Documentation for the Implementation of Hierarchical Algorithm to Approximating the Shortest Paths in Southeastern Michigan Road Network

Yu-Li Chou
Department of Industrial & Operations Engineering
The University of Michigan
Ann Arbor, Michigan 48109-2117

Technical Report 95-32

December 1995

Documentation for the Implementation of Hierarchical Algorithm to Approximating the Shortest Paths in Southeastern Michigan Road Network *

Yu-Li Chou†

December 12, 1995

1 Introduction

This report documents the data files and several C programs which have been written to assist the implementation of Hierarchical Algorithm (*HA*) to solving shortest paths on the Southeastern Michigan (SEMCOG) road network.

Hierarchical Algorithm (see Chou [1], Chou, Romeijn and Smith [2]) is a heuristic approach for solving the shortest paths problems. It was applied to approximating the shortest paths in the random trip-requests simulation based upon the SEMCOG road network. Five versions (*HA1-HA5*) of the Hierarchical Algorithm were proposed. The main differences lie on the choices of the connecting macronodes for each micronode. This process is executed by the subroutine named **approximation** in the programs **haa1.c-haa5.c**. Since most parts of source codes are very similar among **haa1.c** - **haa5.c**, we only document **haa5.c** and **dijkstra.c** in details in this report. These source codes are attached at the end of this report.

2 Documentation for the Data Files (Network Files)

semcog.2 is the original SEMCOG road network. The first number contained in this file is the total number of entries , where each entry represents a street segment (a link) in the network. For further information about link information, please refer section 3. In **semcog.2**, links are classified into 6

*This work was supported in part by the Intelligent Transportation Systems Research Center of Excellence at the University of Michigan.

†Department of Industrial and Operations Engineering,
The University of Michigan, Ann Arbor, Michigan 48109-2117; e-mail: yuli.chou@umich.edu.

types. **semcog.5** is the modified SEMCOG road network where links are classified into two types. The link lengths are adjusted according to the speed limits of their corresponding link classes. In this experiment, the speed limits are set to be 65, 50, 40, 40, 35, 35 miles/hour for class 1 to class 6 links. Note that we use **semcog.5** to generate the input file for producing the two-class SEMCOG map through MAPINFO.

reduce5 is the data file which is modified from the original SEMCOG network; i.e. **semcog.2**. Links with a degree of 2 in the original network file are combined into a single link in **reduce5**. Furthermore, we add centroid zones into **reduce5**, where a node n is a centroid zone if $n \leq 1548$. By adding two dummy links from each centroid to two nearest nodes, we connect centroids to the original road network. The lengths for these dummy links are assigned to be the Euclidean distances between end nodes and the classes of these links are assigned to be class 7. (Note that links are classified into 7 classes in **reduce5**.)

TRIPTAB_85.TXT contains the average trips made between centroid zones for a 24-hour period. Please refer section 3.

centroidnode contains the information of all centroid zones within the Southeastern Michigan area. The first number is the total number of centroid zones, followed by each centroid node number, its x coordinate, and y coordinate.

macroleng.new is the macronetwork defined in the experiment. Each entry represents a macroarc.

microfile contains the node partition information defined in this experiment. For further information, please refer section 3.

3 Documentation for haa5.c

haa5.c is designed, following the procedures of *HAA5*, to compute the lengths of approximate shortest paths for *all pairs* of centroid zones in the SEMCOG network.

3.1 Input and Output Files

There is one input file read by haa5.c which contains the following information: 1: NWfile (Char type), 2: OUTfile (Char), 3: AGREG (Char), 4: MACRO (Char), 5: CENTROID, 6: TRIPDATA, 7: s1 (float), 8: s2 (float), 9: s3 (float), 10: s4 (float), 11: s5 (float), 12: s6 (float), 13: s7 (float). For example, the input file can look like

```
secg output aggnw macnw ctd triptable 65.0 50.0 40.0 40.0 35.0 35.0 35.0
```

The meanings of these items will be described next.

3.1.1 Descriptions of Items in the Input File

- **NWfile:** The SEMCOG road network data file. The first item in this data file is the total number of records included in the file, which is followed by these records. Each record represents a link in the network. For example, here shows partial SEMCOG data file (**reduce5**):

```
8743
14990  15020  1  408  2  54637  311167  54623  295231 ⟵ record 1
14990  1908   1  533  2  54637  311167  54589  332009 ⟵ record 2
...    ...
1451   15100  7  77   2  217957 82811   214487 84883  ⟵ record 8743
```

  where each record contains the following information:
  ANODE: starting node of the link,
  BNODE: end node of the link,
  CLASS: the class corresponding to the link [1 (high level) -7 (low level)],
  LENTH: the length of the link,
  LANES: can be 1 (one way) or 2 (two ways),
  XA : X coordinate of ANODE,
  XA : Y coordinate of ANODE,
  YB : X coordinate of BNODE,
  YB : Y coordinate of BNODE.

- **OUTfile:** The output file name. The output file will provide the information of: the total number of nodes in the network, the average travel time for one O/D pair.

- **AGREG:** This file contains information about nodes in different aggregate classes. For example,

```
23                                          ⟵ total no. subnetworks
16     2     2                              ← no.nodes, no.centroids, no.macros in nw 1
7069   6848                                 ← centroids in nw 1
6826   6638                                 ← macronodes in nw 1
15040  7047  15060  7049  15070  7091       ← micronodes in nw 1
15190  7590  7621   7624  7588   7589       ← micronodes in nw 1
13     3     2                              ← no.nodes, no.centroids, no.macros in nw 2
7159   7174  7283                           ← centroids in nw 2
7160   7272                                 ← macronodes in nw 2
7256   7253  7199   7161  7218   7231       ← micronodes in nw 2
7214   7228                                 ← micronodes in nw 2
...    ...                                  ...
```

- **MACRO:** Macro-network data file. Similar setup to **NWfile.**

```
173
1908  1503  1  1065  2  54589  332009  54673  290369 ← record1
1908  7290  1  402   2  54589  332009  51977  347527...
...   ...
1503  7069  1  2778  2  54673  290369  98189  208682 ← record 173
```

  where record i includes:
  AMACRO: starting node of macroarc i,
  BMACRO: end node of macroarc i,
  CLASS: the class corresponding to the macroarc (=1),
  LENTH: the length of the arc,
  LANES: can be 1 (one way) or 2 (two ways),
  XMA : X coordinate of AMACRO,
  YMA : Y coordinate of AMACRO,
  XMB : X coordinate of BMACRO,
  YMB : Y coordinate of BMACRO.

- **CENTROID:** Contain the information of all centroid zones. For example,

```
1543                  ← total no. of centroid zones
1     349060  304685  ← centroid 1, x-coordinate, y-coordinate
3     349024  305903  ...
4     348472  306098
...   ...
```

- **TRIPDATA:** Contain the information of trips made in a 24-hour period. For example

```
1   20  65  ← the no. of trips from centroid 1 to centroid 20 is 65
3   25  12  ← the no. of trips from centroid 3 to centroid 25 is 25
... ... ...
```

- **s1-s7:** Speed limits (mile/hr.) for class 1 links to class 7 links.

3.2 SUB-ROUTINES

There are several subroutines in the main program; they are listed and described as follows:

- **initialization():**
  Initializing the matrix **shortest**[i][j], **nooftrip**[i][j], $\forall i, j$, where **shortest**[i][j] is the length of shortest path from i to j and **nooftrip**[i][j] is the total number of trips made from i to j.

- **inputCentroid():**
  Read **CENTROID** file. **noofcentroid** is defined as the total number of centroid zones. **node**[i] is defined as the original node number in the SEMCOG file corresponding to node i. node[j] is a centroid zone, for $j = 0, 1 \ldots,$ **noofcentroid** $- 1$.

- **inputTripdata():**
  Read **TRIPDATA** file and fill in matrix **nooftrip**[i][j], for $i, j = 0, 1 \ldots,$ **noofcentroid** $- 1$.

- **inputNetData():**
  Call subroutine **macroNetwork** which reads **MACRO** file and builds the macronetwork structure, where **noOfmacros** is the total number of macronodes, **macronode**[A] is the original node number in the SEMCOG data corresponding to macronode A, **macrodegree**[A] is the degree of macronodes A in the macronetwork, **successor**[A][i] is the ith successor of A in the macronetwork, **length**[A][i] is the length corresponding to macroarc (A,successor[A][i]). Note that **node**[j] is a macronode, for $j =$ **noofcentroid**, ..., **noofcentroid**+**noOfmacros**$-$ 1.
  Call subroutine **Network()** which reads **NWfile** and builds the whole network structure, where **noOfNodes** is the total number of nodes in the network, **node**[A] is the original node number in the SEMCOG data corresponding to node A, **degree**[A] is the degree of node A, **successor**[A][i] is the ith successor of node A, **length**[A][i] is the length corresponding to arc (A,successor[A][i]).

- **aggregation():**
  Read **AGREG** file and build the data structure for each subnetwork. The definitions of several global variables appeared in this subroutine are described as follows:

  - **noofconnect**[i]: the number of macronodes which can connects centroid i to the macronetwork.
  - **frcent**[i][j]: the jth macronode which connects centroid i to the macronetwork.
  - **tocent**[i][j]:
  - **distfrcent**[i][j]: the length of local shortest path (i.e. the shortest path with all intermediate nodes contained in one subnetwork) from centroid i to macronode **frcent**[i][j].
  - **disttocent**[i][j]:

  Local variables are defined as:

  - **nomicronetwork**: the total number of nodes contained in the current subnetwork.
  - **micronode**[i]: the ith node contained in the current subnetwork.
  - **microdegree**[i]: the number of successors of i which are also contained in the current subnetwork.

- **microsucc**[i][j]: the jth successor of node i contained in the current sub-network.

Whenever the structure of a micro-subnetwork is built, a sub-routine **Dijkstra()** is called to computed the local shortest paths from all centroids and the local shortest paths from all macronodes in that subnetwork.

- **macroShort():**
  Dijkstra() is called to compute all shortest paths in the macro-network.
- **approximation():**
  For each pair of centroids, all possibilities of combination are computed; selected the best one as the approximated solution.
- **avgPathlength():**
  Compute the weighted (by the number of trips) average length of approximated shortest paths between all centroids.
- **printResults():**
  print out the results. The results are written in the file **OUTfile**, containing n (the total number of nodes), l (the average length of approximated shortest paths)

There are three important sub-routines called by the subroutines described above during the computation: **Dijkstra()**, **ran2()** and **CPUtime()**.

- **ran2():** providing a uniformly generated number which is between 0 and 1.
- **CPUtime():** indicating the amount of CPU times used by the process so far.
- **Dijkstra():** executing the Dijkstra algorithm for calculate the shortest paths. Note that there are several subroutines executed in **Dijkstra()** — **shiftUp()**, **shiftDown()**, **insertItem()**, and **deletMin()**. These subroutines are used for processing *heap* data structure.

3.2.1 Flowchart of haa5.c

The flow chart of **haa5.c** is shown in Figure 1.

3.3 Source Codes of haa5.c

See Appendix I.

4 Documentation for dijkstra.c

dijkstra.c is designed to compute the shortest paths for all pairs of centroid zones in the SEMCOG road network. The *heap* data structure is used in this program.

4.1 Input File and the Flowchart

There is one input file read by the main program which contains the following information: 1: NWfile (Char type), 2: OUTfile (Char), 3: CENTROID, 4: TRIPDATA, 5: s1 (float), 6: s2 (float), 7: s3 (float), 8: s4 (float), 9: s5 (float), 10: s6 (float), 11: s7 (float). please refer to section 3.

Figure 2 shows the flowchart of **dijkstra.c**.

5 Document for On-Line Trip Requests Simulation

5.1 trshaa5.c and trsdijkstra.c

trshaa5.c and **trsdijkstra.c** are designed to simulate the on-line trip-request versions of **haa5.c** and **dijkstra.c**. Please refer section 3 for input files read by these programs. Fig 3 shows the flowchart of **trshaa5.c**. The main difference between the on-line simulation and the previous deterministic case is that a random number will be first generated according to a uniform distribution. This number will be used to find a corresponding O/D pair through a probabilistic matrix (which is generated according to **TRIPTAB_85.TXT**). If the shortest path information of this O/D pair has not been computed at this moment, numerical calculations will be executed.

References

[1] Y.L. Chou, *Accelerating the Solutions of Dynamic Programs Through State Aggregation.* Ph.D. dissertation, The University of Michigan.

[2] Y.L. Chou, H.E. Romeijn, and R.L. Smith, Approximating Shortest Paths in Large-scale Networks with an Application to Intelligent Transportation Systems. to be summited to Operations Research on Computing.

