),J)
      IF(POINT(KBUCK(I),J).GT.BIG) BIG=POINT(KBUCK(I),J)
10    CONTINUE
      SPREAD=BIG-SMALL
      RETURN
      END
```

```
C- PUT IHALF SMALLEST VALUES IN DIMENSION ID OF THE INTERVAL TO THE LEFT

      FUNCTION REMOVE(IHALF,ID,LFIRST,LAST,N,NX2,K,KBUCK,POINT,S)
      DIMENSION KBUCK(N),POINT(N,K),S(NX2)

      DO 10 I=LFIRST,LAST
10    S(I-LFIRST+1)=POINT(KBUCK(I),ID)
      REMOVE=SELECT(NX2,S,IHALF,LAST-LFIRST+1)

      IEND=LAST
      IWARN=0
      DO 20 IBEGIN=LFIRST,LFIRST+IHALF-1
      IF(POINT(KBUCK(IBEGIN),ID).LE.REMOVE) GOTO 20
15    IF(IWARN.EQ.1) GOTO 30
      IF(POINT(KBUCK(IEND),ID).LT.REMOVE) GOTO 40
      IEND=IEND-1
      IF(IEND.GE.LFIRST+IHALF) GOTO 15

C- IF IWARN=1 THE END OF THE INTERVAL HAVE NOT A VALUE SMALLER THAN IHALF:TH

      IWARN=1
      IEND=LAST
      GO TO 15

30    IF(POINT(KBUCK(IEND),ID).LE.REMOVE) GOTO 40
      IEND=IEND-1
      GO TO 30

40    L=KBUCK(IBEGIN)
      KBUCK(IBEGIN)=KBUCK(IEND)
      KBUCK(IEND)=L
20    CONTINUE

      IF(IWARN.EQ.1) RETURN
      IBEGIN=LFIRST
60    IF(POINT(KBUCK(IEND),ID).LT.REMOVE) GOTO 70
61    IEND=IEND-1
      IF(IEND.LT.LFIRST+IHALF) RETURN
      GOTO 60
70    IF(POINT(KBUCK(IBEGIN),ID).GE.REMOVE) GOTO 80
      IBEGIN=IBEGIN+1
      GOTO 70
80    L=KBUCK(IBEGIN)
      KBUCK(IBEGIN)=KBUCK(IEND)
      KBUCK(IEND)=L
      GOTO 61
      END
```

```
C= SELECT THE KK:TH HIGHEST VALUE OF POINTS OF S

      FUNCTION SELECT(NX2,S,KK,MM)
      DIMENSION S(NX2)

      M=MM
      K=KK
1     IF(1.GT.1) GOTO 10
      SELECT=S(1)
      RETURN

10    I1=0
      I2=0
      I3=0
      DO 20 I=1,M
      IF(S(I).GT.S(K)) GOTO 30
      IF(S(I).LT.S(K)) GOTO 35

      I2=I2+1
      GOTO 20

35    I1=I1+1
      S(M+I1)=S(I)
      GOTO 20

30    I3=I3+1
      S(2*M-I3+1)=S(I)
20    CONTINUE

      IF(I1.LT.K) GOTO 40

      DO 50 I=1,I1
50    S(I)=S(M+I)
      M=I1
      GOTO 1

40    IF(I1+I2.LT.K) GOTO 70
      SELECT=S(K)
      RETURN

70    K=K-I1-I2
      DO 80 I=1,I3
80    S(I)=S(2*M-I+1)
      M=I3
      GOTO 1
      END
```

```
      SUBROUTINE INIT(N,NX2,IFSIZE,IFIRST,NROOT,NODES,PRIOR,LIST,
     *                LINKS,MST,COST)
      DIMENSION IFSIZE(N),IFIRST(N),NROOT(N),NODES(NX2,4),PRIOR(NX2)
      DIMENSION LIST(N),LINKS(N,2),MST(N)

