```
import java.awt.*;
import java.math.*;

class LI { // gerer les listes d'entiers
   int elt; LI next;
   static  LI cons( int x, LI tail)
        { LI cell=new LI(); cell.elt=x; cell.next=tail; return cell;}
   static  boolean isempty( LI liste) { return liste==null ; }
   static  int head( LI liste) { return liste.elt; }
   static  LI tail(  LI liste) { return liste.next; }
   static  LI iaj( int i, int j) { if (i==j) return cons(i, null); return cons(i
, iaj( i+1, j));}
}

class Pt {      double x, y;
        static Pt point( double u, double v) { Pt p=new Pt();  p.x=u; p.y=v; ret
urn p; }
}

class LP {   // GERER LES LISTES DE POINTS
   Pt elt; LP next;
   static  LP cons( Pt x, LP tail)
        { LP cell=new LP(); cell.elt=x; cell.next=tail; return cell;}
   static  boolean isempty( LP liste) { return liste==null ; }
   static  Pt head( LP liste) { return liste.elt; }
   static  LP tail(  LP liste) { return liste.next; }
}

class Arc { int v; double d;
        static Arc arc( int sommet, double distance)
        { Arc arc= new Arc(); arc.v=sommet; arc.d=distance; return arc;}
}
class LA  // GERER LES LISTES D'ARCS
{
   Arc elt;  LA next;
   static LA cons( Arc x, LA tail) { LA cell=new LA(); cell.elt=x; cell.next=tai
l; return cell;}
   static boolean isempty( LA liste) { return liste==null ; }
   static Arc head( LA liste) { return liste.elt; }
   static LA tail(  LA liste) { return liste.next; }
}


public class Dijkstra extends Frame {
        int width, height;
        Pt [] tabpts;
        LA [] tabarcs;
        double [] dist; // distance a la source
        int [] pred; // tableau des predecesseurs dans le PCC de la source au so
mmet

static double square( double d) { return d*d ; }


 // rend la liste des elements de rang pair (le premier est de rang 0 donc pair)
   static public LP pairs( LP l)
   { if (null==l) return null;
     if (LP.isempty( LP.tail( l))) return LP.cons( LP.head( l), null);
     return LP.cons( LP.head( l), pairs( LP.tail( LP.tail( l))));
   }
 // rend la liste des elements de rang impair
   static public LP impairs( LP l)
        { if (LP.isempty(l)) return null; return pairs( LP.tail(l)); }
 // fusion de 2 listes de points triees
   static public LP fusion( LP l1, LP l2)
   { if (LP.isempty( l1)) return l2;
     if (LP.isempty( l2)) return l1;
     Pt e1= LP.head( l1); Pt e2= LP.head( l2);
     if (e1.x <= e2.x || (e1.x==e2.x && e1.y <= e2.y))
```



```
        return LP.cons( e1, fusion( LP.tail( l1), l2));
     return LP.cons( e2, fusion( l1, LP.tail( l2)));
   }
   static public  LP mergesort( LP l)
   { if (LP.isempty( l) || LP.isempty( LP.tail( l))) return l;
     return fusion( mergesort( pairs( l)), mergesort( impairs( l)));
   }

  int etape( LI ls) // on met a jour les voisins du sommet a distance min de la
 source
   {    if( null == ls) return (-1);
        int s= LI.head( ls);
        for( LI li= ls; null != li; li = LI.tail( li))
        {       int i= LI.head( li);
                if ( dist[i] < dist[s]) s=i;
        }
        // ici s est le sommet  le plus proche de la source
        for( LA la= tabarcs[s]; null != la; la= LA.tail( la))
        {
                Arc arc = LA.head( la);
                int v= arc.v; double sv= arc.d;
                if( dist[s] + sv < dist[v]) { dist[v] = dist[s] + sv; pred[v]= s
; }
        }
        return s;
   }
   static LI enlever( LI ls, int s)
   {    LI l= null;