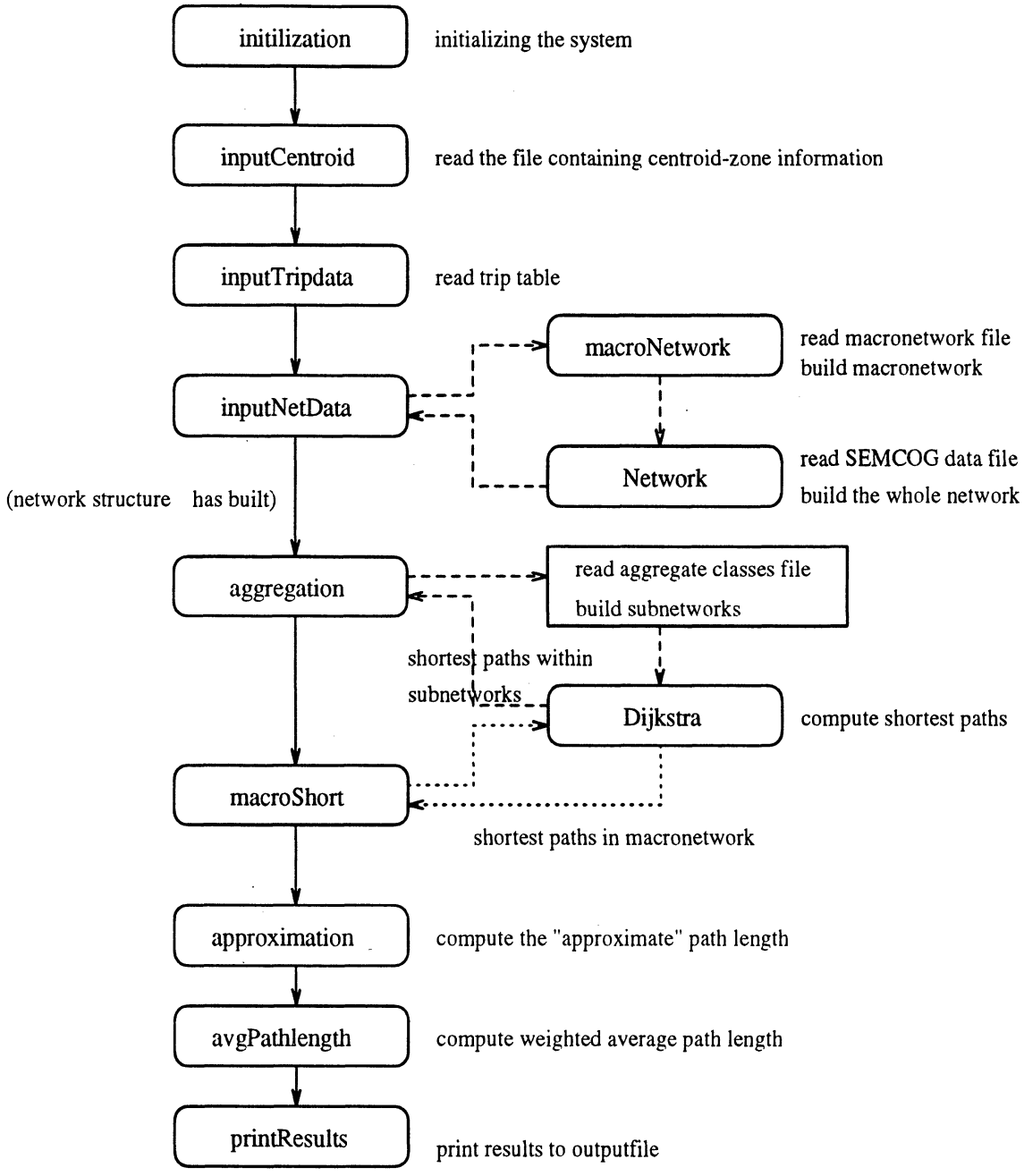

Figure 1: Flowchart of *haa5.c*

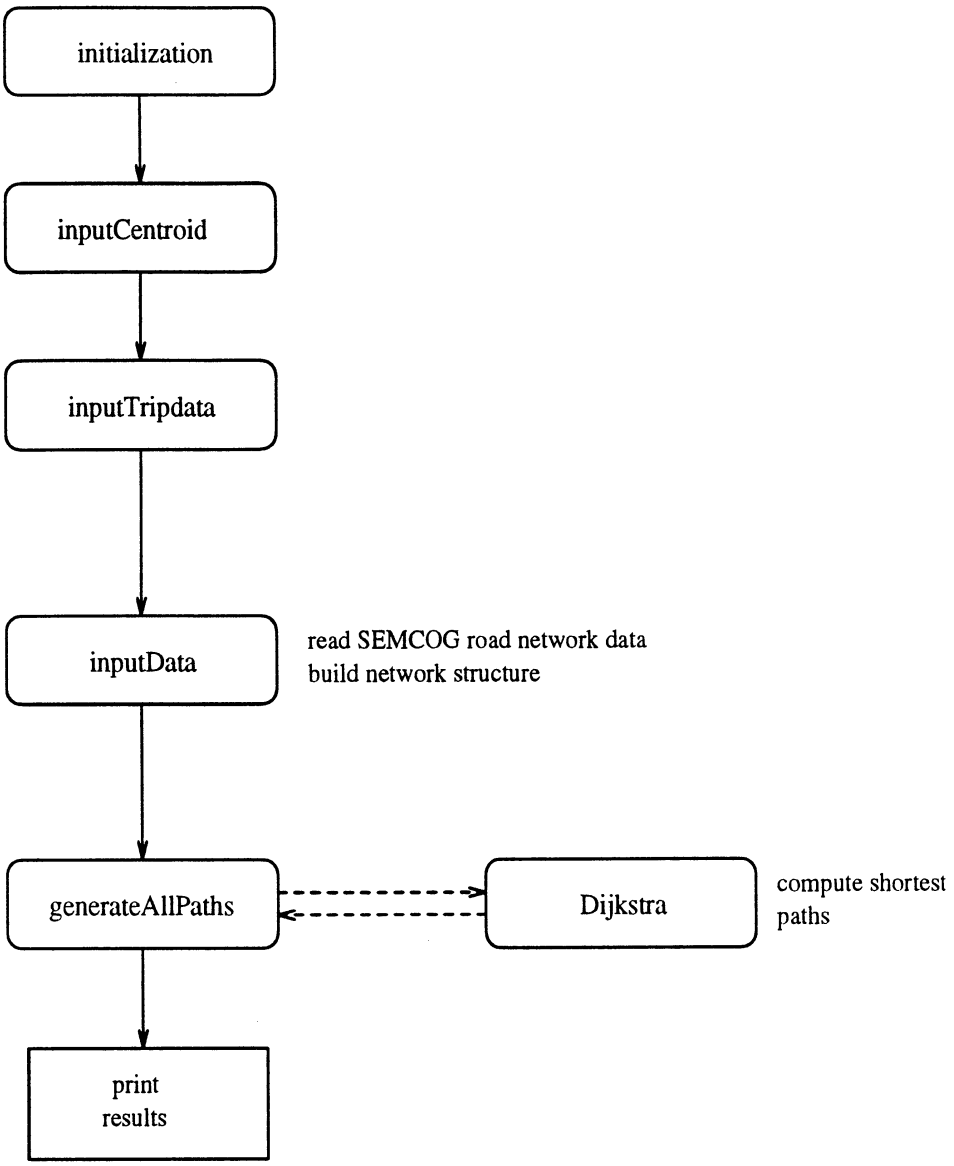

Figure 2: Flowchart of *dijkstra.c*

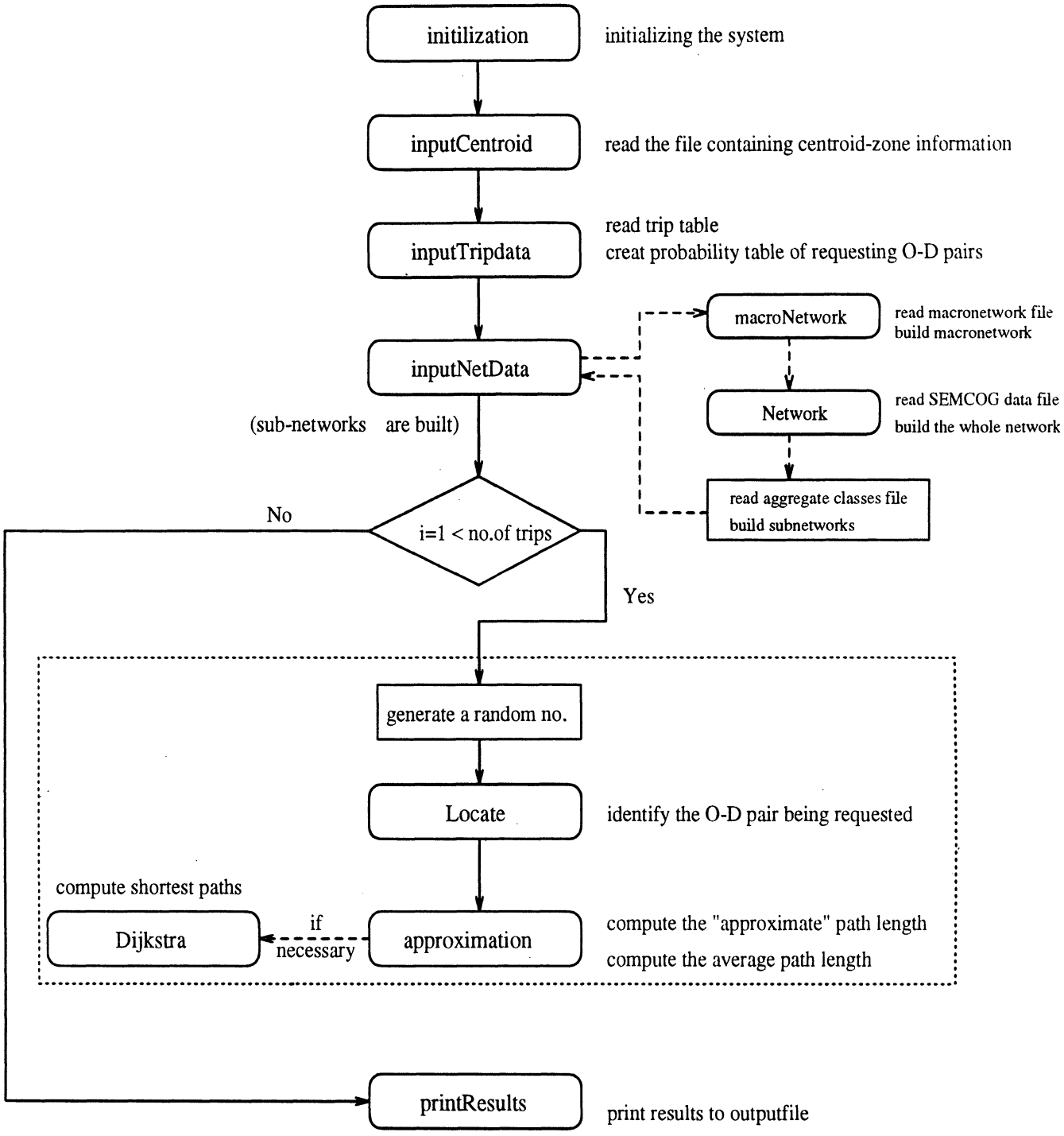

Figure 3: Flowchart of on-line trip request *haa5.c* version

```
/*************************************************************
This program "dijkstra.c" is designed to computed the average
shortest path length between all pairs of centroid zones in
the SMECOG road network by using the Dijkstra Algorithm
*************************************************************


#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/times.h>

#define MAX_NO_OF_NODES 3250
#define MAX_NO_OF_DEGREE 6
#define INFINITY 100000.0
#define STR_LENGTH 100
#define TICKS 100.0

void inputData (char str[STR_LENGTH],
                int *noOfNodes,
                int oldID[MAX_NO_OF_NODES],
                int degree[MAX_NO_OF_NODES],
                int successor[MAX_NO_OF_NODES][MAX_NO_OF_DEGREE],
                float length[MAX_NO_OF_NODES][MAX_NO_OF_DEGREE],
                float s1,
                float s2,
                float s3,
                float s4,
                float s5,
                float s6,
                float s7)

   {
   int i,j,k,dummy,noOfdata,totalnodes,newA,newB,Anode,Bnode,class,
       dummyA, dummyB,lane,xA,yA,xB,yB,tempID;
   float traveltime,speedlimit,temp;
   FILE *fp;


   fp=fopen(str,"r");
   if (fp==NULL)
     {
     printf("couldn't open %s.\n",str);
     exit(0);
     }

   fscanf(fp,"%d",&noOfdata);

   totalnodes=0;

   for (i=0;i<noOfdata;i++)
     {
     fscanf(fp,"%d %d %d %d %d %d %d %d %d",&Anode,&Bnode,&class,&dummy,
            &lane,&xA,&yA,&xB,&yB);
     temp=(float) dummy;

     if (class==1)
       speedlimit=s1;
     else if (class==2)
       speedlimit=s2;
     else if (class==3)
       speedlimit=s3;
     else if (class==4)
       speedlimit=s4;
     else if (class==5)
```

```
  speedlimit=s5;
else if (class==6)
  speedlimit=s6;
else if (class==7)
  speedlimit=s7;

traveltime=temp/speedlimit;

j=0;

while (Anode!=oldID[j] && j<totalnodes)
  j=j+1;

if (Anode!=oldID[j])
  {
  oldID[totalnodes]=Anode;
  newA=totalnodes;
  degree[newA]=1;
  totalnodes=totalnodes+1;
  }
else
  {
  newA=j;
  degree[newA]=degree[newA]+1;
  if (degree[newA]>MAX_NO_OF_DEGREE)
     {
     printf("degree of node %d is greater than MAX_NO_OF_DEGREE\n",
            oldID[newA]);
     exit(0);
     }
  }

j=0;

while (Bnode!=oldID[j] && j<totalnodes)
  j=j+1;

if (Bnode!=oldID[j])
  {
  oldID[totalnodes]=Bnode;
  newB=totalnodes;
  degree[newB]=1;
  totalnodes=totalnodes+1;
  }
else
  {
  newB=j;
  degree[newB]=degree[newB]+1;
  if (degree[newB]>MAX_NO_OF_DEGREE)
     {
     printf("degree of node %d is greater than MAX_NO_OF_DEGREE\n",
            oldID[newB]);
     exit(0);
     }
  }

dummyA=degree[newA]-1;
dummyB=degree[newB]-1;

successor[newA][dummyA]=newB;
length[newA][dummyA]=traveltime;

successor[newB][dummyB]=newA;
length[newB][dummyB]=traveltime;
}
```

```
    fclose(fp);
    *noOfNodes=totalnodes;

    printf("%d (%d)\n", oldID[1345],oldID[1346]);
    for (i=0;i<degree[1345];i++)
      printf("%d(%d)",successor[1345][i],oldID[successor[1345][i]]);
    printf("\n");
    for (i=0;i<degree[1346];i++)
      printf("%d(%d)",successor[1346][i],oldID[successor[1346][i]]);
    printf("\n");

    }

void shiftUp (int moveupitem,
              int heap[MAX_NO_OF_NODES],
              int index[MAX_NO_OF_NODES],
              float dshortest[MAX_NO_OF_NODES])
    {
    int parent,moveupnode,i,j;
    float temp;

    moveupnode=heap[moveupitem];

    parent=moveupitem/2;

    while (dshortest[heap[parent]]>dshortest[moveupnode] &&
           moveupitem>1)
      {
      heap[moveupitem]=heap[parent];
      index[heap[moveupitem]]=moveupitem;
      moveupitem=parent;
      parent=moveupitem/2;
      }

    index[moveupnode]=moveupitem;
    heap[moveupitem]=moveupnode;
    }

void shiftDown (int heapsize,
                int movedownitem,
                int index[MAX_NO_OF_NODES],
                int heap[MAX_NO_OF_NODES],
                float dshortest[MAX_NO_OF_NODES])
    {
    int child,movedownnode,i,j,dummy;

    movedownnode=heap[movedownitem];
    child=2*movedownitem;

    while (child<=heapsize)
      {
      if ((child+1)<=heapsize &&
          dshortest[heap[child+1]]<dshortest[heap[child]])
        child=child+1;

      if (dshortest[heap[child]]>=dshortest[movedownnode])
        return;
      else
        {
        heap[movedownitem]=heap[child];
        index[heap[movedownitem]]=movedownitem;
        movedownitem=child;
        index[movedownnode]=movedownitem;
        heap[movedownitem]=movedownnode;
        child=2*movedownitem;
        }
```

```
    }
  }

void insertItem (int *sizeofHeap,
                 int heap[MAX_NO_OF_NODES],
                 int index[MAX_NO_OF_NODES],
                 int insertnode,
                 float dshortest[MAX_NO_OF_NODES])
  {
  int i,j,moveupitem;

  *sizeofHeap=*sizeofHeap+1;
  heap[*sizeofHeap]=insertnode;
  index[insertnode]=*sizeofHeap;

  if (*sizeofHeap==1)
    return;

  shiftUp(*sizeofHeap,heap,index,dshortest);
  }

void deleteMin (int *sizeofHeap,
                int heap[MAX_NO_OF_NODES],
                int index[MAX_NO_OF_NODES],
                float dshortest[MAX_NO_OF_NODES])
  {
  int i,j;

  heap[1]=heap[*sizeofHeap];
  *sizeofHeap=*sizeofHeap-1;
  i=1;
  index[heap[1]]=1;
  shiftDown(*sizeofHeap,i,index,heap,dshortest);
  }


void Dijstra (int noOfNodes,
              int oldID[MAX_NO_OF_NODES],
              int degree[MAX_NO_OF_NODES],
              float dlength[MAX_NO_OF_NODES][MAX_NO_OF_DEGREE],
              int node[MAX_NO_OF_NODES][MAX_NO_OF_DEGREE],
              int root,
              float dshortest[MAX_NO_OF_NODES])
   {
   int oldvalue,i, currentnode, T,k,sizeofHeap,
        check,index[MAX_NO_OF_NODES],
        label[MAX_NO_OF_NODES],
        dummynode,heap[MAX_NO_OF_NODES];
   float  dummy;

   dshortest[root] = 0.0;
   label[root]=1;
/*   predeccessor[root]=root; */

   for (i=0; i<noOfNodes; i++)
       if (i != root)
         {
          label[i] = 0;
          dshortest[i]=INFINITY;
         }

   sizeofHeap=0;

   for (k=0;k<degree[root];k++)
      {
      dummynode=node[root][k];
```

```
      dshortest[dummynode] = dlength[root][k];

      insertItem(&sizeofHeap,heap,index,dummynode,dshortest);

/*      predeccessor[dummynode]=root; */
      }

   T = noOfNodes-1;

   while (T>0)
      {
        currentnode = heap[1];
        label[currentnode]=1;

        for (k=0;k<degree[currentnode];k++)
           {
           i=node[currentnode][k];
           if (label[i]==0)
              {
              oldvalue=dshortest[i];

              dummy = dshortest[currentnode] + dlength[currentnode][k];
              if (dummy < dshortest[i])
                 {
                 dshortest[i] = dummy;

                 if (oldvalue>=INFINITY)
                   insertItem(&sizeofHeap,heap,index,i,dshortest);
                 else
                   shiftUp(index[i],heap,index,dshortest);
                 }
              }
           }
        deleteMin(&sizeofHeap,heap,index,dshortest);

        T=T-1;
      }

   }

void generateAllPaths (int noOfNodes,
                       int oldID[MAX_NO_OF_NODES],
                       float length[MAX_NO_OF_NODES][MAX_NO_OF_DEGREE],
                       int successor[MAX_NO_OF_NODES][MAX_NO_OF_DEGREE],
                       int degree[MAX_NO_OF_NODES],
                       float shortest[MAX_NO_OF_NODES][MAX_NO_OF_NODES])
   {
   int count,root,i, j;
   float temp_total;

   temp_total=0.0;
   for (root=0; root<noOfNodes; root++)
     {
     Dijstra(noOfNodes,oldID, degree,length,successor,root,
             shortest[root]);
/*     if (root >1345)
        printf("%d\n",root);*/
     }

  }


void printResults (char outputname[STR_LENGTH],
                   int noOfNodes,
                   int oldID[MAX_NO_OF_NODES],
                   float shortest[MAX_NO_OF_NODES][MAX_NO_OF_NODES],
```

```
                float time1,
                float time2)
{
int tempi,tempj,tempnode,i,j,check,count,totalpairs, dummy;
float total_length,temp,avglength;
FILE *fp;

totalpairs=noOfNodes*noOfNodes;

dummy = (float) totalpairs;
total_length=0.0;

fp=fopen(outputname,"w");
if (fp==NULL)
  {
  printf("couldn't open %s\n",outputname);
  exit(0);
  }
check=0;

fprintf(fp,"%d\n",noOfNodes);

/*fprintf(fp,"shortest distances between pairs of nodes :\n\n\t");*/
count=0;

for (i=0;i<noOfNodes;i++)
 {
 for (j=0;j<noOfNodes;j++)
      total_length=total_length+shortest[i][j];
 }

/* fprintf(fp,"nodes on the shortest paths between pairs of nodes :\n\n\t");*/
/* count=0;*/
/* for (i=0;i<noOfNodes;i++)*/
 /* {*/
 /* count=count+1;*/
 /* fprintf(fp,"%5d",oldID[i]);*/
 /* if (count%13==0)*/
   /* fprintf(fp,"\n");*/
 /* }*/
/* fprintf(fp,"\n\n");*/

 /*for (i=0;i<noOfNodes;i++)*/
 /* {*/
/*  fprintf(fp,"%d\t",oldID[i]);*/
 /* count=0;*/
 /* for (j=0;j<noOfNodes;j++)*/
  /*  {*/
   /* count=count+1;*/
   /* fprintf(fp,"%5d",oldID[predeccessor[i][j]]);*/
   /* if (count%13==0)*/
/*     fprintf(fp,"\n");*/
/*    }*/
  /*fprintf(fp,"\n");*/
 /* }*/

 if (check==1)
   printf("warning !! network is not connected\n");

 avglength=total_length/dummy;
 fprintf(fp,"\n average travel time between pairs of nodes =%f (hr.)\n",
         avglength);
 fprintf(fp,"time1=%f\n",time1);
 fprintf(fp,"time2=%f\n",time2);

 fclose(fp);
```

```
    }

float CPUtime (void)
    {
    float cpuTime;
    struct tms localTimeStruct;

    times (&localTimeStruct);

    cpuTime = (localTimeStruct.tms_utime + localTimeStruct.tms_stime) / TICKS;
    return cpuTime;
    }

void main (void)
    {
    int noOfNodes,i,successor[MAX_NO_OF_NODES][MAX_NO_OF_DEGREE],
        degree[MAX_NO_OF_NODES],oldID[MAX_NO_OF_NODES];
    float r, length[MAX_NO_OF_NODES][MAX_NO_OF_DEGREE],
         temp1,temp2,temp3,time1,time2,time3,
         s1,s2,s3,s4,s5,s6,s7,
         shortest[MAX_NO_OF_NODES][MAX_NO_OF_NODES],total_length;

    char inputname[STR_LENGTH], outputname[STR_LENGTH],str3[STR_LENGTH];
    FILE *fp1, *fp;

    fp1 = fopen("inputdijkstra", "r");
    if (fp1==NULL)
       {
       printf (" -> Could not open file <inputexact>\n");
       exit(0);
       }
    fscanf(fp1,"%s %s%f%f%f%f%f%f%f",inputname,outputname,
           &s1,&s2,&s3,&s4,&s5,&s6,&s7);
    fclose(fp1);

    inputData(inputname,&noOfNodes,oldID,degree,successor,length,s1,s2,s3,s4,s5,s6,s7);

