**CODE ADDED**

**Class Polyline**

*Method size*

*Method generalise*

*Method furthersFromSeg*

*Method split*

*Method combinePolyline*

*Method getStartEndSeg*

*Method getStart*

*Method getEnd*

**Class Segment**

*Method segAsPolyline*

*Method pointSegDist*

**Class Dpoint**

*Method setD*

*Method getD*

*Method setI*

*Method getI*

**COMMENTS**

**Class Polyline**

*Method size*

*This method gives the size of the polyline object, how many points in it*

*Method generalize*

It applies the Douglas-Peuker Line Generalization.

First, this method checks whether the polyline has only two points in it. In this case, there is no need to proceed further and the function returns the polyline itself, passing it back to **pp.plotPolylines** in the **Lecture2d.py**.

If there are more than 2 points, the algorithm is calling the method ***furthersFromSeg*** to find the furthest point.

**IF** the furthest point is within the threshold distance (t) we call the method ***getStartEndSeg*** which will return a segment. After we return it as polyline calling the method **segAsPolyline**.

**ELSE** the point is outside of the range, then we split the polyline in two. On this two new polylines, C1 and C2we call the method generalize recursively. The recursion will branch as many times as needed until reaching a cycle in which the condition *“furthest point within the threshold distance (t)”* is respected. All the generalized methods that have been initiated recursively till there will start returning back combined polylines **“return (self.combinePolyline(c1,c2))”**.

When all the recursion branches have been followed backward, the original generalized method will return a polyline combinations of all the tiny polylines under the threshold that the recursion generated.

*Method furthersFromSeg*

*Calls the method* ***getStartEndSeg*** *which gives back a segment between the start and end point. It loops after over all the points of the polyline storing the furthest from the segment, usingthe method* ***pointSegDist****. It returns back a* ***Dpoint*** *element which has as attributes: x,y, max value, and index.*

*Method split*

*It splits a polyline according to the argument “index”. It returns two polylines.*

*Method combinePolyline*

*It combines two polylines in one. It does not record the junction point twice.*

*Method getStartEndSeg*

*Calls the methods* ***getStart*** *and* ***getEnd*** *to generate a segment between the start and the end point.*

*Method getStart*

*Get the start point.*

*Method getEnd*

*Get the end point.*

**Class Segment**

*Method segAsPolyline*

*Store the start and the end points of a segment in a list. The list is after converted into a polyline object.*

*Method pointSegDist*

*Finds the distance between a point and a segment.*

**Class Dpoint**

*Method setD*

*Set the distance attribute*

*Method getD*

*Return the distance attribute*

*Method setI*

*Set the index attribute*

*Method getI*

*Return the index attribute*

The code I uploaded is functioning, and the lines I added have comments. However, despite the fact that I managed to understand what all the little bits of the code are doing I am missing a deep comprehension of the macro-structure. Particularly it is still a bit obscure the nature of the Segment class. While it is very clear what a polyline or a point 2D are doing the deep purpose of the segment class remains unclear. From the polyline class we move to the segment class for certain operations such as get the distance point-segment but after we go back recreating a polyline object, (N.B. the polyline is what generalize returns). Therefore, I see the segment class as an intermediate step and I'm not sure if we could have avoided it.

Questions:

Was it necessary to use the Segment class?

How can I best decide in which classes write the methods when engineering complex code?

For example, **pointSegDist** instead of being in the Segment class could it be a method in the point2D class? Are there any drawbacks to doing so?