Reference Manual

This application was written for use in Project 2 - Graph ADT and Algorithms for ICS311 Fall 2013 at the University of Hawaii at Manoa to understand graph theory and its algorithms.

The methods created were implemented in accordance with the interface given in the project specifications and influenced by Goodrich and Tamassia’s Graph Abstract Data Type.

The main data structures used in the implementation of this application are ArrayLists which were traversed by the use of iterators provided by the Java standard library.

[Javadocs](Javadoc/index.html) - information about specific methods and classes and their functionalities.

Analysis of each method

\*Data Structure used was an Adjacency List so methods that processes adjacent vertices are O(E)

**Graph Class**

public Integer numVertices()

Θ(1) returns the size of the array list

public Integer numEdges()

Θ(1) returns the size of the array list

public Double density()

Θ(1) returns |E| / (|V| \* (|V| - 1))

public Integer largestSCC()

O(n) processes SCC’s and returns largest

public Double largestSCCPercent()

Θ(1) returns percent of largestSCC / numVertices

public boolean areAdjacent(Vertex<T,E> brother, Vertex<T,E> sister)

O(E) loops through one adjacency list to find the other vertex

public Iterator<Vertex<T,E>> vertices()

O(V) returns the iterator of indicated adjacency list

public Iterator<Edge<T,E>> edges()

O(E) returns the iterator of indicated adjacency list

public Iterator<Edge<T,E>> directedEdges()

O(E) returns the iterator of indicated adjacency list

public Iterator<Vertex<T,E>> finished()

O(V) returns the iterator of indicated adjacency list

public Vertex<T,E> getVertex (T key)

O(V) must loop through all the vertices in the vertices adjacency list

//Mutators

public Vertex<T,E> insertVertex(Vertex<T,E> newVertex)

Θ(1) just adds the vertex to the end of the vertices adjacency list

public Edge<T,E> insertDirectedEdge(Vertex<T,E> u, Vertex<T,E> v, Edge<T,E> edge) throws InvalidEdgeException

Θ(1) just adds to the end of corresponding adjacency lists

public Vertex<T,E> removeVertex(Vertex<T,E> v) throws InvalidEdgeException

O(V+E) loops through incident Edges and removes them as well as updating each adjacent vertex

private void removeVertex(Iterator<Vertex<T,E>> itrVert, Vertex<T,E> v)

O(V) loops through the corresponding vertex adjacency list

public Edge<T,E> removeEdge(Edge<T,E> e) throws InvalidEdgeException

O(V+E) updates each end vertex and their corresponding incident edge and adjacent vertex lists

private void removeEdge(Iterator<Edge<T,E>> itrEdge, Edge<T,E> e)

O(V+E)

public void reverseGraph() throws InvalidEdgeException

O(V+E) loops through each edge in the edge adjacency list and updates each vertex’s corresponding adjacency lists

public void depthFS (Iterator<Vertex<T,E>> itrDFS)

O(V) loops through each vertex in the iterator

private void depthFSVisit(Vertex<T, E> current)

O(V) loops through each vertex less the one first visited

**Vertex Class**

public E getAnnotation(T key)

Θ(1)

public Integer getInDeg()

Θ(1) returns the number of edges going in

public Integer getOutDeg()

Θ(1) returns the size of the array list

public Integer getDeg()

Θ(1) returns the total size of the two array lists

public Integer getNumIncidentEdges()

Θ(1) returns the size of the array list

public Integer getNumInAdjacentVertices()

Θ(1) returns the size of the array list

public Integer getNumOutAdjacentVertices()

Θ(1) returns the size of the array list

public Integer getNumAdjacentVertices()

Θ(1) returns the size of the array list

public Vertex<T,E> getOpposite (Edge<T,E> edge) throws InvalidEdgeException

Θ(1) gets opposite vertex from the edge

public Boolean isAdjacentWith(Vertex<T,E> otherVertex)

O(VE loops through adjacent vertices list

public Iterator<Vertex<T,E>> adjacentVertices()

O(E) returns the iterator of indicated adjacency list

public Iterator<Vertex<T,E>> inVertices()

O(E) returns the iterator of indicated adjacency list

public Iterator<Vertex<T,E>> outVertices()

O(E) returns the iterator of indicated adjacency list

public Iterator<Edge<T,E>> incidentEdges()

O(E) returns the iterator of indicated adjacency list

public Iterator<Edge<T,E>> inEdges()

O(E) returns the iterator of indicated adjacency list

public Iterator<Edge<T,E>> outEdges()

O(E) returns the iterator of indicated adjacency list

public void setAnnotation(T key, E value)

O(1)

public E removeAnnotation(T key)

O(1)

public E removeAnnotation(T key)

O(1)

public void removeFromEdges(Iterator<Edge<T,E>> iterator, T remove)

O(E) loops through edges adjacency list

**Arc Class**

public E getAnnotation(T key)

O(1) returns field value

public Vertex<T,E> getDestination() throws InvalidEdgeException

O(1)

public Vertex<T,E> getOrigin() throws InvalidEdgeException

O(1)

public ArrayList<Vertex<T,E>> getEnds()

O(1)

public void addEnds(Vertex<T,E> brother, Vertex<T,E> sister)

O(1)

public void deleteEnds()

O(1)

public void setDirection (Vertex<T,E> brother, Vertex<T,E> sister)

O(1) updates fields and adjacency lists

public void setDirectionTo(Vertex<T,E> newDestination) throws InvalidEdgeException

O(1) updates fields and adjacency lists

public void setDirectionFrom(Vertex<T,E> newOrigin) throws InvalidEdgeException

O(1) updates fields and adjacency lists

public void makeDirected(Vertex<T, E> brother, Vertex<T, E> sister)

O(1) updates fields and adjacency lists

public void makeUndirected()

O(1) updates fields and adjacency lists

public void reverseDirection() throws InvalidEdgeException

O(1) updates fields and adjacency lists

[Return to Index](index.html)