    temp1=CPUtime();

    generateAllPaths(noOfNodes,oldID,length,successor,
                     degree,shortest);

    temp2=CPUtime();

    time1=temp1;
    time2=temp2-temp1;

    printResults(outputname,noOfNodes,oldID,shortest,
                 time1,time2);
    }
```

```
/************************************************************
This program is designed to compute the average path lengths
between all pairs of centroid zones in the SEMCOG road network
using HAA5 (Best HA)
************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/times.h>

#define MAX_NO_OF_NODES 3200
#define MAX_NO_OF_MACROS 150
#define MAX_NO_OF_inMACROS 280
#define MAX_NO_OF_DEGREE 6
#define INFINITY 500000.0
#define STR_LENGTH 100
#define MAX_NO_OF_NW 30
#define MAX_NO_OF_CONNECT 30
#define TICKS 100.0

void intialization (float shortest[MAX_NO_OF_NODES][MAX_NO_OF_NODES])
   {
   int i,j;

   for (i=0;i<MAX_NO_OF_NODES;i++)
     for (j=0;j<MAX_NO_OF_NODES;j++)
       shortest[i][j]=0.0;

   }

void macroNetwork (char macronw[STR_LENGTH],
                   int *totalmacros,
                   int node[MAX_NO_OF_NODES],
                   int degree[MAX_NO_OF_NODES],
                   int macronode[MAX_NO_OF_MACROS],
                   float macrolength[MAX_NO_OF_MACROS][MAX_NO_OF_DEGREE],
                   int macrodegree[MAX_NO_OF_MACROS],
                   int macrosuccessor[MAX_NO_OF_MACROS][MAX_NO_OF_DEGREE],
                   float s1)
   {
   int Anode,Bnode,class,i,j,k,noofmacros,dummy,newA,newB,dummyA,
       lane,xA,xB,yA,yB,dummyB,tempID,noofdata;
   float mlength,traveltime;
   FILE *fp;

   fp=fopen(macronw,"r");
   if (fp==NULL)
     {
     printf("could not open %s\n",macronw);
     exit(0);
     }

   fscanf(fp,"%d",&noofdata);
   *totalmacros=0;

   for (i=0;i<noofdata;i++)
     {
     fscanf(fp,"%d %d %d %d %d %d %d %d %d",&Anode,&Bnode,&class,&dummy,
            &lane,&xA,&yA,&xB,&yB);

     mlength=(float) dummy;
     traveltime=mlength/s1;
```

```
      j=0;
      while (Anode!=node[j] && j<*totalmacros)
        j=j+1;

      if (Anode!=node[j])
        {
        newA=*totalmacros;
        macronode[newA]=newA;
        macrodegree[newA]=1;
        node[newA]=Anode;
        degree[newA]=0;
        *totalmacros=*totalmacros+1;
        }
      else
        {
        newA=j;
        macrodegree[newA]=macrodegree[newA]+1;
        }

      j=0;
      while (Bnode!=node[j] && j<*totalmacros)
        j=j+1;

      if (Bnode!=node[j])
        {
        newB=*totalmacros;
        macronode[newB]=newB;
        macrodegree[newB]=1;
        node[newB]=Bnode;
        degree[newB]=0;
        *totalmacros=*totalmacros+1;
        }
      else
        {
        newB=j;
        macrodegree[newB]=macrodegree[newB]+1;
        }


      dummyA=macrodegree[newA]-1;
      dummyB=macrodegree[newB]-1;

      macrosuccessor[newA][dummyA]=newB;
      macrolength[newA][dummyA]=traveltime;

      macrosuccessor[newB][dummyB]=newA;
      macrolength[newB][dummyB]=traveltime;
      }

    fclose(fp);
    }

void Network (int Anode,
              int Bnode,
              float leng,
              int *totalnodes,
              int node[MAX_NO_OF_NODES],
              float length[MAX_NO_OF_NODES][MAX_NO_OF_DEGREE],
              int degree[MAX_NO_OF_NODES],
              int successor[MAX_NO_OF_NODES][MAX_NO_OF_DEGREE])
    {
    int i,j,k,newA,newB,dummy,dummyA, dummyB,tempID;

    j=0;
    while (Anode!=node[j] && j<*totalnodes)
```

```
    j=j+1;

  if (Anode!=node[j])
    {
    *totalnodes=*totalnodes+1;
    newA=*totalnodes-1;
    node[newA]=Anode;
    degree[newA]=1;
    }
  else
    {
    newA=j;
    degree[newA]=degree[newA]+1;
    if (degree[newA]>MAX_NO_OF_DEGREE)
      {
      printf("degree of node %d is greater than MAX_NO_OF_DEGREE\n",
             Anode);
      exit(0);
      }
    }

  j=0;
  while (Bnode!=node[j] && j<*totalnodes)
    j=j+1;

  if (Bnode!=node[j])
    {
    *totalnodes=*totalnodes+1;
    newB=*totalnodes-1;
    node[newB]=Bnode;
    degree[newB]=1;
    }
  else
    {
    newB=j;
    degree[newB]=degree[newB]+1;
    if (degree[newB]>MAX_NO_OF_DEGREE)
      {
      printf("degree of node %d is greater than MAX_NO_OF_DEGREE\n",
             Bnode);
      exit(0);
      }
    }


  dummyA=degree[newA]-1;
  dummyB=degree[newB]-1;

  successor[newA][dummyA]=newB;
  length[newA][dummyA]=leng;

  successor[newB][dummyB]=newA;
  length[newB][dummyB]=leng;
  }


void inputNetData (char str[STR_LENGTH],
                   char macronw[STR_LENGTH],
                   int *noOfmacros,
                   int *noOfNodes,
                   int node[MAX_NO_OF_NODES],
                   int degree[MAX_NO_OF_NODES],
                   int successor[MAX_NO_OF_NODES][MAX_NO_OF_DEGREE],
                   float length[MAX_NO_OF_NODES][MAX_NO_OF_DEGREE],
                   int macronode[MAX_NO_OF_MACROS],
                   int macrodegree[MAX_NO_OF_MACROS],
```

```
                int macrosuccessor[MAX_NO_OF_MACROS][MAX_NO_OF_DEGREE],
                float macrolength[MAX_NO_OF_MACROS][MAX_NO_OF_DEGREE],
                float s1,
                float s2,
                float s3,
                float s4,
                float s5,
                float s6,
                float s7)

    {
    int i,dummy,noOfdata,totalnodes,Anode,Bnode,class,totalmacros,newA,newB,
        lane,xA,yA,xB,yB,checkA,checkB;
    float traveltime,speedlimit,distance;
    FILE *fp,*fp1;

    fp=fopen(str,"r");
    if (fp==NULL)
      {
      printf("couldn't open %s.\n",str);
      exit(0);
      }

    macroNetwork(macronw,&totalmacros,node,degree,macronode,
             macrolength,macrodegree,macrosuccessor,s1);

    *noOfmacros=totalmacros;

    fscanf(fp,"%d",&noOfdata);

    totalnodes=totalmacros;

    for (i=0;i<noOfdata;i++)
      {
      fscanf(fp,"%d %d %d %d %d %d %d %d %d",&Anode,&Bnode,&class,&dummy,
             &lane,&xA,&yA,&xB,&yB);
      distance=(float) dummy;

      if (class==1)
        speedlimit=s1;
      else if (class==2)
        speedlimit=s2;
      else if (class==3)
        speedlimit=s3;
      else if (class==4)
        speedlimit=s4;
      else if (class==5)
        speedlimit=s5;
      else if (class==6)
        speedlimit=s6;
      else if (class==7)
        speedlimit=s7;

      traveltime=distance/speedlimit;


      Network(Anode,Bnode,traveltime,&totalnodes,node,
             length,degree,successor);

      }
    fclose(fp);

    *noOfNodes=totalnodes;
    }

/*----- makeheap----------------------------*/
```

```
void shiftUp (int moveupitem,
              int heap[MAX_NO_OF_NODES],
              int index[MAX_NO_OF_NODES],
              float dshortest[MAX_NO_OF_NODES],
              int realnode[MAX_NO_OF_inMACROS])
   {
   int parent,moveupnode,i,j;
   float temp;

   moveupnode=heap[moveupitem];

   parent=moveupitem/2;

   while (dshortest[realnode[heap[parent]]]>dshortest[realnode[moveupnode]] &&
          moveupitem>1)
     {
     heap[moveupitem]=heap[parent];
     index[heap[moveupitem]]=moveupitem;
     moveupitem=parent;
     parent=moveupitem/2;
     }

   index[moveupnode]=moveupitem;
   heap[moveupitem]=moveupnode;
   }

void shiftDown (int heapsize,
                int movedownitem,
                int index[MAX_NO_OF_NODES],
                int heap[MAX_NO_OF_NODES],
                float dshortest[MAX_NO_OF_NODES],
                int realnode[MAX_NO_OF_inMACROS])
   {
   int child,movedownnode,i,j,dummy;

   movedownnode=heap[movedownitem];
   child=2*movedownitem;

   while (child<=heapsize)
     {
     if ((child+1)<=heapsize &&
         dshortest[realnode[heap[child+1]]]<dshortest[realnode[heap[child]]])
       child=child+1;

     if (dshortest[realnode[heap[child]]]>=dshortest[realnode[movedownnode]])
       return;
     else
       {
       heap[movedownitem]=heap[child];
       index[heap[movedownitem]]=movedownitem;
       movedownitem=child;
       index[movedownnode]=movedownitem;
       heap[movedownitem]=movedownnode;
       child=2*movedownitem;
       }
     }
  }

void insertItem (int *sizeofHeap,
                 int heap[MAX_NO_OF_NODES],
                 int index[MAX_NO_OF_NODES],
                 int insertnode,
                 float dshortest[MAX_NO_OF_NODES],
                 int realnode[MAX_NO_OF_inMACROS])
  {
```

```
  int i,j,moveupitem;

  *sizeofHeap=*sizeofHeap+1;
  heap[*sizeofHeap]=insertnode;
  index[insertnode]=*sizeofHeap;

  if (*sizeofHeap==1)
    return;

  shiftUp(*sizeofHeap,heap,index,dshortest,realnode);
  }

void deleteMin (int *sizeofHeap,
                int heap[MAX_NO_OF_NODES],
                int index[MAX_NO_OF_NODES],
                float dshortest[MAX_NO_OF_NODES],
                int realnode[MAX_NO_OF_inMACROS])
  {
  int i,j;

  heap[1]=heap[*sizeofHeap];
  *sizeofHeap=*sizeofHeap-1;
  i=1;
  index[heap[1]]=1;
  shiftDown(*sizeofHeap,i,index,heap,dshortest,realnode);
  }

/*-----------makeheap----------------------------*/

void Dijstra (int nw,
              int noofnodes,
              int node[MAX_NO_OF_NODES],
              int realnode[MAX_NO_OF_inMACROS],
              float dlength[MAX_NO_OF_inMACROS][MAX_NO_OF_DEGREE],
              int dsuccessor[MAX_NO_OF_inMACROS][MAX_NO_OF_DEGREE],
              int ddegree[MAX_NO_OF_inMACROS],
              int root,
              float dshortest[MAX_NO_OF_NODES])
   {
   int oldvalue,i,count, currentnode, T,k,
      sizeofHeap,index[MAX_NO_OF_NODES],heap[MAX_NO_OF_NODES],
      check, label[MAX_NO_OF_inMACROS],dummynode;
   float  dummy;
   FILE *fp;

   label[root] = 1;
   dshortest[realnode[root]] = 0.0;

   for (i=0; i<noofnodes; i++)
       if (i != root)
         {
          label[i] = 0;
          dshortest[realnode[i]]=INFINITY;
         }
   sizeofHeap=0;

   for (k=0;k<ddegree[root];k++)
      {
      dummynode=dsuccessor[root][k];
      if (dummynode>=0)
        {
        dshortest[realnode[dummynode]] = dlength[root][k];
        insertItem(&sizeofHeap,heap,index,dummynode,dshortest,realnode);
        }
      }
```

```
   T = noofnodes-1;

   while (T>0)
      {
        if (sizeofHeap<=0  && nw>=0)
          {
          printf("nodes %d in micronetwork %d is not completely connected !! \n",
                 node[realnode[root]],nw);
          exit (0);
          }
         else if (sizeofHeap<=0 && nw<0)
          {
          printf("nodes %d in macronetwork is not completely connected !! \n",
