ASSIGNMENT 6

For this assignmnent I maintained graph of different things by using hashtable.

CLASSES INCLUDED:-

arraylist

DHNode

Graph_node

Hashtable

Edge

Pair

Point

Queue

Shape

Triangle

arraylist :- Implemented own arraylist using array and when added more then current then copied the initial array to new array of 2*capacity. It have normal functions .

DHNode:- It is node for the Hashtable.

Graph_node:- This node is actually a triplet and is generic type.

Functions : 1. hashCode :- Overrided the hashCode() method and implemented own.

2. equals: Overrided the equals function as recquired .

Hashtable:-

Index arraylist is maintained to store the indexes where key is hashed finally in hashtable so iteration will become easier.

Hashcode used for hashing :- Here in the assignment we have cordinates as a key, so I have generated

the hash by this :- ((31)^2)(x.hashCode())+(31) (y.hashCode())+ (z.hashCode()) . As float have its inbuilt hashCode method . 31 is a prime number so this is a type of poylnomial prime hashing to minimise the collisions.

Functions:- 1.insert :-Trivial insert as done in previous assignment. The new things is if load factor become more the 0.5 then I have to rehash.

Rehash():- Tablesize is doubled then old data is copied to new double sized array and now my new array will become the main array(I.e. changed the references).

Contains and get function are trivial .

Queue:- Trivial implementation of queue using linked type structure.

Edge : - this class stores all the information about an edge .

There is arraylist of triangle present in edge which is the list of triangles which have common edge.

Functions:-

addtriangle :- This adds a given triangle with common edge to triangles arraylist , this also updates the triangles neighbours list.

Edge length:- this returns the edge length .

Check :- this returns the dot product of two points of an edge.

Mesh :- this gives the triangles list size.

Others are trivial .

Pair :- Same as previous assignment . TRIVIAL

Point :- This stores the information about the points .

It have three lists :- Its neighbouring points ,incident triangles ,edges .

Functions :

Addtriangle :- This adds a given triangle to the list of triangles . This also updates the extended neighbour list of triangles.

CompareTo :- this is to compare two points .

Boolean Equal(x,y):- this check whether two points are equal or not . As given in specification there can be error of 0.001f , so if difference between two points are less than that then return true else return false.

Hashcode :- Same as described in hashtable .

Triangle :- This have basic information and two arraylists neighbours and extended neighbours ,these are maintained as said earlier.

Functions:- boundary_check :- Check if triangle is boundary triangle.

Others are trivial .

SHAPE :-

MAINTAINED DATA STRUCUTES :-

List of triangles

List of edges

List of points

Hashtable of points

Hashtable of triangles

FUNCTIONS :-

1.check_collinear :- it takes three edges and check whether three points are collinear or not .

2.mergesort:- Different functions for different Use of sorting.

3. Distance :- Distance between two points.

4.ADD_TRIANGLE :-

All the pre storing before queries happens in this function . Collinearity is checked in this function.All the data structures are updated in this function.

5.MESH:- MESH TYPE is returned , I have maintained a mesh_type gloabally and I am computing mesh type in ADD_TRIANGLE.Logic is that when an triangle is added and all the things are updated then we check the mesh of three edges and if one of them is greater then is 2 then mesh_type is set to –1 and returned. If not then we take product of mesh of all edges ,then checks for different cases, as an edge have mesh 1 or 2 because when an edge have mesh more than 2 then mesh_type is set to -1. So finally we return the mesh_type by checking appropriately whether it is positive negative or zero.

6.INCIDENT TRIANGLES of point :- TRIVIAL I have list of triangles stored in a point.

SIMILARLY as above for edges neighbours of point,neighbours of point,face neighbours of point.

7.BOUNDARY_EDGES:- a edge is boundary if its triangles list size is 1 . AS recquired SORTING IS DONE.

8. TRIANGLE_NEIGHBOR_OF_EDGE:- BY using point data structure , I get the edge then triangles list of edge is answer.

9. NEIGHBORS_OF_TRIANGLE:- returned the list stored in that triangle.

10.EDGE_NEIGHBOUR_TRIANGLE :- returned the edges of the triangle if exists.

11. VERTEX_NEIGHBOR_TRIANGLE:- returned the vertex of triangle if it exists.

12. EXTENDED_NEIGHBOR_TRIANGLE:- If triangle exists then returned it extended neighbour list.

13. COUNT_CONNECTED_COMPONENTS :- Did a depth first search on all triangles list.

14. IS_CONNECTED:- ALSO did DFS for one of triangle and search the other triangle in its neighbours list.

15.CENTROID_OF_COMPONENT:- DID THE DFS TYPE serching on that point and then visited the neighbours and simultaneously changing the centroid also.

16.CENTROID:- SIMILART TO CENTROID, but here iterated over all the points and calculated centroid.Then finally sorted the centroids in the given order .

17.MAX_DIAMETER:- Iterated over all the triangles and founded the diameter of components ,Diameter of component is founded by using Breadth First search(BFS) on that triangle . FOR bfs i have used a queue and did a normal BFS. I have maintained a hop in each triangle so logic is that when going from one triangle to its neighbour then the hop of that neighbour is increased by (hop of parent triangle+1).And also maintained the number of triangles in a component.

18.CLOSEST_COMPONENTS :- firstly I have found all the points in a component for that I have maitaned arraylist of arraylist and used dfs to do so. Then I have to just iterate over this list and check the distance between two points in different components

TIME COMPLEXITY:-

V=number of vertices in the graph like vertex=triangle,point etc.

E=number of edges in the graph.

m=number of things(like triangles etc.) in answer query as my implementation returns arraylist of “things” in O(1) but intereface recquires array so copying takes O(m)time.

1. MESH:- O(1)

2. INCIDENT TRIANGLES of point:- O(1 *m)

3. neighbours of point,neighbours of point,face neighbours of point takes same O(1*m).

4. BOUNDARY_EDGES:- O(E+m+mlogm) mlogm is for mergesort.

5. TRIANGLE_NEIGHBOR_OF_EDGE :- O(1*g+k) g=number of comparisons recquired to get edge from one point to other in one of the points edgelist. k= number of triangles in that edge.

6. . NEIGHBORS_OF_TRIANGLE:- O(1*g) g= number of neighbours of triangle given.

7. . EDGE_NEIGHBOUR_TRIANGLE:- O(1)

8.. VERTEX_NEIGHBOR_TRIANGLE :- O(1)

9. . EXTENDED_NEIGHBOR_TRIANGLE:- O(1*k) k= number of extended neigbours of that triangle.

10. COUNT_CONNECTED_COMPONENTS:- O(n*(V+E) +k) n=number of triangles, V=vertices of graph and

E=edges of graph , actually it takes O(n) time to go to each triangle and in that we do DFS on that triangle’s neighbours so time taken is O(number of connected neighbours of that triangle). And k= for unmarking the number of visited triangles.

11. . IS_CONNECTED :- O(1* (V+E)) +k) k is same as above.

12. CENTROID_OF_COMPONENT:- O(1* (V+E)+k)k same as above.

13.CENTROID:- O(j*(V+E)+k+m)k =same as above, j=number of points

14. MAX_DIAMETER : O((t *(V+E)+k)   t=number of triangles, k= same as above.

15.CLOSEST_COMPONENTS :- O(j* O(V+E)+ number of components *number of points in that component) so O(n^2).

16.ADD_TRIANGLE:- O(n) where n is number of operations in total overall taking linear time complexity.