COL106
Assignment 6

ROHIT AGARWAL
2018EE10494

I have maintained RBTrees storing all the edges, points and added triangles.
And a linkedList of components. An int for storing a sno.

Component class:
Stores an int as id for identification. And a linkedList for triangles contained in it. Has a getPoint method which returns linkedList of all the points in that component by doing a BFS on triangles in that component. Similar to as done in getCentroid() (explained later). O(t + t*(n)) where t is no. of triangles in that component and n is average no. of neighbouring triangles a triangle has. (Big O of BFS Explained Later).

Edge object stores its length, its points and all the triangles connected to it in a LinkedList,

Point Object stores all the Points, Edges and Triangles connected to it in LinkedLists, its x,y,z and a boolean value visited to be used in BFS.

Triangle class stores its points and edges, all neighbouring triangles in a linked list, a int sno for arrival order, a boolean visited and an int bfs_no for use in BFS, a reference connectedTo to another triangle it is connected to. It has a isConnectedTo() method which returns the triangle itself if connectedTo is null else returns connectedTo.isConnectedTo(). It also stores reference to a component. But anytime component of a triangle is needed component of this triangle.connectedTo() is used.

ADDTRIANGLE:
Takes input and forms point and edge objects. Check triangle validity. If not valid then return false. Searches for existing edges and points in their RBTrees (O(log(e) + log(p)) where e and p are no. of edges and points) and if found, replace these newly formed Objects by the stored ones, else add them in the RBTree (O(log(e) + log(p)) where e and p are no. of edges and points). Form the triangle with sno++ and then go its edges and for each connected triangle for each edge, add this triangle to their triangle linkedlist and vice versa. (O(n) where n is no. of connected triangles) This triangle is then added to the triangles RBTree.(O(log(t)) where t is no. of triangles) This triangle is then added to triangle list of its points and edges and true is returned. (O(contant) for this part.)

For maitaing components:
In triangle constructor a new Component() is formed with id = sno and that triangle is added to its triangle linkedList. Now when ADDTRIANGLE is run, when triangle is made its component is

added to components linkedList. When adding neighbours to triangles, a merge funtion is called for each edge. What it does for each edge is that it calls isConnectedTo() for new triangle and one of its connected triangle to get triangles t1 and t2 and get their components. If components are different then appends the triangle linked list of smaller component to the triangle linkedList of bigger component. And sets the the connectedTo value of t1 or t2, whichever belongs to smaller component, as the isConnectedTo() of other one. Then it deleted the smaller component from components linkedlist. This is like DSU.
O(constant).

OVER ALL TIME COMPLEXITY OF ADD_TRIANGLE = O(log(e) + log(p) + log(t) + n) where e, p , t are no. of edges, points and triangles respectively, n is no. of connected triangles through the edges.

Explaining BFS:
First go through the all the triangles in the component and mark them unvisited (O(t)). Mark the first triangle visited and add it to a queue (Made by LinkedList). Now while queue is not empty: Remove triangle from queue, do whatever operation is required with the triangle. Now go through its neighbouring triangle linkedList, and mark each unvisited triangle visited and add it to the queue. O(t + t*(z +n)) where t is no. of triangles in that component and n is average no. of neighbouring triangles a triangle has. The z comes from the complexity of the operation performed on each triangle which depends on for what purpose BFS is done.

TYPEMESH:
Go through RBTree of edges and see size of Triangles LinkedList stored in them. If any size is >2, return 3. If all 2, return 1, If some are 1, some are 2, return 2. Time Complexity - O(t) where t is the number of edges.

BOUNDARYEDGES:
Go through RBTree of edges and see size of Triangles LinkedList stored in them. If any size is 1, add that edge to a new LinkedList, finally make an array from the list and then apply mergeSort according to length on it and return it. O(e + e*log(e)) where e = no. of edges.

COUNT_CONNECTED_COMPONENTS:
Return size of components linkedlist. O(1)

NEIGHBOURS_OF_TRIANGLES:
Searches the RBTree of triangle to get it. Makes an array from the triangles stored in the neighbouring triangles linkedList in the triangle. This is already sorted by arrival order. O(log(t) + n), where t is total no. of triangles and n is the no. of neighbouring triangles.

