# CS 100: Section Assignment 11

(For the week of April 19)

Section assignments are discussed in section and are not submitted for grading. They relate to recent lecture topics and usually to the current Programming Assignment. Prelim questions are based on Section Assignments, Programming Assignments, and Lecture examples.

**1.** Refer to the class `PointsInPlane`. Modify the `paint` method (in between comments) so that it draws all ten points but just connects the two points that are maximally separated.

**2.** Refer to the class `PointsInPlane`. Modify the `paint` method (in between the comments) so that it draws all ten points and connects each point to its nearest neighbor.

**3.** In the class `PointsInPlane,` implement a method

```
public static double roundtrip(int[][] D, int[] path)
// D is an n-by-n pairwise distance array and path is a length n itinerary array.
// Yields the roundtrip distance of a trip defined by path.
```

**4.** Given that the method `roundtrip` has been implemented, modify the `paint` method (in between the comments) so that it displays the shortest of the round trips defined by `findPath(D,0)`, `findPath(D,1)`,…,`findPath(D,9).`

```
public class PointsInPlane extends Frame
{
    public void paint(Graphics g)
    // Draws an approximately shortest round trip path through 10 points.
    {
           g.fillRect(0,0,1000,1000);
           int[] h0 = { 100, 450, 200, 300, 400, 650, 500, 850, 150, 150};
           int[] v0 = { 400, 100, 250, 500, 350, 250, 100, 300, 200, 400};
           double[][] D = distTable(h0,v0);

       // ***************************************************************
           int[] s = findPath(D,0);
           showPath(g,h0,v0,s);
           showCities(g,h0,v0);
       // ***************************************************************
     }

    public static double[][] distTable(int[] h, int[] v)
    // Yields an n-by-n “pairwise distance array” defined by points (h[i],v[i]), i=0,…,n-1
    // The (i,j) entry is the distance from a city at (h[i],v[i]) to
    // a city at (h[j],v[j]). Here n = h.length = v.length.
    {
       int n = h.length;
       double[][] D = new double[n][n];
       for(int i=0;i<n;i++)
       // Record all the distances from the ith point. Exploit symmetry.
       {
          D[i][i] = 0;
          for(int j=0;j<i;j++)
          {
             D[i][j] = Math.sqrt((h[i]-h[j])*(h[i]-h[j]) + (v[i]-v[j])*(v[i]-v[j]));
             D[j][i] = D[i][j];
          }
       }
       return D;
    }
```

```
public static int[] findPath(double[][] D, int firstStop)
 // Assume D is n-by-n array whose (i,j) entry represents the distance between
 // city i and city j. firstStop must satisfy 0<=firstStop<n.
 // Yields a length n array whose values are a permutation of 0,1,...,n-1.
 // The ith component of the array represents the ith stop
 // on a trip that starts at city firstStop and visits each city exactly once.
 // The itinerary returned approximates, while not of guaranteed minimal length, tends to
 // be close to optimal in that regard.
 {
    int n = D.length;

    // A boolean array that keeps track of the cities visited...
    boolean[] beenThere = new boolean[n];
    for(int i=0;i<n;i++)
       beenThere[i] = false;
    beenThere[firstStop] = true;

    // Get ready for the trip...
    int[] city = new int[n];       // city[i] will be the index of the ith city visited.
    int c   = firstStop;           // c is the index of the current city.
    city[0] = c;

    int k=0;
    int nextStop;
    double minDist;

    for(int i=1;i<=n-1;i++)
    // Find the i-th city to visit
    {
       // Find the first unvisited city.
       k=0;
       while(beenThere[k])
          k++;

       minDist = D[c][k];
       nextStop = k;

       // Now try to find a closer unvisited city.
       while(k<n)
       {
          if(!beenThere[k] && D[c][k]<minDist)
          {
             minDist = D[c][k];
             nextStop = k;
          }
          k++;
       }
       // nextStop = index of the closest unvisited city. Go there.

       c = nextStop;
       city[i] = c;
       beenThere[c] = true;
    }
    return city;
 }
```

```
    public static void showPath(Graphics g, int[] h, int[] v, int p[])
    // Let n be the length of h,v, and p. Assume p is a permutation of the integers 0,1,…,n-1
    // Connects (h[0],[v[0]),…,(h[n-1],v[n-1]) in the order
    // specified by p (with return to the starting point), i.e.,
    //
    //  (h[p[0]],v[p[0]]), (h[p[1]],v[p[1]]),..., (h[p[n-1]],v[p[n-1]]), (h[p[0]],v[p[0]])
    {
       g.setColor(Color.red);
       int a,b;
       int n = h.length;

       for (int i=0;i<n;i++)
       {
          a = p[i];
          b = p[(i+1)%n];
          g.drawLine(h[a],v[a],h[b],v[b]);
       }
    }

    public static void showCities(Graphics g, int[] h, int[] v)
    // Displays the points (h[0],v[0]),...,(h[n-1],v[n-1]) where n = h.length = v.length
    {
       int n = h.length;
       int r = 8;
       g.setFont(new Font("TimesRoman",Font.BOLD,14));
       for(int i=0;i<n;i++)
       {
          g.setColor(Color.yellow);
          g.fillOval(h[i]-r,v[i]-r,2*r,2*r);
          g.setColor(Color.black);
          g.drawString(String.valueOf(i),h[i]-r/2,v[i]+r/2);
       }
    }
}
```