                 node[realnode[root]]);
          exit (0);
          }

        currentnode=heap[1];
        label[currentnode]=1;

        for (k=0;k<ddegree[currentnode];k++)
           {
           i=dsuccessor[currentnode][k];
           if (label[i]==0 && i>=0)
              {
              oldvalue=dshortest[realnode[i]];

              dummy = dshortest[realnode[currentnode]] +
                      dlength[currentnode][k];
              if (dummy < dshortest[realnode[i]])
                 {
                 dshortest[realnode[i]] = dummy;
                 if (oldvalue>=INFINITY)
                   insertItem(&sizeofHeap,heap,index,i,dshortest,realnode);
                 else
                   shiftUp(index[i],heap,index,dshortest,realnode);
                 }
              }
           }

        deleteMin(&sizeofHeap,heap,index,dshortest,realnode);

        T=T-1;
      }

   }

void aggregation(char str1[STR_LENGTH],
                 int node[MAX_NO_OF_NODES],
                 int degree[MAX_NO_OF_NODES],
                 int successor[MAX_NO_OF_NODES][MAX_NO_OF_DEGREE],
                 float length[MAX_NO_OF_NODES][MAX_NO_OF_DEGREE],
                 int noOfmacros,
                 float shortest[MAX_NO_OF_NODES][MAX_NO_OF_NODES],
                 int micronetwork[MAX_NO_OF_NODES],
                 int noofconnect[MAX_NO_OF_NW],
                 int connect[MAX_NO_OF_NW][MAX_NO_OF_CONNECT])
   {
   int dummy,i,j,k,temp_node,nomicronetwork,temp,root,rootID,nodeID,
       microdegree[MAX_NO_OF_inMACROS],count,
       micronode[MAX_NO_OF_inMACROS],totalnode,totalpair,
       microsucc[MAX_NO_OF_inMACROS][MAX_NO_OF_DEGREE];
   float temp_dist,microlength[MAX_NO_OF_inMACROS][MAX_NO_OF_DEGREE];
   FILE *fp,*fp1;

   fp=fopen(str1,"r");
```

```
if (fp==NULL)
  {
  printf("couldn't open %s.\n",str1);
  exit(0);
  }


fscanf(fp,"%d",&nomicronetwork);

for (i=0;i<nomicronetwork;i++)
  {
  fscanf(fp,"%d %d",&totalnode,&noofconnect[i]);

  printf("%d\n",i);

  for (j=0;j<noofconnect[i];j++)
    {
    fscanf(fp,"%d",&temp_node);

    k=0;
    while (temp_node!=node[k])
       k=k+1;

    connect[i][j]=k;
    micronode[j]=k;
    }

  for (j=noofconnect[i];j<totalnode;j++)
    {
    fscanf(fp,"%d",&temp_node);

    k=0;
    while (temp_node!=node[k])
       k=k+1;

    micronetwork[k]=i;
    micronode[j]=k;
    }

  for (j=0;j<totalnode;j++)
    {
    temp_node=micronode[j];
    microdegree[j]=degree[temp_node];

    for(count=0;count<degree[temp_node];count++)
        {
        dummy=successor[temp_node][count];

        k=0;
        while (dummy!=micronode[k] && k<totalnode)
          k=k+1;

        if (k>totalnode-1)
          {
          microlength[j][count]=INFINITY;
          microsucc[j][count]=-1;
          }
        else
          {
          microlength[j][count]=length[temp_node][count];
          microsucc[j][count]=k;
          }
        }
    }

  for (root=0;root<totalnode;root++)
```

```
        Dijstra(i,totalnode,node,micronode, microlength,microsucc,microdegree,root,
              shortest[micronode[root]]);


     }

     fclose(fp);

    }


void macroShort (int noOfmacros,
                 int node[MAX_NO_OF_NODES],
                 int macronode[MAX_NO_OF_MACROS],
                 float macrolength[MAX_NO_OF_MACROS][MAX_NO_OF_DEGREE],
                 int macrodegree[MAX_NO_OF_MACROS],
                 int macrosuccessor[MAX_NO_OF_MACROS][MAX_NO_OF_DEGREE],
                 float shortest[MAX_NO_OF_NODES][MAX_NO_OF_NODES])
   {
   int mroot,i,j,totalpair,dummy,p;
   float temp_short[MAX_NO_OF_MACROS];

   dummy=-1;

   for(mroot=0;mroot<noOfmacros;mroot++)
      {
      Dijstra(dummy,noOfmacros,node,macronode,macrolength,macrosuccessor,macrodegree,
           mroot,temp_short);

      for (i=0;i<noOfmacros;i++)
        if (temp_short[macronode[i]]<shortest[macronode[mroot]][macronode[i]]
           || shortest[macronode[mroot]][macronode[i]]==0.0)
          shortest[macronode[mroot]][macronode[i]]=temp_short[macronode[i]];
       }
     }


void approximation (int noOfNodes,
                    int noOfmacros,
                    float shortest[MAX_NO_OF_NODES][MAX_NO_OF_NODES],
                    int micronetwork[MAX_NO_OF_NODES],
                    int noofconnect[MAX_NO_OF_NW],
                    int connect[MAX_NO_OF_NW][MAX_NO_OF_CONNECT])
   {
   int  i,tonodei,tonodej,dummy,dummy1,dummy2,sumdist,j,
        l,m,count,k,noofnextnw;
   float temp_dist,temp,euclidean;


   for (i=0;i<noOfNodes;i++)
     for (j=0;j<noOfNodes;j++)
      if (shortest[i][j]==0.0 && i!=j)
        {
        if (i<noOfmacros && j>=noOfmacros)
          dummy=micronetwork[j];
        else if (i>=noOfmacros && j<noOfmacros)
          dummy=micronetwork[i];
        else if (i>=noOfmacros && j>=noOfmacros)
          {
          dummy1=micronetwork[i];
          dummy2=micronetwork[j];
          }

        if ((i<noOfmacros && j>=noOfmacros) ||
            (i>=noOfmacros && j<noOfmacros))
          {
```

```
          temp_dist=INFINITY;

          for(k=0;k<noofconnect[dummy];k++)
           {
           temp=shortest[i][connect[dummy][k]]+
               shortest[connect[dummy][k]][j];

           if (temp<temp_dist)
             temp_dist=temp;
           }

          shortest[i][j]=temp_dist;
          }

        else if (i>=noOfmacros && j>=noOfmacros)
          {
          temp_dist=INFINITY;

          for(k=0;k<noofconnect[dummy1];k++)
            for(l=0;l<noofconnect[dummy2];l++)
             {
             temp=shortest[i][connect[dummy1][k]]+
                  shortest[connect[dummy1][k]][connect[dummy2][l]]+
                  shortest[connect[dummy2][l]][j];
             if (temp<temp_dist)
               temp_dist=temp;
             }

         shortest[i][j]=temp_dist;
         }
       }
   }

void printResults (char str[STR_LENGTH],
                   int noOfNodes,
                   int node[MAX_NO_OF_NODES],
                   float shortest[MAX_NO_OF_NODES][MAX_NO_OF_NODES],
                   float time1,
                   float time2,
                   float time3,
                   float time4)
   {
   int count1,i,j,totalpairs,count;
   float avgleng,totaleng,dummy;
   FILE *fp,*fp1;

   totaleng=0.0;

   fp=fopen(str,"w");
   if (fp==NULL)
     {
     printf("couldn't open %s.\n",str);
     exit(0);
     }


   totalpairs=noOfNodes*noOfNodes;
   dummy=(float) totalpairs;


   fprintf(fp,"%d\n ",noOfNodes);


   for (i=0;i<noOfNodes;i++)
    {
    for (j=0;j<noOfNodes;j++)
```

```
      totaleng=totaleng+shortest[i][j];
/*    fprintf(fp,"\n");*/
    }



 /*  fprintf(fp,"predeccessors of nodes on the paths :\n\n\t");*/
  /* count=0;*/
  /* for (i=0;i<noOfNodes;i++)*/
   /* {*/
   /* count=count+1;*/
   /* fprintf(fp,"%5d",node[i]);*/
   /* if (count%13==0)*/
    /*  fprintf(fp,"\n");*/
    /*}*/
  /* fprintf(fp,"\n\n");*/

  /* for (i=0;i<noOfNodes;i++)*/
   /* { */
   /* fprintf(fp,"%d\t",node[i]);*/
   /* count=0;*/
   /* for (j=0;j<noOfNodes;j++)*/
    /*  {*/
     /* count=count+1;*/
     /* fprintf(fp,"%5d",node[predeccessor[i][j]]);*/
     /* if (count%13==0)*/
/*      fprintf(fp,"\n");*/
  /*    }*/
   /* fprintf(fp,"\n");*/
    /*}*/

/*   fprintf(fp,"\n\n");*/

   avgleng=totaleng/dummy;
   fprintf(fp,"average length between each pair of nodes = %f\n",avgleng);
   fprintf(fp,"time1=%f\n",time1);
   fprintf(fp,"time2=%f\n",time2);
   fprintf(fp,"time3=%f\n",time3);
   fprintf(fp,"time4=%f\n",time4);

   fclose(fp);
   }

float CPUtime (void)
   {
   float cpuTime;
   struct tms localTimeStruct;

   times (&localTimeStruct);

   cpuTime = (localTimeStruct.tms_utime + localTimeStruct.tms_stime) / TICKS;
   return cpuTime;
   }

void main (void)
   {
   int node[MAX_NO_OF_NODES],macronode[MAX_NO_OF_MACROS],
       noOfNodes,degree[MAX_NO_OF_NODES],macrodegree[MAX_NO_OF_MACROS],
       noOfmacros,successor[MAX_NO_OF_NODES][MAX_NO_OF_DEGREE],
       macrosuccessor[MAX_NO_OF_MACROS][MAX_NO_OF_DEGREE],
       i,j,micronetwork[MAX_NO_OF_NODES],
       noofconnect[MAX_NO_OF_NW],
       connect[MAX_NO_OF_NW][MAX_NO_OF_CONNECT];
   float length[MAX_NO_OF_NODES][MAX_NO_OF_DEGREE],
         macrolength[MAX_NO_OF_MACROS][MAX_NO_OF_DEGREE],
         s1,s2,s3,s4,s5,s6,s7;
```

```
float temp1,temp2,temp3,temp4,shortest[MAX_NO_OF_NODES][MAX_NO_OF_NODES],
      time1,time2,time3,time4;

char inputname[STR_LENGTH], outputname[STR_LENGTH],
     micronw[STR_LENGTH],macronw[STR_LENGTH];
FILE *fp1;

fp1 = fopen("inputenum", "r");
if (fp1==NULL)
   {
   printf("could not open <<inputenum>>\n");
   exit(0);
   }

fscanf(fp1,"%s %s %s %s%f%f%f%f%f%f%f%",inputname,outputname,micronw,macronw,
      &s1,&s2,&s3,&s4,&s5,&s6,&s7);
fclose(fp1);

intialization (shortest);

temp1=CPUtime();

inputNetData (inputname,macronw,&noOfmacros,&noOfNodes,node,
              degree,successor,length,macronode,
              macrodegree,macrosuccessor,macrolength,s1,s2,s3,s4,s5,s6,s7);

temp2=CPUtime();

aggregation (micronw,node,degree,successor,length,
             noOfmacros,shortest,micronetwork,noofconnect,connect);

macroShort (noOfmacros,node,macronode,macrolength,macrodegree,
            macrosuccessor,shortest);

temp3=CPUtime();

approximation(noOfNodes,noOfmacros,shortest,micronetwork,
              noofconnect,connect);

temp4=CPUtime();

time1=temp1;
time2=temp2-temp1;
time3=temp3-temp2;
time4=temp4-temp3;

printResults(outputname,noOfNodes,node,shortest,time1,time2,time3,time4);

}
```

```
/*********************************************************

This program "trsdijkstra.c" is designed to simulate
the on-line trip requests simulation based on the
SEMCOG road network data using the Dijkstra Algorithm

*********************************************************/

#include <stdio.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/times.h>

#define MAX_NO_OF_NODES 4735
#define MAX_CENTROIDS 1550
#define MAX_NO_OF_DEGREE 15
#define INFINITY 500000.0
#define STR_LENGTH 100
#define TICKS 100.0

#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)

double rand2 (idum)
  long *idum;
  {
  int j;
  long k;
  static long idum2=123456789;
  static long iy=0;
  static long iv[NTAB];
  float temp;

  if (*idum<=0)
    {
    if (-(*idum)<1) *idum=1;
    else *idum=-(*idum);
    idum2=(*idum);

    for (j=NTAB+7;j>=0;j--)
      {
      k=(*idum)/IQ1;
      *idum=IA1*(*idum-k*IQ1)-k*IR1;
      if (*idum<0) *idum += IM1;
      if (j<NTAB) iv[j]=*idum;
      }
    iy=iv[0];
    }
    k=(*idum)/IQ1;