C-- INITIALLY EACH SUBTREE CONSISTS OF A SINGLE POINT

      DO 20 I=1,N
      IFSIZE(I)=1
20    IFIRST(I)=I

C-- INITIALLY EACH POINT HAVE ZERO PRIORITIES

      DO 30 I=1,N
      NROOT(I)=I
      NODES(I,1)=1
      NODES(I,2)=I
      NODES(I,3)=-1
      NODES(I,4)=-1
30    PRIOR(I)=0.0
      DO 40 I=N+1,2*N
      DO 50 J=1,4
50    NODES(I,J)=-1
40    PRIOR(I)=0.0

C-- FORM A PRIORITY QUEUE OF THE SIZES

      DO 60 I=1,N
      LIST(I)=-1
      LINKS(I,1)=I-1
60    LINKS(I,2)=I+1
      LINKS(1,1)=-1
      LINKS(N,2)=-1
      LIST(1)=1

C-- INITIALLY THE MINIMAL SPANNING TREE IS EMPTY

      DO 70 I=1,N
70    MST(I)=-1
      COST=0.0
      RETURN
      END
```

```
      SUBROUTINE NEW(IPLACE,LINK,N,LIST,LINKS)
      DIMENSION LIST(N),LINKS(N,2)

      IF(IPLACE.EQ.0) GOTO 20

10    LINK=LINKS(LINK,2)
      IF(LINK.EQ.-1) GOTO 20
      RETURN

20    IPLACE=IPLACE+1
      IF(LIST(IPLACE).EQ.-1) GOTO 20
      LINK=LIST(IPLACE)
      RETURN
      END



      FUNCTION NEXT(LITTLE,N,NX2,NPER,LOGN,K,ABSMAX,POINT,NODES,PRIOR,
     *              NROOT,KNICK,KDISC,PARTI,KBSIZE,BLOW,BHIGH,ISTACK,TEMP)
      DIMENSION POINT(N,K)
      DIMENSION KNICK(N),KDISC(NPER),PARTI(NPER)
      DIMENSION BLOW(K),BHIGH(K),ISTACK(0/LOGN),TEMP(0/LOGN)
      DIMENSION NODES(NX2,4),PRIOR(NX2),NROOT(N)

      L=NROOT(LITTLE)
2     IF(L.LE.N) GOTO 100
      DO 10 I=1,3
      J=NODES(L,I)
      IF(PRIOR(J).EQ.PRIOR(L)) GOTO 20
10    CONTINUE
      STOP 'STOP 1'
20    L=NODES(L,I)
      GOTO 2

100   IF(LITTLE.EQ.NODES(NODES(L,1),2)) GOTO 200
      NEXT=L
      RETURN

C- THE PRIORITY OF THE TOP NODE POINTS TO THE SAME SUBTREE

200   CALL SEARCH(L,NODES(L,1),PRIOR(L),LITTLE,N,NX2,NPER,LOGN,K,
     *            ABSMAX,POINT,KNICK,KDISC,PARTI,KBSIZE,BLOW,BHIGH,
     *            ISTACK,TEMP,NODES)

25    IF(L.EQ.NROOT(LITTLE)) GOTO 2
      L=NODES(L,4)
      IF(NODES(L,3).EQ.-1) GOTO 30
      PRIOR(L)=AMIN1(PRIOR(NODES(L,1)),PRIOR(NODES(L,2)),
     *               PRIOR(NODES(L,3)))
      GOTO 25
30    PRIOR(L)=AMIN1(PRIOR(NODES(L,1)),PRIOR(NODES(L,2)))
      GOTO 25
      END
```

```
      SUBROUTINE SEARCH(NP,NEAR,DIST,INTNTS,N,NX2,NPER,L,K,ABSMAX,
     *     POINT,KBUCK,KDISC,PARTI,KBSIZE,BLOW,BHIGH,ISTACK,TEMP,NODES)
      DIMENSION POINT(N,K)
      DIMENSION KBUCK(N),KDISC(NPER),PARTI(NPER)
      DIMENSION BLOW(K),BHIGH(K),ISTACK(0/L),TEMP(0/L)
      DIMENSION NODES(NX2,4)