        for( ; null != ls; ls=LI.tail( ls))
                if( LI.head( ls) != s) l= LI.cons( LI.head( ls), l);
        return l;
   }

   void enchainer( LI ls)
        { for(;;)
          {  int s=etape( ls);
             if( -1==s) return;
             ls = enlever( ls, s);
          }
        }
   void PCC( int src) // calcul des + courts chemins
   {
        int n = tabarcs.length;
        dist = new double[n];
        for( int s=0 ; s<n; s++) dist[s]=1e10;
        dist[ src]= 0.;
        pred= new int[n];
        for( int s=0 ; s<n; s++) pred[s]= -1;
        pred[src]= src;
        LI ls = LI.iaj( 0, n-1);
        enchainer( LI.iaj( 0, n-1));
        // ici dist[] et pred[] sont corrects
   }
   void generer( int n, double rayon)
   {
        System.out.println( "ENTREE DANS generer");

        LP l=LP.cons( Pt.point(0.,0.), null);
        for( int i=1 ; i<n;  )
        {       double x=  Math.random();
                double y=  Math.random();
                if ( 0.1 <= x && x <= 0.3 && 0.1 <= y && y <= 0.3) continue;
                if ( square( x - 0.6) + square( y - 0.6) < square (0.25)) contin
ue;
                l= LP.cons( Pt.point( x, y), l); i++ ;
        }
        l= mergesort( l);
        tabpts = new Pt[ n];
        for (int k=0; l != null; l=LP.tail(l), k++) tabpts[k] =  LP.head(l);
```

```
        tabarcs= new LA[ n]; for( int i=0; i<n; i++) tabarcs[i]=null;
        for( int s=0; s<n; s++)
        {       double sx=tabpts[s].x; double sy= tabpts[s].y;
                for( int t=s+1; t<n && tabpts[t].x - tabpts[s].x <= rayon; t++)
                {       double tx= tabpts[t].x; double ty=tabpts[t].y;
                        double st = Math.sqrt( square( tx-sx) + square( ty-sy));
                        if (st < rayon)
                        {       tabarcs[t]= LA.cons( Arc.arc( s, st), tabarcs[t]
);
                                tabarcs[s]= LA.cons( Arc.arc( t, st), tabarcs[s]
);
                        }
                }
        }

        // PCC( (int) (((double) (n-1) ) * Math.random()));
        System.out.println( "CALCUL PCC");
        PCC( 0 );
        System.out.println( "SORTIE DE  generer");
   }

   public void init() {
      System.out.println( "ENTREE DS INIT");
      width = getSize().width;
      height = getSize().height;
      setBackground( Color.white );
      System.out.println( "SORTIE DE INIT");
   }

   void tracer_segment( Graphics g, int i, int j)
   {
        double scale = (double) (getSize().width);
        int xi= (int) (scale * tabpts[i].x); int yi= 10 + (int) (scale * tabpts[
i].y);
        int xj = (int) (scale * tabpts[j].x); int yj = 10 + (int) (scale * tabpt
s[j].y);
        g.drawLine( xi, yi, xj, yj);
   }
   public void paint( Graphics g )
   {

      g.setColor( Color.gray );
      for ( int i = 0; i < tabarcs.length - 1; ++i )
      {
        for( LA l= tabarcs[i]; null != l; l= LA.tail( l))
        { Arc arc= LA.head( l);
          int j = arc.v;
          tracer_segment( g, i, j);
        }
      }

      //g.setColor( Color.black );
      g.setColor( Color.green);
      g.setColor( Color.red);
      for ( int i = 0; i < tabarcs.length - 1; ++i )
      { int j= pred[i]; if (j != -1) tracer_segment( g, i, j); }

   }

public static void main (String arg[]) {
  Dijkstra f = new Dijkstra();
  f.init();
  f.generer( 4000, 0.04);
  f.setBounds(0, 0, 900, 900);
  f.setVisible(true);
 }

}
```