    *idum=IA1*(*idum-k*IQ1)-k*IR1;
    if (*idum<0) *idum += IM1;
```

```
    k=idum2/IQ2;
    idum2=IA2*(idum2-k*IQ2)-k*IR2;
    if (idum2<0) idum2 += IM2;
    j=iy/NDIV;
    iy=iv[j]-idum2;
    iv[j]=*idum;
    if (iy<1) iy += IMM1;
    if ((temp=AM*iy)>RNMX) return RNMX;
    else return temp;
    }

void intialization (double pnooftrip[MAX_CENTROIDS][MAX_CENTROIDS],
                    float shortest[MAX_CENTROIDS][MAX_CENTROIDS])
   {
   int i,j;

   for (i=0;i<MAX_CENTROIDS;i++)
     for (j=0;j<MAX_CENTROIDS;j++)
       {
       shortest[i][j]=INFINITY;
       pnooftrip[i][j]=0.0;
       }

   }

void inputCentroid (char centroid[STR_LENGTH],
                    int oldID[MAX_NO_OF_NODES],
                    int degree[MAX_NO_OF_NODES],
                    int *noofcentroid)
  {
  int noofdata,i,j,dummyx,dummyy;
  FILE *fp;

  fp=fopen(centroid,"r");
   if (fp==NULL)
     {
     printf("**could not open %s\n",centroid);
     exit(0);
     }

  fscanf(fp,"%d",&noofdata);

  for (i=0;i<noofdata;i++)
    {
    fscanf(fp,"%d %d %d",&oldID[i],&dummyx,&dummyy);
    degree[i]=0;
    }
  fclose(fp);

  *noofcentroid=noofdata;
  }


void inputTripdata (char tripdata[STR_LENGTH],
                    int noofcentroid,
                    int oldID[MAX_NO_OF_NODES],
                    double pnooftrip[MAX_CENTROIDS][MAX_CENTROIDS])
  {
  int newA,newB,i,j,Anode,Bnode,dummy,centnode,zone;
  double cdf,trips;
  FILE *fp;

  fp=fopen(tripdata,"r");
   if (fp==NULL)
     {
     printf("**could not open %s\n",tripdata);
```

```
      exit(0);
      }

  while (!feof(fp))
    {
    fscanf(fp,"%d %d %d",&Anode,&Bnode,&dummy);
    trips=(double)dummy;

    j=0;
    while (Anode!=oldID[j] && j<noofcentroid)
      j=j+1;

    if (Anode==oldID[j])
      newA=j;
    else
      {
      printf("error:could not find node %d in triptable\n",Anode);
      exit(0);
      }

    j=0;
    while (Bnode!=oldID[j] && j<noofcentroid)
      j=j+1;

    if (Bnode==oldID[j])
      newB=j;
    else
      {
      printf("error:could not find node %d in triptable\n",Bnode);
      exit(0);
      }

    pnooftrip[newA][newB]=trips/12753236.0;
    }

  fclose(fp);

  cdf=0.0;
  for (i=0;i<noofcentroid;i++)
    for (j=0;j<noofcentroid;j++)
      if (pnooftrip[i][j]>0.0)
         {
         pnooftrip[i][j]=cdf+pnooftrip[i][j];
         cdf=pnooftrip[i][j];
         }

  }

void inputData (char str[STR_LENGTH],
                int noofcentroid,
                int *noOfNodes,
                int oldID[MAX_NO_OF_NODES],
                int degree[MAX_NO_OF_NODES],
                int successor[MAX_NO_OF_NODES][MAX_NO_OF_DEGREE],
                float length[MAX_NO_OF_NODES][MAX_NO_OF_DEGREE],
                float s1,
                float s2,
                float s3,
                float s4,
                float s5,
                float s6,
                float s7)
   {
   int i,j,k,dummy,noOfdata,totalnodes,newA,newB,Anode,Bnode,class,
       dummyA, dummyB,lane,xA,yA,xB,yB,tempID;
   float traveltime,speedlimit,temp;
```

```
FILE *fp;


fp=fopen(str,"r");
if (fp==NULL)
  {
  printf("couldn't open %s.\n",str);
  exit(0);
  }

fscanf(fp,"%d",&noOfdata);

totalnodes=noofcentroid;

for (i=0;i<noOfdata;i++)
  {
  fscanf(fp,"%d %d %d %d %d %d %d %d %d",&Anode,&Bnode,&class,&dummy,
         &lane,&xA,&yA,&xB,&yB);
  temp=(float) dummy;

  if (class==1)
    speedlimit=s1;
  else if (class==2)
    speedlimit=s2;
  else if (class==3)
    speedlimit=s3;
  else if (class==4)
    speedlimit=s4;
  else if (class==5)
    speedlimit=s5;
  else if (class==6)
    speedlimit=s6;
  else if (class==7)
    speedlimit=s7;

  traveltime=temp/speedlimit;

  j=0;

  while (Anode!=oldID[j] && j<totalnodes)
    j=j+1;

  if (Anode!=oldID[j])
    {
    oldID[totalnodes]=Anode;
    newA=totalnodes;
    degree[newA]=1;
    totalnodes=totalnodes+1;
    }
  else
    {
    newA=j;
    degree[newA]=degree[newA]+1;
    if (degree[newA]>MAX_NO_OF_DEGREE)
       {
       printf("degree of node %d is greater than MAX_NO_OF_DEGREE\n",
              oldID[newA]);
       exit(0);
       }
    }

  j=0;

  while (Bnode!=oldID[j] && j<totalnodes)
    j=j+1;
```

```
      if (Bnode!=oldID[j])
        {
        oldID[totalnodes]=Bnode;
        newB=totalnodes;
        degree[newB]=1;
        totalnodes=totalnodes+1;
        }
      else
        {
        newB=j;
        degree[newB]=degree[newB]+1;
        if (degree[newB]>MAX_NO_OF_DEGREE)
           {
           printf("degree of node %d is greater than MAX_NO_OF_DEGREE\n",
                  oldID[newB]);
           exit(0);
           }
        }


      dummyA=degree[newA]-1;
      dummyB=degree[newB]-1;

      successor[newA][dummyA]=newB;
      length[newA][dummyA]=traveltime;

      successor[newB][dummyB]=newA;
      length[newB][dummyB]=traveltime;
      }

   fclose(fp);
   *noOfNodes=totalnodes;

   }

void shiftUp (int moveupitem,
              int heap[MAX_NO_OF_NODES],
              int index[MAX_NO_OF_NODES],
              float dshortest[MAX_NO_OF_NODES])
   {
   int parent,moveupnode,i,j;
   float temp;

   moveupnode=heap[moveupitem];

   parent=moveupitem/2;

   while (dshortest[heap[parent]]>dshortest[moveupnode] &&
          moveupitem>1)
     {
     heap[moveupitem]=heap[parent];
     index[heap[moveupitem]]=moveupitem;
     moveupitem=parent;
     parent=moveupitem/2;
     }

   index[moveupnode]=moveupitem;
   heap[moveupitem]=moveupnode;
   }

void shiftDown (int heapsize,
                int movedownitem,
                int index[MAX_NO_OF_NODES],
                int heap[MAX_NO_OF_NODES],
                float dshortest[MAX_NO_OF_NODES])
   {
```

```
    int child,movedownnode,i,j,dummy;

    movedownnode=heap[movedownitem];
    child=2*movedownitem;

    while (child<=heapsize)
      {
      if ((child+1)<=heapsize &&
          dshortest[heap[child+1]]<dshortest[heap[child]])
        child=child+1;

      if (dshortest[heap[child]]>=dshortest[movedownnode])
        return;
      else
        {
        heap[movedownitem]=heap[child];
        index[heap[movedownitem]]=movedownitem;
        movedownitem=child;
        index[movedownnode]=movedownitem;
        heap[movedownitem]=movedownnode;
        child=2*movedownitem;
        }
      }
  }

void insertItem (int *sizeofHeap,
                 int heap[MAX_NO_OF_NODES],
                 int index[MAX_NO_OF_NODES],
                 int insertnode,
                 float dshortest[MAX_NO_OF_NODES])
  {
  int i,j,moveupitem;

  *sizeofHeap=*sizeofHeap+1;
  heap[*sizeofHeap]=insertnode;
  index[insertnode]=*sizeofHeap;

  if (*sizeofHeap==1)
    return;

  shiftUp(*sizeofHeap,heap,index,dshortest);
  }

void deleteMin (int *sizeofHeap,
                int heap[MAX_NO_OF_NODES],
                int index[MAX_NO_OF_NODES],
                float dshortest[MAX_NO_OF_NODES])
  {
  int i,j;

  heap[1]=heap[*sizeofHeap];
  *sizeofHeap=*sizeofHeap-1;
  i=1;
  index[heap[1]]=1;
  shiftDown(*sizeofHeap,i,index,heap,dshortest);
  }


void Dijstra (int noOfNodes,
              int oldID[MAX_NO_OF_NODES],
              int degree[MAX_NO_OF_NODES],
              float dlength[MAX_NO_OF_NODES][MAX_NO_OF_DEGREE],
              int node[MAX_NO_OF_NODES][MAX_NO_OF_DEGREE],
              int root,
              int destination,
              float dshortest[MAX_NO_OF_NODES],
```

```
            int label[MAX_NO_OF_NODES])
{
int oldvalue,i, currentnode, T,k,sizeofHeap,
     check,index[MAX_NO_OF_NODES],
     dummynode,heap[MAX_NO_OF_NODES];
float  dummy;

dshortest[root] = 0.0;
label[root]=1;

for (i=0; i<noOfNodes; i++)
    if (i != root)
      {
       label[i] = 0;
       dshortest[i]=INFINITY;
      }

sizeofHeap=0;

for (k=0;k<degree[root];k++)
   {
   dummynode=node[root][k];
   dshortest[dummynode] = dlength[root][k];

   insertItem(&sizeofHeap,heap,index,dummynode,dshortest);

/*      predeccessor[dummynode]=root; */
   }

T = noOfNodes-1;

while ((T>0) && (currentnode!=destination))
   {
     currentnode = heap[1];
     label[currentnode]=1;

     for (k=0;k<degree[currentnode];k++)
        {
        i=node[currentnode][k];
        if (label[i]==0)
           {
           oldvalue=dshortest[i];

           dummy = dshortest[currentnode] + dlength[currentnode][k];
           if (dummy < dshortest[i])
              {
              dshortest[i] = dummy;

              if (oldvalue>=INFINITY)
                insertItem(&sizeofHeap,heap,index,i,dshortest);
              else
                shiftUp(index[i],heap,index,dshortest);
              }
           }
        }
     deleteMin(&sizeofHeap,heap,index,dshortest);

     T=T-1;
   }

}

float CPUtime (void)
   {
   float cpuTime;
   struct tms localTimeStruct;
```

```
   times (&localTimeStruct);

   cpuTime = (localTimeStruct.tms_utime + localTimeStruct.tms_stime) / TICKS;
   return cpuTime;
   }


void reInitialization (float shortest[MAX_CENTROIDS][MAX_CENTROIDS],
                       int noofcentroid)
  {
  int i,j;

  for (i=0;i<noofcentroid;i++)
    for (j=0;j<noofcentroid;j++)
       shortest[i][j]=INFINITY;

  }

void Locate(double pnooftrip[MAX_CENTROIDS][MAX_CENTROIDS],
            double randomNo,
            int *o,
            int *d,
            int noofcentroid)
   {
   int k,ilow,jlow,iupper,jupper,i,j,dummy,midpoint,lowbound,upperbound;

   i=noofcentroid-1;
   j=noofcentroid-1;
   while(pnooftrip[i][j]<=0.0)
     j=j-1;

   if (j<0)
     {
     printf("warnning !!\n");
     exit(0);
     }

   upperbound=noofcentroid*i+j+1;

   i=0;
   j=0;
   while (pnooftrip[i][j]<=0.0)
     j=j+1;

   if (j>noofcentroid-1)
     {
     printf("warnning !!\n");
     exit(0);
     }

   lowbound=noofcentroid*i+j+1;
   ilow=i;
   jlow=j;

   for (k=0;k<11;k++)
     {
     midpoint=(upperbound+lowbound)/2;
     i=(midpoint-1)/noofcentroid;
     j=(midpoint-1)%noofcentroid;
     while (pnooftrip[i][j]<=0.0)
       if (j==0)
         {
         j=noofcentroid-1;
         i=i-1;
         }
```

```
        else
          j=j-1;

      if (randomNo>pnooftrip[i][j])
        {
        lowbound=i*noofcentroid+j+1;
        ilow=i;
        jlow=j;
        }
      else
        {
        upperbound=i*noofcentroid+j+1;
        iupper=i;
        jupper=j;
        }
      }

   while (randomNo>pnooftrip[ilow][jlow])
      if (jlow==noofcentroid-1)
        {
        ilow=ilow+1;
        jlow=0;
        }
      else
        jlow=jlow+1;

   *o=ilow;
   *d=jlow;
   }

void generateAllPaths (int noofcentroid,