      DIST=2*ABSMAX*SQRT(FLOAT(K))
      DO 10 I=1,K
      BLOW(I)=-(ABSMAX+DIST)
10    BHIGH(I)=ABSMAX+DIST
      DO 20 I=0,L
      ISTACK(I)=-1
20    TEMP(I)=-1

      I=0
      ISTACK(I)=1
100   NODE=ISTACK(I)

C-- ARE WE IN THE LEAF LEVEL

      IF(NODE.LE.NPER) GOTO 110
      CALL EXAM(KBSIZE*MOD(NODE+NPER+1-2**L,NPER+1)+1,NP,NEAR,DIST,
     *          INTNTS,N,NX2,K,POINT,KBUCK,KBSIZE,NODES)
      I=I-1
      GOTO 100

C-- HAVE WE BEEN ALREADY IN THE CLOSER SON

110   IF(ISTACK(I+1).EQ.2*NODE.OR.ISTACK(I+1).EQ.2*NODE+1) GOTO 200
      IF(POINT(NP,KDISC(NODE)).GT.PARTI(NODE)) GOTO 120

C-- THE CLOSER SON IS IN THE LEFT

      TEMP(I)=BHIGH(KDISC(NODE))
      BHIGH(KDISC(NODE))=PARTI(NODE)
      I=I+1
      ISTACK(I)=2*NODE
      GOTO 100

C-- THE CLOSER SON IS IN THE RIGHT

120   TEMP(I)=BLOW(KDISC(NODE))
      BLOW(KDISC(NODE))=PARTI(NODE)
      I=I+1
      ISTACK(I)=2*NODE+1
      GOTO 100

C-- HAVE WE BEEN IN THE FARTHER SON

200   IF(POINT(NP,KDISC(NODE)).GT.PARTI(NODE).AND.
     *     ISTACK(I+1).EQ.2*NODE.OR.
     *     POINT(NP,KDISC(NODE)).LE.PARTI(NODE).AND.
     *     ISTACK(I+1).EQ.2*NODE+1) GOTO 300

C-- IS THE FARTHER SON IN THE LEFT OR IN THE RIGHT

      IF(2*NODE.NE.ISTACK(I+1)) GOTO 210
      BHIGH(KDISC(NODE))=TEMP(I)
      TEMP(I)=BLOW(KDISC(NODE))
      BLOW(KDISC(NODE))=PARTI(NODE)
      ISTACK(I+1)=2*NODE+1
      IF(IBOUND(NP,DIST,BLOW,BHIGH,N,K,POINT).EQ.1) GOTO 300
      I=I+1
      GOTO 100

210   BLOW(KDISC(NODE))=TEMP(I)
      TEMP(I)=BHIGH(KDISC(NODE))
      BHIGH(KDISC(NODE))=PARTI(NODE)
      ISTACK(I+1)=2*NODE
      IF(IBOUND(NP,DIST,BLOW,BHIGH,N,K,POINT).EQ.1) GOTO 300
      I=I+1
      GOTO 100

C-- LOOK IF WE CAN STOP THE SEARCH

300   IF(2*NODE.EQ.ISTACK(I+1)) GOTO 310
      BLOW(KDISC(NODE))=TEMP(I)
      GOTO 320
310   BHIGH(KDISC(NODE))=TEMP(I)
320   IF(IBALL(NP,DIST,BLOW,BHIGH,N,K,POINT).EQ.1) RETURN
      I=I-1
      GOTO 100
      END
```

```
      SUBROUTINE EXAM(KFIRST,NP,NEAR,DIST,INTHIS,N,NX2,K,POINT,KBUCK,
     *                KBSIZE,NODES)
      DIMENSION POINT(N,K), KBUCK(N), NODES(NX2,4)

      KEND=MINO(N,KFIRST+KBSIZE-1)
      DO 10 I=KFIRST,KEND

C- THE POINT IS REJECTED IF IT BELONGS TO THE SUBTREE INTHIS

      IF(NODES(KBUCK(I),2).EQ.INTHIS) GOTO 10

C- IS THE POINT CLOSER THAN POINTS BEFORE

      SUM=0
      DO 20 J=1,K
20    SUM=SUM+(POINT(NP,J)-POINT(KBUCK(I),J))**2
      IF(DIST.LE.SQRT(SUM)) GOTO 10
      DIST=SQRT(SUM)
      NEAR=KBUCK(I)
10    CONTINUE
      RETURN
      END



