CSCI 3412 – Algorithms

Jeremy Diamond

Problem Set 4

Part I – Problem Statement

Given a list of all stars within 10 parsecs of our sun (herby called Sol) find a path of low distance to visit all of them. The solution must be some type of greedy algorithm as this problem (The Traveling Salesman Problem) has no known solution that can be run in a reasonable time.

Part 2 – Design

The suggested heuristic is to, at each step, go to the nearest unvisited star. This can be implemented in $O(n^2)$. Notably though this is not the case for the above dataset it has been proven that this method can result in worst possible distance (https://www.sciencedirect.com/science/article/pii/S0166218X01001950).

Part 3 – Implementation

My implementation was in java and relies on, an adt called Star, and the internal functions of java. In java I found it easiest to simply read in the csv line by line int a string. I split that string along the ',' characters and convert the needed numbers into doubles. My Star adt has 6 data members. 2 strings for name and number of the star and 4 doubles for distance and xyz location. There are 2 appropriate constructors and a function for calculating new distances at each step. Each line in the csv is read into a new Star. In java many data structures are put in the general category of "Collections" this includes the ArrayList class. All Collections have an internal function called a "Comparator" which returns 1, 0, or -1 for a given comparison of 2 objects. For any "Collection" the internal sort function relies on this "Comparator". I overload this class and function to make use of the internal sort for an ArrayList of Star objects by distance.

With the Above functionality built my main places all stars in an ArrayList called baseStars. At each step the top star on the sorted list is places in a pointer called CWS (currently working star) and the previous CWS is placed on a queue. Then each star's distance from CWS is calculated and the list is resorted. Following the termination of the main loop I then output the contents of the queue summing he recorded distances.

Part 4 – Results

All 327 stars are traveled too in 554.9656377679933 parsecs. This is surprisingly good for such a simple heuristic however better heuristics for the problem are know too exist and it is an active area of research. I would not be surprised if there is a great deal more optimization to be done.

Part 5 – Conclusions

This project forced me to learn new things about greedy algorithms, java as a language, and perspective on problem solving. It's been a great semester, see you in "Principals of Programming" in the fall.

Part 6 – Code

- Star Class

```
public class Star {
   // data
    public String number = ""; // used fo string.isEmpty
    public String name = "";
    public double distance;
    public double xPosition;
    public double yPosition;
    public double zPosition;

    //constructors

   public Star (String thenum, String thename, double dist, double x, double y,double z)
   {
      number = thenum;
      name = thename;
      distance = dist;
      xPosition = x;
      yPosition = y;
      zPosition = z;
   }

   public Star (String thenum, double dist, double x, double y,double z)
   {
      number = thenum;
      distance = dist;
      xPosition = x;
      yPosition = y;
      zPosition = z;
   }

   public void distanceCalc (Star CWS) //implimtation of the 3d distance formula
   {//CWS = currently working star
      double diffX = (this.xPosition)-(CWS.xPosition);//abs value not needed because they will be squared
      double diffY = (this.yPosition)-(CWS.yPosition);
      double diffZ = (this.zPosition)-(CWS.zPosition);

      diffX = diffX*diffX;
      diffY = diffY*diffY;
      diffZ = diffZ*diffZ;

      double squSum = diffX+ diffY + diffZ;

      this.distance = Math.sqrt(squSum);
   }


}
```

- SortByDistance Comparator

```
class SortByDistance implements Comparator<Star> //this class allows one to use native java sort on ADT star
{

   public int compare(Star a, Star b)
   {

      if(a.distance > b.distance){
         return 1;
      }

      else if (a.distance < b.distance)
      {
         return -1;
      }

      else{
         return 0;
      }
   }

}
```

- CSV Parser function

```
//based on https://www.mkyong.com/java/how-to-read-and-parse-csv-file-in-java/
   public static void parser(ArrayList<Star> theStars)
   {
      String csvFile = "hygxyz.csv";
      BufferedReader br = null;
      String line = "";
      String cvsSplitBy = ",";


      String name;
      String number;
      double distFloat = 0;
      double xFloat = 0;
      double yFloat = 0;
      double zFloat = 0;

      Star tempStar;
      SortByDistance readin = new SortByDistance();

      int count = 0;
      ArrayList testints = new ArrayList();
      testints.add(1);
      testints.add(50);
      testints.add(6);



      try {

         br = new BufferedReader(new FileReader(csvFile));
         line = br.readLine();
         while ((line = br.readLine()) != null) {

            // use comma as separator
            String[] nextStar = line.split(cvsSplitBy);

           // System.out.println("Star ID : "+ nextStar[0]+" distance :" + nextStar[9]);

            distFloat = Float.parseFloat(nextStar[9]);
            if (distFloat <= 10)
            {
              //System.out.println("Star ID : "+ nextStar[0]+" distance :" + nextStar[9]+ " xpos : "+ nextStar[17] );


              xFloat = Float.parseFloat(nextStar[17]);
              yFloat = Float.parseFloat(nextStar[18]);
              zFloat = Float.parseFloat(nextStar[19]);

              if (nextStar[6].isEmpty()){
                 tempStar = new Star(nextStar[0],distFloat,xFloat,yFloat,zFloat);
              }
```

```
            else{
               tempStar = new Star(nextStar[0],nextStar[6],distFloat,xFloat,yFloat,zFloat);
            }

            theStars.add(tempStar);
         }

      }

      Collections.sort(theStars, new SortByDistance());

   } catch (FileNotFoundException e) {
      e.printStackTrace();
   } catch (IOException e) {
      e.printStackTrace();
   } finally {
      if (br != null) {
         try {
            br.close();
         } catch (IOException e) {
            e.printStackTrace();
         }
      }
   }
}
```

- Main Function

```
public class HW4 {

  /**
   * @param args the command line arguments
   */
  public static void main(String[] args) {
    // TODO code application logic here
    ArrayList<Star> baseStars = new ArrayList<Star>();
    parser(baseStars);

    Star CWS; //currently working star
    Queue<Star> order = new LinkedList<>(); //needed data for preducing a queue

    order.add(baseStars.get(0)); //block adds sol to order and sets up next nearest star as CWS
    baseStars.remove(0);
    order.add(baseStars.get(0));
    CWS = baseStars.get(0);
    baseStars.remove(0);
    double distanceTotal = 0;
    int size = baseStars.size(); //needs to be fixed for this to work

    for (int j = 0; j < size; j++) //places all stars into order and claculates there distances O(n^2)
    {
      for (int i = 0; i < baseStars.size(); i++)
      {
        baseStars.get(i).distanceCalc(CWS);

      }
      Collections.sort(baseStars, new SortByDistance());
      order.add(baseStars.get(0));
      CWS = baseStars.get(0);
      baseStars.remove(0);
    }

    while (!(order.isEmpty())) //prints results. is distructive
    {
      distanceTotal += order.peek().distance;
      if (order.peek().name.isEmpty())
      {
        System.out.println("Number of Star: "  + order.peek().number + " Diastance added: " +
order.peek().distance );
      }
      else
      {
        System.out.println("Name of Star: "  + order.peek().name + " Diastance added: " +
order.peek().distance );
      }
      order.remove();
    }
    System.out.println("Diastance of path: " + distanceTotal);
```

```
    }
}
```