                       int noOfNodes,
                       float *totaleng,
                       int oldID[MAX_NO_OF_NODES],
                       float length[MAX_NO_OF_NODES][MAX_NO_OF_DEGREE],
                       int successor[MAX_NO_OF_NODES][MAX_NO_OF_DEGREE],
                       int degree[MAX_NO_OF_NODES],
                       float shortest[MAX_CENTROIDS][MAX_CENTROIDS],
                       int o,
                       int d)
   {
   int label[MAX_NO_OF_NODES],j;
   float dshort[MAX_NO_OF_NODES],temp_total;


   if (shortest[o][d]<INFINITY)
         *totaleng=*totaleng+shortest[o][d];
   else
      {
      Dijstra(noOfNodes,oldID, degree,length,successor,o,d,dshort,label);
      *totaleng=*totaleng+dshort[d];

      for (j=0;j<noofcentroid;j++)
         if (label[j]==1)
             shortest[o][j]=dshort[j];
      }


  }
/*
void printResults (FILE *fp,
                   float time)
   {
   int count1,i,j,totalpairs,count;
   float temp_trip,timeslice;
```

```
    count=0;
    for (i=0;i<noOfNodes;i++)
     {
     count=count+1;
     fprintf(fp,"%5d",node[i]);
     if (count%13==0)
       fprintf(fp,"\n");
     }
    fprintf(fp,"\n\n");

    for (i=0;i<noOfNodes;i++)
     {
     fprintf(fp,"%d\t",node[i]);
     count=0;
     for (j=0;j<noOfNodes;j++)
       {
      count=count+1;
       fprintf(fp,"%5d",node[predeccessor[i][j]]);
       if (count%13==0)
         fprintf(fp,"\n");
       }
     fprintf(fp,"\n");
     }

    fprintf(fp,"\n\n");

    fprintf(fp,"%.8f\t%f\t%f\n",timeslice,avgleng,time);

    }*/

  void main (void)
    {
    int j,noOfNodes,i,successor[MAX_NO_OF_NODES][MAX_NO_OF_DEGREE],
        degree[MAX_NO_OF_NODES],oldID[MAX_NO_OF_NODES],totaltrips,
        o,d,seed,noOftrips,noofcentroid;
    long *idum;
    float r, length[MAX_NO_OF_NODES][MAX_NO_OF_DEGREE],avgleng,
         time,temp1,temp2,temp3,temp4,temp5,time1,time2,time3,time4,time5,
         shortest[MAX_CENTROIDS][MAX_CENTROIDS],total_length,temp6,time6,
         searchtime,temp_trip,s1,s2,s3,s4,s5,s6,s7,totaleng;
    double pnooftrip[MAX_CENTROIDS][MAX_CENTROIDS],randomNo;
    char inputname[STR_LENGTH], outputname[STR_LENGTH],str3[STR_LENGTH],
         centroid[STR_LENGTH],tripdata[STR_LENGTH];
    FILE *fp1, *fp;

    printf("start dijkstra\n");
    fp1 = fopen("inputdijkstra", "r");
    if (fp1==NULL)
       {
       printf (" -> Could not open file <inputdijkstra>\n");
       exit(0);
       }
    fscanf(fp1,"%s%s%s%s%f%f%f%f%f%f%f%d",inputname,outputname,centroid,tripdata,
           &s1,&s2,&s3,&s4,&s5,&s6,&s7,&seed);
    fclose(fp1);

    idum=&seed;

    fp = fopen(outputname, "w");
    if (fp==NULL)
       {
       printf (" -> Could not open file %s\n",outputname);
       exit(0);
       }

    intialization (pnooftrip,shortest);
```

```
temp1=CPUtime();

inputCentroid (centroid,oldID,degree,&noofcentroid);

temp2=CPUtime();

inputTripdata (tripdata,noofcentroid,oldID,pnooftrip);

temp3=CPUtime();

inputData(inputname,noofcentroid,&noOfNodes,oldID,degree,successor,length,
          s1,s2,s3,s4,s5,s6,s7);

temp4=CPUtime();

time1=temp1;
time2=temp2-temp1;
time3=temp3-temp2;
time4=temp4-temp3;
fprintf(fp,"time 1: %f\n",time1);
fprintf(fp,"time 2: %f\n",time2);
fprintf(fp,"time 3: %f\n",time3);
fprintf(fp,"time 4: %f\n",time4);

totaleng=0.0;

for (i=1;i<1201;i++)
  {
  if (i%30==0)
    printf("%d\n",i);

  searchtime=0.0;
  temp2=CPUtime();

  for (j=0;j<148;j++)
    {
    temp3=CPUtime();
    randomNo=rand2(idum);
    Locate(pnooftrip,randomNo,&o,&d,noofcentroid);
    temp4=CPUtime();

    generateAllPaths(noofcentroid,noOfNodes,
                  &totaleng,oldID,length,successor,
                  degree,shortest,o,d);

    searchtime=searchtime+temp4-temp3;
    }

  temp5=CPUtime();
  time=temp5-temp2-searchtime;
  fprintf(fp,"%.9f\t",time);
  noOftrips=i*148;
  temp_trip=(float) noOftrips;
  avgleng=totaleng/temp_trip;
  fprintf(fp,"%f\n",avgleng);
  }


printf("finish dijkstra\n");
fclose(fp);
}
```

```
/*****************************************************
This program "trshaa5.c" is designed to simulate the
on-line trip requests simulation based on the SEMCOG
road network data using HAA5 (Best HA)
*****************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/times.h>

#define MAX_NO_OF_NODES 4733
#define MAX_CENTROIDS 1543
#define MAX_NO_OF_MACROS 160
#define MAX_NO_OF_inMACROS 345
#define MAX_NW 23
#define MAX_inCENTROIDS 140
#define MAX_NO_OF_CONNECT 40
#define MAX_NO_OF_DEGREE 15
#define INFINITY 500000.0
#define STR_LENGTH 100
#define TICKS 100.0

#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)

double rand2 (idum)
  long *idum;
  {
  int j;
  long k;
  static long idum2=123456789;
  static long iy=0;
  static long iv[NTAB];
  float temp;

  if (*idum<=0)
    {
    if (-(*idum)<1) *idum=1;
    else *idum=-(*idum);
    idum2=(*idum);

    for (j=NTAB+7;j>=0;j--)
      {
      k=(*idum)/IQ1;
      *idum=IA1*(*idum-k*IQ1)-k*IR1;
      if (*idum<0) *idum += IM1;
      if (j<NTAB) iv[j]=*idum;
      }
    iy=iv[0];
    }
    k=(*idum)/IQ1;
    *idum=IA1*(*idum-k*IQ1)-k*IR1;
```

```
    if (*idum<0) *idum += IM1;
    k=idum2/IQ2;
    idum2=IA2*(idum2-k*IQ2)-k*IR2;
    if (idum2<0) idum2 += IM2;
    j=iy/NDIV;
    iy=iv[j]-idum2;
    iv[j]=*idum;
    if (iy<1) iy += IMM1;
    if ((temp=AM*iy)>RNMX) return RNMX;
    else return temp;
    }

void intialization (double pnooftrip[MAX_CENTROIDS][MAX_CENTROIDS],
                    float shortest[MAX_CENTROIDS][MAX_CENTROIDS],
                    float disttocent[MAX_CENTROIDS][MAX_NO_OF_CONNECT],
                    float distfrcent[MAX_CENTROIDS][MAX_NO_OF_CONNECT],
                    float macroshort[MAX_NO_OF_MACROS][MAX_NO_OF_MACROS])
   {
   int i,j;

   for (i=0;i<MAX_CENTROIDS;i++)
     for (j=0;j<MAX_NO_OF_CONNECT;j++)
         {
         distfrcent[i][j]=INFINITY;
         disttocent[i][j]=INFINITY;
         }

   for (i=0;i<MAX_NO_OF_MACROS;i++)
     for (j=0;j<MAX_NO_OF_MACROS;j++)
        macroshort[i][j]=INFINITY;

   for (i=0;i<MAX_CENTROIDS;i++)
     for (j=0;j<MAX_CENTROIDS;j++)
       {
       shortest[i][j]=INFINITY;
       pnooftrip[i][j]=0.0;
       }
   }

void inputCentroid (char centroid[STR_LENGTH],
                    int node[MAX_NO_OF_NODES],
                    int *noofcentroid)
  {
  int noofdata,i,j,dummyx,dummyy;
  FILE *fp;

  fp=fopen(centroid,"r");
   if (fp==NULL)
     {
     printf("**could not open %s\n",centroid);
     exit(0);
     }

  fscanf(fp,"%d",&noofdata);

  for (i=0;i<noofdata;i++)
    fscanf(fp,"%d %d %d",&node[i],&dummyx,&dummyy);

  fclose(fp);

  *noofcentroid=noofdata;
  }


void inputTripdata (char tripdata[STR_LENGTH],
                    int noofcentroid,
```

```
                    int node[MAX_NO_OF_NODES],
                    double pnooftrip[MAX_CENTROIDS][MAX_CENTROIDS])
{
int check,totalnode,newA,newB,i,j,Anode,dummy,Bnode,centnode,zone;
double cdf,trips,totaltrip;
FILE *fp;

fp=fopen(tripdata,"r");
 if (fp==NULL)
   {
   printf("**could not open %s\n",tripdata);
   exit(0);
   }

while (!feof(fp))
  {
  fscanf(fp,"%d %d %d",&Anode,&Bnode,&dummy);
  trips=(double)dummy;

  j=0;
  while (Anode!=node[j] && j<noofcentroid)
    j=j+1;

  if (Anode==node[j])
    newA=j;
  else
    {
    printf("error:could not find node %d in triptable\n",Anode);
    exit(0);
    }

  j=0;
  while (Bnode!=node[j] && j<noofcentroid)
    j=j+1;

  if (Bnode==node[j])
    newB=j;
  else
    {
    printf("error:could not find node %d in triptable\n",Bnode);
    exit(0);
    }
  pnooftrip[newA][newB]=trips/12753236.0;
  }

fclose(fp);

cdf=0.0;
for (i=0;i<noofcentroid;i++)
  for (j=0;j<noofcentroid;j++)
    if (pnooftrip[i][j]>0.0)
      {
      pnooftrip[i][j]=cdf+pnooftrip[i][j];
      cdf=pnooftrip[i][j];
      }
}

void macroNetwork (char macronw[STR_LENGTH],
                   int *totalmacros,
                   int noofcentroid,
                   int node[MAX_NO_OF_NODES],
                   int degree[MAX_NO_OF_NODES],
                   int macronode[MAX_NO_OF_MACROS],
                   float macrolength[MAX_NO_OF_MACROS][MAX_NO_OF_DEGREE],
                   int macrodegree[MAX_NO_OF_MACROS],
                   int macrosuccessor[MAX_NO_OF_MACROS][MAX_NO_OF_DEGREE],
```

```
                float s1)
{
int lastnode,Anode,Bnode,class,i,j,k,noofmacros,dummy,newA,newB,dummyA,
    lane,xA,xB,yA,yB,dummyB,tempID,noofdata,newnodeA,newnodeB;
float traveltime,mlength;
FILE *fp;

fp=fopen(macronw,"r");
if (fp==NULL)
  {
  printf("**could not open %s\n",macronw);
  exit(0);
  }

fscanf(fp,"%d",&noofdata);
*totalmacros=0;
node[noofcentroid]=0;

for (i=0;i<noofdata;i++)
  {
  fscanf(fp,"%d %d %d %d %d %d %d %d %d",&Anode,&Bnode,&class,&dummy,
         &lane,&xA,&yA,&xB,&yB);

  mlength=(float) dummy;
  traveltime=mlength/s1;


  j=noofcentroid;
  while (Anode!=node[j] && j<*totalmacros+noofcentroid)
    j=j+1;

  if (Anode!=node[j])
    {
    newnodeA=*totalmacros;
    newA=*totalmacros+noofcentroid;
    macronode[newnodeA]=newA;
    macrodegree[newnodeA]=1;
    node[newA]=Anode;
    degree[newA]=0;
    *totalmacros=*totalmacros+1;
    }
  else
    {
    newnodeA=j-noofcentroid;
    macrodegree[newnodeA]=macrodegree[newnodeA]+1;
    }

  j=noofcentroid;
  while (Bnode!=node[j] && j<*totalmacros+noofcentroid)
    j=j+1;

  if (Bnode!=node[j])
    {
    newnodeB=*totalmacros;
    newB=*totalmacros+noofcentroid;
    macronode[newnodeB]=newB;
    macrodegree[newnodeB]=1;
    node[newB]=Bnode;
    degree[newB]=0;
    *totalmacros=*totalmacros+1;
    }
  else
    {
    newnodeB=j-noofcentroid;
    macrodegree[newnodeB]=macrodegree[newnodeB]+1;
    }
```

```
      dummyA=macrodegree[newnodeA]-1;
      dummyB=macrodegree[newnodeB]-1;

      macrosuccessor[newnodeA][dummyA]=newnodeB;
      macrolength[newnodeA][dummyA]=traveltime;

      macrosuccessor[newnodeB][dummyB]=newnodeA;
      macrolength[newnodeB][dummyB]=traveltime;
      }

   fclose(fp);
   }

void Network (int Anode,
              int Bnode,
              float leng,
              int *totalnodes,
              int node[MAX_NO_OF_NODES],
              float length[MAX_NO_OF_NODES][MAX_NO_OF_DEGREE],
              int degree[MAX_NO_OF_NODES],
              int successor[MAX_NO_OF_NODES][MAX_NO_OF_DEGREE])
   {
   int i,j,k,newA,newB,dummy,dummyA, dummyB,tempID;

   j=0;
   while (Anode!=node[j] && j<*totalnodes)
     j=j+1;

   if (Anode!=node[j])
     {
     node[*totalnodes]=Anode;
     degree[*totalnodes]=1;
     newA=*totalnodes;