C- DO THE BALL WHOSE RADIUS IS DIST AND MIDDLE IS QUERY INTERSECT THE BOX
C- WHICH IS IN BOUNDS BLOW AND BHIGH

      FUNCTION IBOUND(NP,DIST,BLOW,BHIGH,N,K,POINT)
      DIMENSION POINT(N,K),BLOW(K),BHIGH(K)

      SUM=0
      DO 10 I=1,K
      IF(POINT(NP,I).GE.BLOW(I)) GOTO 20

C- LOWER THAN LOW BOUNDARY

      SUM=SUM+(POINT(NP,I)-BLOW(I))**2
      IF(SQRT(SUM).GT.DIST) GOTO 50
      GOTO 10

20    IF(POINT(NP,I).LE.BHIGH(I)) GOTO 10

C- HIGHER THAN HIGH BOUNDARY

      SUM=SUM+(POINT(NP,I)-BHIGH(I))**2
      IF(SQRT(SUM).GT.DIST) GOTO 50
10    CONTINUE
      IBOUND=0
      RETURN
50    IBOUND=1
      RETURN
      END



C- IS THE BALL WHOSE RADIUS IS DIST AND MIDDLE IS QUERY IN THE BOX
C- WHICH IS IN BOUNDS BLOW AND BHIGH

      FUNCTION IBALL(NP,DIST,BLOW,BHIGH,N,K,POINT)
      DIMENSION POINT(N,K),BLOW(K),BHIGH(K)

      DO 10 I=1,K
      IF(ABS(POINT(NP,I)-BLOW(I)).LE.DIST) GOTO 20
      IF(ABS(POINT(NP,I)-BHIGH(I)).LE.DIST) GOTO 20
10    CONTINUE
      IBALL=1
      RETURN
20    IBALL=0
      RETURN
      END
```

```
C- COMBINE THE PRIORITY QUEUES OF THE SUBTREES LITTLE AND NEIRST.
C- PRIORITY QUEUES ARE 2-3 TREES. NFREE IS FREE SPACE INDICATOR.

      SUBROUTINE UNION(LITTLE,NEIRST,NFREE,N,NX2,NODES,PRIOR,NROOT)
      DIMENSION NODES(NX2,4),PRIOR(NX2),NROOT(N)

      LD=IDEPTH(NROOT(LITTLE),N,NX2,NODES)
      ND=IDEPTH(NROOT(NEIRST),N,NX2,NODES)
      IF(LD.NE.ND) GOTO 10

C- THE DEPTH OF THE 2-3 TREES IS EQUAL

      L=NROOT(NEIRST)
      M=NROOT(LITTLE)
20    NODES(NFREE,1)=L
      NODES(NFREE,2)=M
      PRIOR(NFREE)=AMIN1(PRIOR(L),PRIOR(M))
      NODES(L,4)=NFREE
      NODES(M,4)=NFREE
      NROOT(NEIRST)=NFREE
      NFREE=NFREE+1
      RETURN

C- THE TREE OF NH IS DEEPER THAN THAT OF LH

10    LH=LITTLE
      NH=NEIRST
      IF(LD.LT.ND) GOTO 30
      LH=NEIRST
      NH=LITTLE
30    L=NROOT(NH)
      DO 40 I=1,IABS(ND-LD)
      J=3
      IF(NODES(L,3).EQ.-1) J=2
40    L=NODES(L,J)
      M=NROOT(LH)

