Mandatory Project-1

Manoj Kumar Ellanti

netId:mxe150630

1. CircularList.Java contains code for the implementation of CircularList and Iterator in it. Header is null in this list.
2. Graph is read in the Driver main method using Graph.readGraph().
3. Driver contains a method breakGraphIntoTours(Graph g) which returns a list of circularLists of all the tours made.

**Loop Invariants:**

Initialization:

1. Initially Exception is false, since we start at 0 visited vertices , k=0 visited edges.

Maintenance:

1. All the vertices seen are added in discovered arraylist and seen flag of vertex is marked as true.
2. An Iterator Iterarr[ ] is created to store all the iterators of the Vertices.
3. Initially the start vertex is first Vertex of graph.
4. The corresponding vertex position in the circular List is marked in HashmapdiscoveredPosition for Easier Stitching phase.
5. If an iterator exists for it we retrieve the currentEdge using .next(), else we create one store it in the Iterarr[ ] and traverse till we get an unvisited edge.
6. If all edges are visited we go for start =next Vertex in discovered arraylist.
7. Else we make this edge visited and increment the k, the other vertex is taken and repeated with the steps 5,6,7,8 until the other edge is start.
8. At any point in loop exception means that an edge does not exist unvisited or a cycle does not exist either case it gives an error[no Euler path found].
9. If a circular List is found it is added to the final List.

Termination:

1. Whenever k is Edgecount or an exception arises loop ends.

4. stitchTours(List<CircularList <Vertex>>) and stitches all the vertices according to their position in the Map.

**Loop Invariants:**

Initialization:

1. Initially cl1 contains the first tour and the k is the current tour stitched , since k=1 only one tour is stitched as of now.

Maintenance:

1. Cl2 is the Tour that is currently stitched, the discoveredPositionMap gives position of cl2.header.next element.
2. Corresponding Swapping is done using a dummy Entry object to merge the lists.
3. K is incremented to continue the loop.

Termination:

1. Loop is terminated when all the tours are Stitched k==tourslength .

Returns a CircularList<Vertex> {all the tours in an order}.

5. verifyTour(Graph g, CircularList<Vertex> tour) method returns a boolean value whether tour is a valid path.

Edge class is added with attributes Keytomap, visitedverification, inMap, KeytoMap is given value from+” ”+to to make a unique key.

inMap =false initially, VisitedVerification is also false.

All the edges are mapped in to verifyMap with their KeytoMap as Key and their Edges as value. Edge.inMap is made true.

For the input List<CircularList> for every vertex Start and Start.next we see if there exists an edge in verifymap which is visitedVerificaion is false.

**Loop Invariant:**

Initially s=start.element+ “ “+start.next.element, s1=start.element+ “ “+start.next.element.

Maintenance:

T=verifyMap.get(s)||verifyMap.get(s1).

If T is null no such edge in Map=> no edge in graph return false.

Else

* T.visitedverified is true, Edge is already verified => repetition of edges return false.
* T.visitedverified is false, T.visitedverified =true.

Start=start.next, k++ for loop continuation.

Termination:

Whenever k>=toursize-1. All the vertices are verified in the tour for corresponding edges in the graph.

Return true.

6. Driver.java contains all the methods to read the input, to calculate sub tours if there exists any and to stitch and verify them.