     *totalnodes=*totalnodes+1;
     }
   else
     {
     newA=j;
     degree[newA]=degree[newA]+1;
     if (degree[newA]>MAX_NO_OF_DEGREE)
       {
       printf("degree of node %d is greater than MAX_NO_OF_DEGREE\n",
              Anode);
       exit(0);
       }
     }

   j=0;
   while (Bnode!=node[j] && j<*totalnodes)
     j=j+1;

   if (Bnode!=node[j])
     {
     node[*totalnodes]=Bnode;
     degree[*totalnodes]=1;
     newB=*totalnodes;
     *totalnodes=*totalnodes+1;
     }
   else
     {
     newB=j;
     degree[newB]=degree[newB]+1;
     if (degree[newB]>MAX_NO_OF_DEGREE)
       {
```

```
        printf("degree of node %d is greater than MAX_NO_OF_DEGREE\n",
              Bnode);
        exit(0);
        }
      }


   dummyA=degree[newA]-1;
   dummyB=degree[newB]-1;

   successor[newA][dummyA]=newB;
   length[newA][dummyA]=leng;

   successor[newB][dummyB]=newA;
   length[newB][dummyB]=leng;
   }


void inputNetData (char str[STR_LENGTH],
                   char str1[STR_LENGTH],
                   char macronw[STR_LENGTH],
                   int *noOfmacros,
                   int *noOfNodes,
                   int noofcentroid,
                   int node[MAX_NO_OF_NODES],
                   int macronode[MAX_NO_OF_MACROS],
                   int macrodegree[MAX_NO_OF_MACROS],
                   int macrosuccessor[MAX_NO_OF_MACROS][MAX_NO_OF_DEGREE],
                   float macrolength[MAX_NO_OF_MACROS][MAX_NO_OF_DEGREE],
                   int microdegree[MAX_NW][MAX_NO_OF_inMACROS],
                   int micronode[MAX_NW][MAX_NO_OF_inMACROS],
                   float microlength[MAX_NW][MAX_NO_OF_inMACROS][MAX_NO_OF_DEGREE],
                   int microsucc[MAX_NW][MAX_NO_OF_inMACROS][MAX_NO_OF_DEGREE],
                   int network[MAX_CENTROIDS],
                   int newID[MAX_CENTROIDS],
                   int noofconnect[MAX_NW],
                   int nocentroids[MAX_NW],
                   int totalnode[MAX_NW],
                   float s1,
                   float s2,
                   float s3,
                   float s4,
                   float s5,
                   float s6,
                   float s7)
   {
   int j,i,dummy,noOfdata,Anode,Bnode,class,totalmacros,newA,newB,
       temp_node,lane,xA,yA,xB,yB,checkA,checkB,degree[MAX_NO_OF_NODES],
       successor[MAX_NO_OF_NODES][MAX_NO_OF_DEGREE],totalnodes,
       count,k,nomicronetwork;
   float speedlimit,length[MAX_NO_OF_NODES][MAX_NO_OF_DEGREE],
         distance,traveltime;
   FILE *fp,*fp1;

   fp=fopen(str,"r");
   if (fp==NULL)
     {
     printf("couldn't open %s.\n",str);
     exit(0);
     }

   for (i=0;i<noofcentroid;i++)
     degree[i]=0;

   macroNetwork(macronw,&totalmacros,noofcentroid,node,degree,macronode,
            macrolength,macrodegree,macrosuccessor,s1);
```

```
*noOfmacros=totalmacros;

fscanf(fp,"%d",&noOfdata);

totalnodes=totalmacros+noofcentroid;

for (i=0;i<noOfdata;i++)
  {
  fscanf(fp,"%d %d %d %d %d %d %d %d %d",&Anode,&Bnode,&class,&dummy,
         &lane,&xA,&yA,&xB,&yB);
  distance=(float) dummy;

  if (class==1)
    speedlimit=s1;
  else if (class==2)
    speedlimit=s2;
  else if (class==3)
    speedlimit=s3;
  else if (class==4)
    speedlimit=s4;
  else if (class==5)
    speedlimit=s5;
  else if (class==6)
    speedlimit=s6;
  else if (class==7)
    speedlimit=s7;

  traveltime=distance/speedlimit;

  Network(Anode,Bnode,traveltime,&totalnodes,node,
          length,degree,successor);

  }
fclose(fp);

*noOfNodes=totalnodes;

fp=fopen(str1,"r");
if (fp==NULL)
  {
  printf("couldn't open %s.\n",str1);
  exit(0);
  }

fscanf(fp,"%d",&nomicronetwork);

for (i=0;i<nomicronetwork;i++)
  {
  fscanf(fp,"%d %d %d",&totalnode[i],&nocentroids[i],&noofconnect[i]);

  for (j=0;j<nocentroids[i];j++)
    {
    fscanf(fp,"%d",&temp_node);

    k=0;
    while (temp_node!=node[k])
       k=k+1;

    micronode[i][j]=k;
    newID[k]=j;
    network[k]=i;
    }
  for (j=nocentroids[i];j<totalnode[i];j++)
    {
    fscanf(fp,"%d",&temp_node);
```

```
      k=0;
      while (temp_node!=node[k])
         k=k+1;

      micronode[i][j]=k;
      }


    for (j=0;j<totalnode[i];j++)
      {
      temp_node=micronode[i][j];
      microdegree[i][j]=degree[temp_node];
      for(count=0;count<degree[temp_node];count++)
          {
          dummy=successor[temp_node][count];

          k=0;
          while (dummy!=micronode[i][k] && k<totalnode[i])
            k=k+1;

          if (k>totalnode[i]-1)
            {
            microlength[i][j][count]=INFINITY;
            microsucc[i][j][count]=-1;
            }
          else
            {
            microlength[i][j][count]=length[temp_node][count];
            microsucc[i][j][count]=k;
            }
          }

      }

   }

   fclose(fp);


  }

/*----- makeheap----------------------------*/

void shiftUp (int moveupitem,
              int heap[MAX_NO_OF_inMACROS],
              int index[MAX_NO_OF_inMACROS],
              float dshortest[MAX_NO_OF_inMACROS])
   {
   int parent,moveupnode,i,j;
   float temp;

   moveupnode=heap[moveupitem];

   parent=moveupitem/2;

   while (dshortest[heap[parent]]>dshortest[moveupnode] &&
          moveupitem>1)
     {
     heap[moveupitem]=heap[parent];
     index[heap[moveupitem]]=moveupitem;
     moveupitem=parent;
     parent=moveupitem/2;
     }

   index[moveupnode]=moveupitem;
```

```
    heap[moveupitem]=moveupnode;
    }

void shiftDown (int heapsize,
                int movedownitem,
                int index[MAX_NO_OF_inMACROS],
                int heap[MAX_NO_OF_inMACROS],
                float dshortest[MAX_NO_OF_inMACROS])
    {
    int child,movedownnode,i,j,dummy;

    movedownnode=heap[movedownitem];
    child=2*movedownitem;

    while (child<=heapsize)
      {
      if ((child+1)<=heapsize &&
          dshortest[heap[child+1]]<dshortest[heap[child]])
        child=child+1;

      if (dshortest[heap[child]]>=dshortest[movedownnode])
        return;
      else
        {
        heap[movedownitem]=heap[child];
        index[heap[movedownitem]]=movedownitem;
        movedownitem=child;
        index[movedownnode]=movedownitem;
        heap[movedownitem]=movedownnode;
        child=2*movedownitem;
        }
      }
  }

void insertItem (int *sizeofHeap,
                 int heap[MAX_NO_OF_inMACROS],
                 int index[MAX_NO_OF_inMACROS],
                 int insertnode,
                 float dshortest[MAX_NO_OF_inMACROS])
  {
  int i,j,moveupitem;

  *sizeofHeap=*sizeofHeap+1;
  heap[*sizeofHeap]=insertnode;
  index[insertnode]=*sizeofHeap;

  if (*sizeofHeap==1)
    return;

  shiftUp(*sizeofHeap,heap,index,dshortest);
  }

void deleteMin (int *sizeofHeap,
                int heap[MAX_NO_OF_inMACROS],
                int index[MAX_NO_OF_inMACROS],
                float dshortest[MAX_NO_OF_inMACROS])
  {
  int i,j;

  heap[1]=heap[*sizeofHeap];
  *sizeofHeap=*sizeofHeap-1;
  i=1;
  index[heap[1]]=1;
  shiftDown(*sizeofHeap,i,index,heap,dshortest);
  }
```

```
/*-----------makeheap----------------------------*/

void Dijstra (int nw,
              int noofnodes,
              float dlength[MAX_NO_OF_inMACROS][MAX_NO_OF_DEGREE],
              int dsuccessor[MAX_NO_OF_inMACROS][MAX_NO_OF_DEGREE],
              int ddegree[MAX_NO_OF_inMACROS],
              int root,
              int destination,
              float dshortest[MAX_NO_OF_inMACROS],
              int label[MAX_NO_OF_inMACROS])
   {
   int oldvalue,i,count, currentnode, T,k,
       sizeofHeap,index[MAX_NO_OF_inMACROS],heap[MAX_NO_OF_inMACROS],
       check,dummynode;
   float  dummy;
   FILE *fp;

   label[root] = 1;
   dshortest[root] = 0.0;

   for (i=0; i<noofnodes; i++)
       if (i != root)
         {
          label[i] = 0;
          dshortest[i]=INFINITY;
         }
   sizeofHeap=0;

   for (k=0;k<ddegree[root];k++)
      {
      dummynode=dsuccessor[root][k];
      if (dummynode>=0)
        {
        dshortest[dummynode] = dlength[root][k];
        insertItem(&sizeofHeap,heap,index,dummynode,dshortest);
        }
      }

   T = noofnodes-1;
   while ((T>0) && (currentnode!=destination))
      {
        if (sizeofHeap<=0  && nw>=0)
          {
          printf("micronetwork %d is not completely connected ! \n",nw);
          exit(0);
          }
         else if (sizeofHeap<=0 && nw<0)
          {
          printf("macronetwork is not completely connected !! \n");
          exit (0);
          }

        currentnode=heap[1];
        label[currentnode]=1;

        for (k=0;k<ddegree[currentnode];k++)
           {
           i=dsuccessor[currentnode][k];
           if (label[i]==0 && i>=0)
              {
              oldvalue=dshortest[i];

              dummy = dshortest[currentnode] +
                      dlength[currentnode][k];
              if (dummy < dshortest[i])
```

```
                {
                dshortest[i] = dummy;
                if (oldvalue>=INFINITY)
                  insertItem(&sizeofHeap,heap,index,i,dshortest);
                else
                  shiftUp(index[i],heap,index,dshortest);
                }
              }
            }

        deleteMin(&sizeofHeap,heap,index,dshortest);

        T=T-1;
      }

   }

void macroShort (int noOfmacros,
                 int macronode[MAX_NO_OF_MACROS],
                 float macrolength[MAX_NO_OF_MACROS][MAX_NO_OF_DEGREE],
                 int macrodegree[MAX_NO_OF_MACROS],
                 int macrosuccessor[MAX_NO_OF_MACROS][MAX_NO_OF_DEGREE],
                 float macroshort[MAX_NO_OF_MACROS][MAX_NO_OF_MACROS])
   {
   int mroot,i,j,totalpair,dummy,p,destination,label[MAX_NO_OF_inMACROS];

   dummy=-1;
   destination=noOfmacros+1;

   for(mroot=0;mroot<noOfmacros;mroot++)
      Dijstra(dummy,noOfmacros,macrolength,macrosuccessor,
              macrodegree,mroot,destination,macroshort[mroot],label);


   }

float CPUtime (void)
   {
   float cpuTime;
   struct tms localTimeStruct;

   times (&localTimeStruct);

   cpuTime = (localTimeStruct.tms_utime + localTimeStruct.tms_stime) / TICKS;
   return cpuTime;
   }

void reInitilization (float shortest[MAX_CENTROIDS][MAX_CENTROIDS],
                      float distfrcent[MAX_CENTROIDS][MAX_NO_OF_CONNECT],
                      float disttocent[MAX_CENTROIDS][MAX_NO_OF_CONNECT],
                      int noofcentroid)
   {
   int i,j,k;

   for (i=0;i<noofcentroid;i++)
     for (j=0;j<MAX_NO_OF_CONNECT;j++)
         {
         distfrcent[i][j]=INFINITY;
         disttocent[i][j]=INFINITY;