C- TRY TO INSERT THE NEW NODE M INTO THE 2-3 TREE OF NH

100   IF(L.EQ.NROOT(NH)) GOTO 20
      IPAPA=NODES(L,4)
      IF(NODES(IPAPA,3).NE.-1) GOTO 50

C- IPAPA CAN ADOPT THE NODE M WITHOUT DIFFICULTIES

      NODES(IPAPA,3)=M
      NODES(M,4)=IPAPA
      NROOT(NEIRST)=NROOT(NH)
35    L=NODES(L,4)
      IF(L.EQ.-1) RETURN
      IF(NODES(L,3).EQ.-1) GOTO 36
      PRIOR(L)=AMIN1(PRIOR(NODES(L,1)),PRIOR(NODES(L,2)),
     *               PRIOR(NODES(L,3)))
      GOTO 35
36    PRIOR(L)=AMIN1(PRIOR(NODES(L,1)),PRIOR(NODES(L,2)))
      GOTO 35

C- WE HAVE TO TAKE A NEW NODE WHICH ADOPTS THE NODE M AND ONE OF IPAPA'S SONS

50    IF(PRIOR(IPAPA).EQ.PRIOR(L)) L=NODES(IPAPA,2)
      NODES(NFREE,1)=L
      NODES(NFREE,2)=M
      NODES(L,4)=NFREE
      NODES(M,4)=NFREE
      PRIOR(NFREE)=AMIN1(PRIOR(M),PRIOR(L))
      IF(L.EQ.NODES(IPAPA,2)) NODES(IPAPA,2)=NODES(IPAPA,3)
      NODES(IPAPA,3)=-1
      L=IPAPA
      M=NFREE
      NFREE=NFREE+1
      GOTO 100
      END

C- WHAT IS THE DEPTH OF THE PRIORITY QUEUE OF WHICH ROOT IS L

      FUNCTION IDEPTH(LTOP,N,NX2,NODES)
      DIMENSION NODES(NX2,4)

      IDEPTH=1
      L=LTOP
10    IF(L.LE.N) RETURN
      L=NODES(L,1)
      IDEPTH=IDEPTH+1
      GOTO 10
      END
```

```
C- UPDATE THE SIZE OF NEIRST IN THE PRIORITY QUEUE OF THE SIZES

      SUBROUTINE UPDATE(LITTLE,NEIRST,N,LIST,LINKS,IFSIZE)
      DIMENSION LIST(N),LINKS(N,2),IFSIZE(N)

C- REMOVE THE OLD NOTATION OF THE SIZE OF NEIRST FROM THE QUEUE

      IBEFO=LINKS(NEIRST,1)
      IAFTER=LINKS(NEIRST,2)
      IF(IBEFO.NE.-1) GOTO 30
      LIST(IFSIZE(NEIRST))=IAFTER
      IF(IAFTER.NE.-1) LINKS(IAFTER,1)=-1
      GOTO 50
30    LINKS(IBEFO,2)=IAFTER
      IF(IAFTER.NE.-1) LINKS(IAFTER,1)=IBEFO

C- ADD THE NEW SIZE OF NEIRST INTO THE QUEUE

50    NEW=IFSIZE(LITTLE)+IFSIZE(NEIRST)
      L=LIST(NEW)
      LIST(NEW)=NEIRST
      LINKS(NEIRST,1)=-1
      LINKS(NEIRST,2)=L
      IF(L.NE.-1) LINKS(L,1)=NEIRST
      RETURN
      END



      SUBROUTINE MERGE(IX,IY,N,NX2,IFSIZE,IFIRST,NODES)
      DIMENSION IFSIZE(N),IFIRST(N),NODES(NX2,4)

      NEIRST=NODES(IY,2)
      LITTLE=NODES(IX,2)
      L=IFIRST(LITTLE)
10    IF(NODES(L,3).EQ.-1) GOTO 20
      NODES(L,2)=NEIRST
      L=NODES(L,3)
      GOTO 10
20    NODES(L,2)=NEIRST
      NODES(L,3)=IFIRST(NEIRST)
      IFIRST(NEIRST)=IFIRST(LITTLE)
      IFSIZE(NEIRST)=IFSIZE(NEIRST)+IFSIZE(LITTLE)
      RETURN
      END



      SUBROUTINE EDGE(IX,IY,N,MST)
      DIMENSION MST(N)

      L=MST(IX)
      MST(IX)=IY
10    IF(L.EQ.-1) RETURN
      M=L
      L=MST(L)
      MST(M)=IX
      IX=M
      GOTO 10
      END
```