```c
#include<stdio.h>
#include<stdlib.h>

struct graph* gr=NULL;
struct mh* mhp=NULL;
struct node{
    int data;
    int w;
    struct node* next;
};

struct list{
    struct node* head;
};

struct graph{
    int v;
    struct list* arr;
};

struct node* cnode(int data,int w){
    struct node* temp=(struct node*)malloc(sizeof(struct node));
    temp->data=data;
    temp->w=w;
    temp->next=NULL;

     return temp;
}

struct graph* cgraph(int v){
    struct graph* gr=(struct graph*)malloc(sizeof(struct graph));
    gr->v=v;
    gr->arr=(struct list*)malloc(v*sizeof(struct list));
    for(int i=0;i<v;++i)
     gr->arr[i].head=NULL;

     return gr;

}
void addedge(int u,int v,int w){
    struct node* temp=cnode(v,w);
    temp->next=gr->arr[u].head;
    gr->arr[u].head=temp;

    temp=cnode(u,w);
    temp->next=gr->arr[v].head;
    gr->arr[v].head=temp;

    temp=NULL;
    free(temp);

}

struct mhn{
```

```
 55:     int v,dist;
 56:
 57: };
 58:
 59: struct mh{
 60:     int size,capacity;
 61:     int* pos;
 62:     struct mhn** arr;
 63:
 64: };
 65:
 66: struct mhn* cmhn(int v,int dist){
 67:     struct mhn* temp=(struct mhn*)malloc(sizeof(struct mhn));
 68:     temp->dist=dist;
 69:     temp->v=v;
 70:     return temp;
 71:
 72: }
 73:
 74: struct mh* cmh(int c){
 75:     struct mh* temp=(struct mh*)malloc(sizeof(struct mh));
 76:     temp->size=0;
 77:     temp->capacity=c;
 78:     temp->pos=(int*)malloc(c*sizeof(int));
 79:     temp->arr=(struct mhn**)malloc(c*sizeof(struct mhn*));
 80:       return temp;
 81: }
 82:
 83: void swap(struct mhn**a,struct mhn**b){
 84:     struct mhn* temp=*a;
 85:     *a=*b;
 86:     *b=temp;
 87: }
 88:
 89: void heapify(int pos){
 90:     int s,l,r;
 91:     s=pos;
 92:     l=2*pos+1;
 93:     r=2*pos+2;
 94:
 95:  int data;hp->size&&mhp->arr[l]->dist<mhp->arr[
 96:       s=l;
 97:
 98:     if(r<mhp->size&&mhp->arr[r]->dist<mhp->arr[s]->dist)
 99:        s=r;
100:
101:     if(s!=pos){
102:     struct mhn* sm=mhp->arr[s];
103:     struct mhn* ipos=mhp->arr[pos];
104:
105:     mhp->pos[ipos->v]=s;
106:     mhp->pos[sm->v]=pos;
107:
108:     swap(&mhp->arr[s],&mhp->arr[pos]);
```

```
109:     heapify(s);
110: 
111:      }
112: }
113: 
114: bool isempty(){
115:     return mhp->size==0;
116: }
117: 
118: bool ispresent(int v){
119:     if(mhp->pos[v]<mhp->size)
120:      return true;
121:    return false;
122: }
123: 
124: struct mhn* extract_min(){
125:  int data;
126:      return NULL;
127: 
128:     struct mhn* root=mhp->arr[0];
129:     struct mhn* lnode=mhp->arr[mhp->size-1];
130:     mhp->pos[lnode->v]=0;
131:     mhp->pos[root->v]=mhp->size-1;
132:     mhp->size--;
133: 
134:     mhp->arr[0]=lnode;
135: 
136:     heapify(0);
137:     return root;
138: 
139: }
140: 
141: 
142: void modify(int v,int dist){
143: 
144:     int i=mhp->pos[v];
145:     mhp->arr[i]->dist=dist;
146:     while(i&&(mhp->arr[i]->dist<mhp->arr[(i-1)/2]->dist)){
147: 
148:         mhp->pos[mhp->arr[i]->v]=(i-1)/2;
149:         mhp->pos[mhp->arr[(i-1)/2]->v]=i;
150:         swap(&mhp->arr[i],&mhp->arr[(i-1)/2]);
151: 
152:         i=(i-1)/2;
153: 
154:     }
155: 
156: }
157: void printArr(int dist[], int n)
158: {
159:     printf("Vertex   Distance from Source\n");
160:     for (int i = 0; i < n; ++i)
161:         printf("%d \t\t %d\n", i, dist[i]);
162: }
```

```
163:
164: void dijstra(int src){
165:     int dist[gr->v];
166:     mhp=cmh(gr->v);
167:     for(int i=0;i<gr->v;++i){
168:         dist[i]=INT_MAX;
169:         mhp->arr[i]=cmhn(i,dist[i]);
170:         mhp->pos[i]=i;
171:     }
172:
173:     mhp->arr[src]=cmhn(src,dist[src]);
174:     mhp->pos[src]=src;
175:     dist[src]=0;
176:     modify(src,dist[src]);
177:
178:     mhp->size=gr->v;
179:
180:      /* dist[src]=0;
181:     mhp->arr[src]=cmhn(0,dist[0]);
182:     mhp->pos[0]=0;*/
183:
184:     mhp->size=gr->v;
185:     while(!isempty()){
186:         struct mhn* temp=extract_min();
187:         int u=temp->v;
188:         struct node* temp1=gr->arr[u].head;
189:         while(temp1){
190:
191:             int v=temp1->data;
192:             if(ispresent(v)&&dist[v]>dist[u]+temp1->w)
193:             {
194:                 dist[v]=dist[u]+temp1->w;
195:                 modify(v,dist[v]);
196:
197:             }
198:
199:             temp1=temp1->next;
200:         }
201:
202:     }
203:     printArr(dist,gr->v);
204: }
205:
206: int main(){
207:     int V = 9;
208:     gr= cgraph(V);
209:     addedge( 0, 1, 4);
210:     addedge( 0, 7, 8);
211:     addedge( 1, 2, 8);
212:     addedge( 1, 7, 11);
213:     addedge( 2, 3, 7);
214:     addedge( 2, 8, 2);
215:     addedge( 2, 5, 4);
216:     addedge( 3, 4, 9);
```

```
217:     addedge( 3, 5, 14);
218:     addedge( 4, 5, 10);
219:     addedge( 5, 6, 2);
220:     addedge( 6, 7, 1);
221:     addedge(6, 8, 6);
222:     addedge(7, 8, 7);
223:
224:     dijstra(0);
225: }
226:
```