EDGE_NEIGHBOUR_TRIANGLE:
Searches the RBTree of triangle to get it. Makes an array out of the edges stored in it. Returns it. O(log(t)) where t is total no. of triangles.

VERTEX_NEIGHBOR_TRIANGLE:
Searches the RBTree of triangle to get it. Makes an array out of the points stored in it. Returns it. O(log(t)) where t is total no. of triangles.

EXTENDED_NEIGHBOUR_TRIANGLE:
Searches the RBTree of triangle to get it. For each point, go through its triangle linked list and add triangles (which is not equal to our triangle) to a new RBTree. This eliminates duplicates. Then goes through Tree to add them in an array. Sort this array according to arrival order through merge sort. O(log(t) + n*log(n) + n + n*log(n)) where t is total number of triangles and t is the number of triangles connected to the three points.

INCIDENT_TRIANGLES:
Searches Point in the RBTree. Makes array out of triangles stored in triangles linkedlist of the point. This is already sorted. Return it. O(log(p) + n) where p is total no. of points and n is the no. of connected triangles.

NEIGHBORS_OF_POINT
Searches Point in the RBTree. Makes array out of triangles stored in points linkedlist of the point. Return it. O(log(p) + n) where p is total no. of points and n is the no. of connected points.

EDGE_NEIGHBORS_OF_POINT
Searches Point in the RBTree. Makes array out of triangles stored in edges linkedlist of the point. Return it. O(log(p) + n) where p is total no. of points and n is the no. of connected edges.

FACE_NEIGHBORS_OF_POINT
Same as INCIDENT_TRIANGLES.

IS_CONNECTED:
Searches Triangles in RBTree and then compares their component ids. If same then return true, else false. O(log(t)) where t is total no. of triangles

TRIANGLE_NEIGHBOR_OF_EDGE:
Searches Edge in the RBTree. Makes array out of triangles stored in triangles linkedlist of the edge. This is already sorted. Return it. O(log(p) + n) where p is total no. of edges and n is the no. of connected triangles.

MAXIMUM DIAMETER:
Goes through component LinkedList and checks their size to get the component with max no. of triangles.
Now apply BFS for all the triangles of that component. When marking all triangles as unvisited as set bfs_no of each triangle as 0. Maintain an int i. For each triangle removed from queue put i

= bfs_no stored in triangle. For each neighbouring triangle that is added in queue, set their bfs_no as i+1.
At end of each BFS, check the final i and store the max one. Return it.

Return the diameter found. O(c + t(t+t*(n)) where c in no. of components and t is no. of triangles in biggest component and n is no. of neighbouring triangles of each triangle.

CENTROID:
Calls getCentroid for each component and forms an array of returned values. Then applies mergeSort on them and return them. O(c*(t+t*n) + c*log(c)) where c is no of components and t is no. of triangles in each component and n is no. of neighbouring triangles

CENTROID_OF_COMPONENT:
Search point in RBtree and then get a component from any triangle it is connected to. Pass this to getCentroid(). O (log(P) + t+t*(n)) where t is no. of triangles in the component and n is no. of neighbouring triangles of each triangle and P is no. of total points.

getCentroid():
Maintains three floats x, y, z. And a points_counter.
Takes a component object as parameter. Applies BFS on the triangle graph in it. Here points are also marked as visited or not. For each unvisited point of each unvisited triangles add its x,y,z values to the maintained total x, y, z. And increase point counter.
At end divide x,y,z by counter and return them in an array.
O(t+t*(n)) where t is no. of triangles in the component and n is no. of neighbouring triangles of each triangle.

CLOSEST_COMPONENTS:
Make an array of LinkedLists storing points. Call getPoints for all the components and stores points in that array of LinkedLists.
For each point in each LinkedList go through each point in other LinkedLists and compare distance between points and get the min dist and the corresponding points. Return the points in an array. O(c*(t+t*(n) + p(1)*p(2) + p(1)*p(3) + …. + p(c-1)*p(c)) where c is no. of components, t is no. of triangles in each component and n is no. of neighbouring triangles. p(i) is no. of points in ith component.