COL106 Assignment6
Dipen Kumar (2018CS50098)

No imports used here. I have used self-designed Arraylist, Trie and MaxHeap.

Arraynode.java- This class create a node which store a generic type value. It has a method called value() which return the generic value. This class is created in order to help me create Arraylist where I needed to create an array of generic value T which is not possible hence created array of Arraynode which further stored generic value T. As clear from above statement I have used array to create Arraylist.

Arraylist.java- This class has a field of type array<Arraynode<T>> and also an integer field named size which stores the number of elements stored in array. It has five methods-

1. Size()- return size. O(1)
2. Get(int i)- return the element at i^{th} index in array. O(1)
3. Add(T value)- add the element at index number size as this is the immediate next position in array. If the array is full then increase its size twice and then add. When it add it also increases the size of the Arraylist by 1. O(1);
4. Remove(int 0)- return the element at i^{th} index and replace its value with $(i+1)^{th}$ element and $(i+1)^{th}$ with $(i+2)^{th}$ and so on. It assign last index i.e. size-1 with null. Finally decrease the size by 1. O(size)
5. Set(int i,T value)- assign i^{th} element with value=value. O(1)

TrieNode.java- I have used the same trie data structure as given in assignment4 and assignment5 with slight changes. First was the length of array here is 13 and second I have used only insert and search method. This is the node which store value and array of TrieNode of length 13.

Trie.java- It has two methods-

1. Insert(String word,T value)- It goes to word.charAt(i)-'-' index of array and its array and its array till the word continue and add the value to that node. O(n) where n is the length of the word.
2. Search(String word)- It follows the above mentioned approach and return the value at that node. O(n) where n is the length of the word.

Node.java- This is used for MaxHeap. This already created for assignment4 and assignment5. This class stores generic value and a integer value Time which will break the conflict when priority of the generic value is same giving preference for first in to first out.

MaxHeap.java- Two methods described below in brief-

1. Insert(T element)- Creates a node and fill the field of type T with element and add to the last of the Arraylist used for maintaining maxheap. Then it bubble it up governed by compareTo method comparison. Tie is broken by time. O(logn) where n is the input size in the Arraylist.
2. ExtractMax()- It returns the value at index numbered 0 and then fill this with index numbered size-1 and then bubble it down to its required place. O(logn) n is the input size in the Arraylist.

Point.java- Stores an array of float of size 3, x, y and z. three arraylist of type point, edge and triangle which stores the point neighbor, edge neighbor and triangle neighbor of the given point respectively. An index number to refer its location in global arraylist of point in shape.java for searching this point in arraylist with O(1). It has compare to method which prioritize the point first with x coordinate then with y coordinate and finally with z coordinate. Another extra method I have used here is getKey(). This returns a string hashed from the x y z coordinates of the point.

Edge.java- Stores an array of Point of size 2. Float named length_sq which stores the square of the length of the edge. Arraylist of triangle which stores the triangle neighbor of the edge. compareTo method return 1 if length is lesser else -1 if greater else 0. getKey() here also return a string that is the name of the edge used for inserting and searching in trie.

Triangle.java- Three points stored in array. Integer filed named time stores the time of the arrival of triangle which will help to sort the triangles with respect to their arrival time. Arraylist of point, edge and triangle, triangle will store the list of point neighbor, edge neighbor, extended neighbor and triangle neighbor respectively of the triangle. Integer index will store the address of the triangle in the arraylist of entire triangle created in the shape.java. This make searching this particular triangle in the arraylist in O(1). compareTo compare on the basis of the time of the arrival. getKey() returns a string representing unique name to the triangle.