         }

   for (i=0;i<noofcentroid;i++)
     for (j=0;j<noofcentroid;j++)
       shortest[i][j]=INFINITY;
   }
```

```
void Locate(double pnooftrip[MAX_CENTROIDS][MAX_CENTROIDS],
            double randomNo,
            int *o,
            int *d,
            int noofcentroid)
   {
   int k,ilow,jlow,iupper,jupper,i,j,dummy,midpoint,lowbound,upperbound;

   i=noofcentroid-1;
   j=noofcentroid-1;
   while(pnooftrip[i][j]<=0.0)
     j=j-1;

   if (j<0)
     {
     printf("warnning !!\n");
     exit(0);
     }

   upperbound=noofcentroid*i+j+1;

   i=0;
   j=0;
   while (pnooftrip[i][j]<=0.0)
     j=j+1;

   if (j>noofcentroid-1)
     {
     printf("warnning !!\n");
     exit(0);
     }

   lowbound=noofcentroid*i+j+1;
   ilow=i;
   jlow=j;

   for (k=0;k<11;k++)
     {
     midpoint=(upperbound+lowbound)/2;
     i=(midpoint-1)/noofcentroid;
     j=(midpoint-1)%noofcentroid;
     while (pnooftrip[i][j]<=0.0)
       if (j==0)
         {
         j=noofcentroid-1;
         i=i-1;
         }
       else
         j=j-1;

     if (randomNo>pnooftrip[i][j])
       {
       lowbound=i*noofcentroid+j+1;
       ilow=i;
       jlow=j;
       }
     else
       {
       upperbound=i*noofcentroid+j+1;
       iupper=i;
       jupper=j;
       }
     }

   while (randomNo>pnooftrip[ilow][jlow])
      if (jlow==noofcentroid-1)
```

```
        {
        ilow=ilow+1;
        jlow=0;
        }
      else
        jlow=jlow+1;

   *o=ilow;
   *d=jlow;
   }

void approximation (int noofcentroid,
                    int noOfmacros,
                    int macronode[MAX_NO_OF_MACROS],
                    int macrodegree[MAX_NO_OF_MACROS][MAX_NO_OF_DEGREE],
                    int macrosuccessor[MAX_NO_OF_MACROS][MAX_NO_OF_DEGREE],
                    float macrolength[MAX_NO_OF_MACROS][MAX_NO_OF_DEGREE],
                    int microdegree[MAX_NW][MAX_NO_OF_inMACROS],
                    int micronode[MAX_NW][MAX_NO_OF_inMACROS],
                    float microlength[MAX_NW][MAX_NO_OF_inMACROS][MAX_NO_OF_DEGREE],
                    int microsucc[MAX_NW][MAX_NO_OF_inMACROS][MAX_NO_OF_DEGREE],
                    int network[MAX_CENTROIDS],
                    int totalnode[MAX_NW],
                    int nocentroids[MAX_NW],
                    int noofconnect[MAX_NW],
                    int newID[MAX_CENTROIDS],
                    int node[MAX_NO_OF_NODES],
                    int o,
                    int d,
                    int frcent[MAX_CENTROIDS][MAX_NO_OF_CONNECT],
                    int tocent[MAX_CENTROIDS][MAX_NO_OF_CONNECT],
                    float disttocent[MAX_CENTROIDS][MAX_NO_OF_CONNECT],
                    float distfrcent[MAX_CENTROIDS][MAX_NO_OF_CONNECT],
                    float *totaleng,
                    float macroshort[MAX_NO_OF_MACROS][MAX_NO_OF_MACROS],
                    float shortest[MAX_CENTROIDS][MAX_CENTROIDS])
   {
   int  i,l,dummyi,dummyj,totalpairs,dummy,j,count,k,noofnextnw,
        temp1,root,label[MAX_NO_OF_inMACROS],rootID,destination;
   float temp,localshort[MAX_NO_OF_inMACROS],temp_shortest,temp_t1,temp_t2;


   if (shortest[o][d]<INFINITY)
         *totaleng=*totaleng+shortest[o][d];
   else
         {
         if (network[o]==network[d])
           {
           Dijstra(network[o],totalnode[network[o]],microlength[network[o]],
                   microsucc[network[o]],
                   microdegree[network[o]],newID[o],newID[d],localshort,label);

           for (j=0;j<nocentroids[network[o]];j++)
             if (label[j]==1)
               shortest[o][micronode[network[o]][j]]=localshort[j];

           for (j=0;j<noofconnect[network[o]];j++)
             if (label[j]==1)
               {
               temp1=j+nocentroids[network[o]];
               frcent[o][j]=micronode[network[o]][temp1]-noofcentroid;
               distfrcent[o][j]=localshort[temp1];
               }

           *totaleng=*totaleng+shortest[o][d];
           }
```

```
        else
          {
          if (distfrcent[o][0]>=INFINITY)
            {
            destination=totalnode[network[o]]+1;
            Dijstra(network[o],totalnode[network[o]],microlength[network[o]],
                  microsucc[network[o]],
                  microdegree[network[o]],newID[o],destination,localshort,label);

            for (j=0;j<nocentroids[network[o]];j++)
               shortest[o][micronode[network[o]][j]]=localshort[j];

            for (j=0;j<noofconnect[network[o]];j++)
              {
              temp1=j+nocentroids[network[o]];
              frcent[o][j]=micronode[network[o]][temp1]-noofcentroid;

              distfrcent[o][j]=localshort[temp1];
              }
            }

          if (disttocent[d][0]>=INFINITY)
            {
            destination=totalnode[network[d]]+1;
            for (root=0;root<noofconnect[network[d]];root++)
              {
              rootID=root+nocentroids[network[d]];
              Dijstra(network[d],totalnode[network[d]],microlength[network[d]],
                    microsucc[network[d]],microdegree[network[d]],
                    rootID,destination,localshort,label);

              for (j=0;j<nocentroids[network[d]];j++)
                {
                disttocent[micronode[network[d]][j]][root]=localshort[j];
                tocent[micronode[network[d]][j]][root]
                   =micronode[network[d]][rootID]-noofcentroid;
                }
              }
            }
          temp_shortest=INFINITY;

          for (l=0;l<noofconnect[network[o]];l++)
            for (j=0;j<noofconnect[network[d]];j++)
                {
                temp=distfrcent[o][l]+disttocent[d][j]+
                       macroshort[frcent[o][l]][tocent[d][j]];
                if (temp<temp_shortest)
                   temp_shortest=temp;
                }

          if (temp_shortest<shortest[o][d])
             shortest[o][d]=temp_shortest;
          *totaleng=*totaleng+shortest[o][d];
          }
        }

    }

/*
void printResults (FILE *fp,
                   float totaltime,
                   float searchtime,
                   float avgleng,
                   int noOftrips)
    {
    int count1,i,j,totalpairs,count;
```

```
   fp=fopen(str,"w");
   if (fp==NULL)
     {
     printf("couldn't open %s.\n",str);
     exit(0);
     }

   fprintf(fp,"predeccessors of nodes on the paths :\n\n\t");
   count=0;
   for (i=0;i<noOfNodes;i++)
    {
    count=count+1;
    fprintf(fp,"%5d",node[i]);
    if (count%13==0)
      fprintf(fp,"\n");
    }
   fprintf(fp,"\n\n");

   for (i=0;i<noOfNodes;i++)
    {
    fprintf(fp,"%d\t",node[i]);
    count=0;
    for (j=0;j<noOfNodes;j++)
      {
     count=count+1;
      fprintf(fp,"%5d",node[predeccessor[i][j]]);
      if (count%13==0)
        fprintf(fp,"\n");
      }
    fprintf(fp,"\n");
    }

   fprintf(fp,"\n\n");

   fprintf(fp,"%d\t%f\t%f\t%f\n",noOftrips,avgleng,totaltime,searchtime);
   }
*/

void main (void)
   {
   int node[MAX_NO_OF_NODES],macronode[MAX_NO_OF_MACROS],
       noOfNodes,microdegree[MAX_NW][MAX_NO_OF_inMACROS],
       micronode[MAX_NW][MAX_NO_OF_inMACROS],macrodegree[MAX_NO_OF_MACROS],
       noOfmacros,microsucc[MAX_NW][MAX_NO_OF_inMACROS][MAX_NO_OF_DEGREE],
       macrosuccessor[MAX_NO_OF_MACROS][MAX_NO_OF_DEGREE],
       i,j,noOftrips,noofcentroid,network[MAX_CENTROIDS],
       newID[MAX_CENTROIDS],o,d,frcent[MAX_CENTROIDS][MAX_NO_OF_CONNECT],
       tocent[MAX_CENTROIDS][MAX_NO_OF_CONNECT],
       totalnode[MAX_NW],nocentroids[MAX_NW],noofconnect[MAX_NW];
   float temp_trip,totaltime,time1,time2,time3,time4,
         microlength[MAX_NW][MAX_NO_OF_inMACROS][MAX_NO_OF_DEGREE],
         time7,temp7,temp1,temp2,temp3,temp4,time5,time6,temp5,temp6,
         avgleng,s1,s2,s3,s4,s5,s6,s7,searchtime,time,
         disttocent[MAX_CENTROIDS][MAX_NO_OF_CONNECT],
         distfrcent[MAX_CENTROIDS][MAX_NO_OF_CONNECT],
         macrolength[MAX_NO_OF_MACROS][MAX_NO_OF_DEGREE],
         shortest[MAX_CENTROIDS][MAX_CENTROIDS],totaleng,
         macroshort[MAX_NO_OF_MACROS][MAX_NO_OF_MACROS];
   long seed,*idum;
   double pnooftrip[MAX_CENTROIDS][MAX_CENTROIDS],randomNo;
   char inputname[STR_LENGTH], outputname[STR_LENGTH],centroid[STR_LENGTH],
        tripdata[STR_LENGTH],micronw[STR_LENGTH],macronw[STR_LENGTH];
   FILE *fp1;

   printf("start enumerate\n");
```

```
fp1 = fopen("inputenum", "r");
if (fp1==NULL)
   {
   printf (" -> Could not open file <inputenum>\n");
   exit(0);
   }

fscanf(fp1,"%s%s%s%s%s%s%f%f%f%f%f%f%f%ld",inputname,outputname,micronw,
   macronw,centroid,tripdata,&s1,&s2,&s3,&s4,&s5,&s6,&s7,&seed);

fclose(fp1);

idum=&seed;

fp1 = fopen(outputname, "w");
if (fp1==NULL)
   {
   printf (" -> Could not open file <%s>\n",outputname);
   exit(0);
   }

intialization (pnooftrip,shortest,disttocent,distfrcent,macroshort);

temp1=CPUtime();

inputCentroid(centroid,node,&noofcentroid);

temp2=CPUtime();

inputTripdata(tripdata,noofcentroid,node,pnooftrip);

temp3=CPUtime();

inputNetData (inputname,micronw,macronw,&noOfmacros,&noOfNodes,noofcentroid,
              node,macronode,macrodegree,macrosuccessor,macrolength,
              microdegree,micronode,microlength,microsucc,network,
              newID,noofconnect,nocentroids,totalnode,s1,s2,s3,s4,s5,
              s6,s7);

temp4=CPUtime();

time1=temp1;
time2=temp2-temp1;
time3=temp3-temp2;
time4=temp4-temp3;

fprintf(fp1,"time1:%f time2:%f time3:%f time4:%f\n",time1,time2,time3,time4);

totaleng=0.0;
temp2=CPUtime();
macroShort (noOfmacros,macronode,macrolength,macrodegree,
            macrosuccessor,macroshort);
temp3=CPUtime();
fprintf(fp1,"%.8f\n",temp3-temp2);

for (i=1;i<1201;i++)
  {
  searchtime=0.0;
  if (i%30==0)
     printf("%d\n",i);

  temp2=CPUtime();

  for (j=0;j<148;j++)
    {
    temp3=CPUtime();
```

```
    randomNo=rand2(idum);
    Locate(pnooftrip,randomNo,&o,&d,noofcentroid);
    temp4=CPUtime();

    approximation(noofcentroid,noOfmacros,macronode,macrodegree,
                macrosuccessor,macrolength,microdegree,micronode,microlength,
                microsucc,network,totalnode,nocentroids,
                noofconnect,newID,node,o,d,frcent,tocent,disttocent,distfrcent,
                &totaleng,macroshort,shortest);

    searchtime=searchtime+temp4-temp3;

    }
  temp5=CPUtime();
  time=temp5-temp2-searchtime;
  fprintf(fp1,"%.9f\t",time);
noOftrips=i*148;
temp_trip=(float)noOftrips;
avgleng=totaleng/temp_trip;
fprintf(fp1,"%f\n",avgleng);
  }

printf("finish enumerate\n");
fclose(fp1);
}
```