Shape.java- Data Structures used here are-
1. Trie1 will store all points in the graph in a Trie
2. Trie2 will store all edges in the graph in a Trie
3. Trie3 will store all triangle in the graph in a Trie
4. Alist1 will store all points in the graph in a Arraylist
5. Alist2 will store all edges in the graph in a Arraylist
6. Alist3 will store all triangles in the graph in a Arraylist

18 methods to be implemented are described below with their runtime complexities-
1. add_triangle(float[] triangle_coord)- This creates three points a, b, c. if they are already present in trie1 then these references will point to the previously existed objected created earlier else we will add these points in alist1 and trie1. Create three edges with these points. Let edges be ab, bc, ca and check

it in trie2 if exists then update these references to previously created object else add in trie2 and alist2. Also if an edge was not present earlier that means those two points were not neighbor of each other so update their neighbor list of point and edge. Create a triangle with these three points and if triangle was present in trie3 then return false because duplicate entry has come else add that triangle to alist3 and trie3. Also update the triangle point neighbor with its vertices and edge neighbor with its edges. In order to fill its extended neighbor fill the list with all the triangle in list of all its vertices. Then add this newly created triangle in the list of its vertices to avoid making one neighbor to oneself. In order to sort these triangle in order of their arrival time I have used maxheap. First insert all then extract all with 2nlogn steps giving time complexity O(nlogn). Same logic was applied to create arraylist of triangle neighbor of a triangle with difference that now we will fetch triangles from triangle neighbor of all edges of the triangle. Then add this new triangle to all triangle neighbor of its edges to avoid duplicacy. Sort with the help of MaxHeap. Also neighbor is symmetric relation i.e. if t1 is neighbor of t2 then t2 is neighbor of t1. This is taken care of. Complexity: searching in trie is order length of string + adding in arraylist is order constant + inserting triangle in neighbor list of new triangle and vice versa with order size of neighbor + sorting order of O(nlogn)

2. Type_Mesh()- Iterate over all edges in alist2 and check the size of the triangle neighbor of the edges. Decide accordingly if any size is greater than 2 then improper mesh. Else if all are equal to 2 then proper else if semi proper. Complexity O(n) where n is the size of the arraylist of edges.
3. Boundary_Edges()- Again iterate over all edges in alist2 add those in max heap who are boundary edges (boundary edges has triangle neighbor size =1). Then extract it from max heap to array of edges and return it. Complexity O(nlogn) where n is the size of the arraylist of edges.
4. Count_connected_components()- Apply breath first search and count the number of connected component in undirected graph, where nodes are triangles who are connected by sharing a common edge. O(n+m) where n is number of nodes and m is number of edges.
5. Neighbor_of_triangle()- Iterate through all triangles in the list of triangle neighbor of given triangle and add in the triangle array and return that array. Complexity O(n) where n is the size of list of triangle neighbor.
6. Edge_neighbor_triangle()- Iterate through edge neighbor of the triangle. Complexity O(n) is the size of neighbor.
7. Vertex_neighbor_triangle()- Iterate over point neighbor. Complexity O(n)
8. Extended_neighbor_triangle()- Iterate over extended neighbor. Complexity O(n)
9. Incident_triangles()- Iterate over triangle neighbor of the given point. Complexity O(n) n is the size of array list.
10. Neighbor_of_point()- O(n)

11. Edge_neighbor_point()- O(n)
12. Face_neighbor_point()- O(n)
13. Is_connected()- Apply breath first search transversal from one point till other point is found or else if every other node is visited to confirm not visited. Complexity O(n+m) n is nodes and m is edges in that component.
14. Triangle_neighbor_edge()- O(n)
15. Maximum_diameter()- At first apply one breath first search and find out the largest component. O(n+m) . Then in that component apply BFS number of nodes times to compute farthest distance from each with in that component and return maxima. Complexity O(n(n+m)) or if m is worst then $O(n^3)$.
16. Centroid()- Apply BFS and compute as you visit a point include in centroid calculation not to include same point two times. When one component is completed insert this centroid in maxheap and do same for next component till all are done. Extract from maxheap. It will be sorted and return it. Complexity O(nlogn +m)
17. Centroid_of_component()- take any triangle neighbor to that point and from that triangle apply BFS and calculate the centroid of the component in which that triangle lie. O(n+m)
18. Closest_components()- Apply one BST in whole graph and maintained lists of all those points that lie in one component. Hence number of such list is number of different components. O(n+m) till now. Now from one component take one point and from another take another point and calculate the distance between them and return those two point with minimum distance. Complexity $O(n^2)$ where n is the number of points in the graph.