Team Shapiro/Criner/Baharani

```
void     addVertex(V vertex)
void     addEdge(V vsource, V vdes, double weight)
boolean  connected(V vsource, V vdes)
public   List<V> shortestPath(V vsource, V vdes)
public   List<V> search(V vsource, V vdes)
public   Double  weight(V vsource, V vdes)
public   Set<V>  reachable(V vertex)
public   Map<V, Double> reachable(V vertex)
```

unweighted undirected | Map

weighted directed | Map

Adj. matrix.

ected => public

directed => public

Methods

int getNumOfVertices

Set<V> getNeighbors(V)

boolean addVertex(V, Set<V>)

boolean addEdge(V1, V2)

boolean addConnection(V_1, V_2)

Set<V> getAllVertices

boolean isConnected(V_1, V_2)

Weighted Directed

double getWeight(V_1, V_2)

Set getVerticesInto(V_1) // all vertices pointing to a node

Set getNeighbors(V_1) // all vertices V_1 points to

It's personal

int getNumOfVertices()

Set<V> getAllVertices()

Boolean addVertex(v, map<v, double>)

boolean addConnection(v_1, v_2, d)

boolean passAllTest()

Vertex Class

Variables— V label

vertices connected to it is one weight (1 if unweighted) → Map<Vertex, Double>

A B

~~B A~~

Methods:

getWeight(Vertex) // Weight on edge from this Vertex to passed Vertex, 0 if no edge

getLabel()

getVerticesPointedTo() // Return Key set

addEdge(Vertex, Double)

removeEdge(Vertex)

containsEdge(Vertex)

Team — — — —

UnWeightedGraph<E>

HashSet<Vertex> vertices;

constructor(...) →

constructor(Collection<Vertex>) →

constructor() →

addVertex →

deleteVertex →

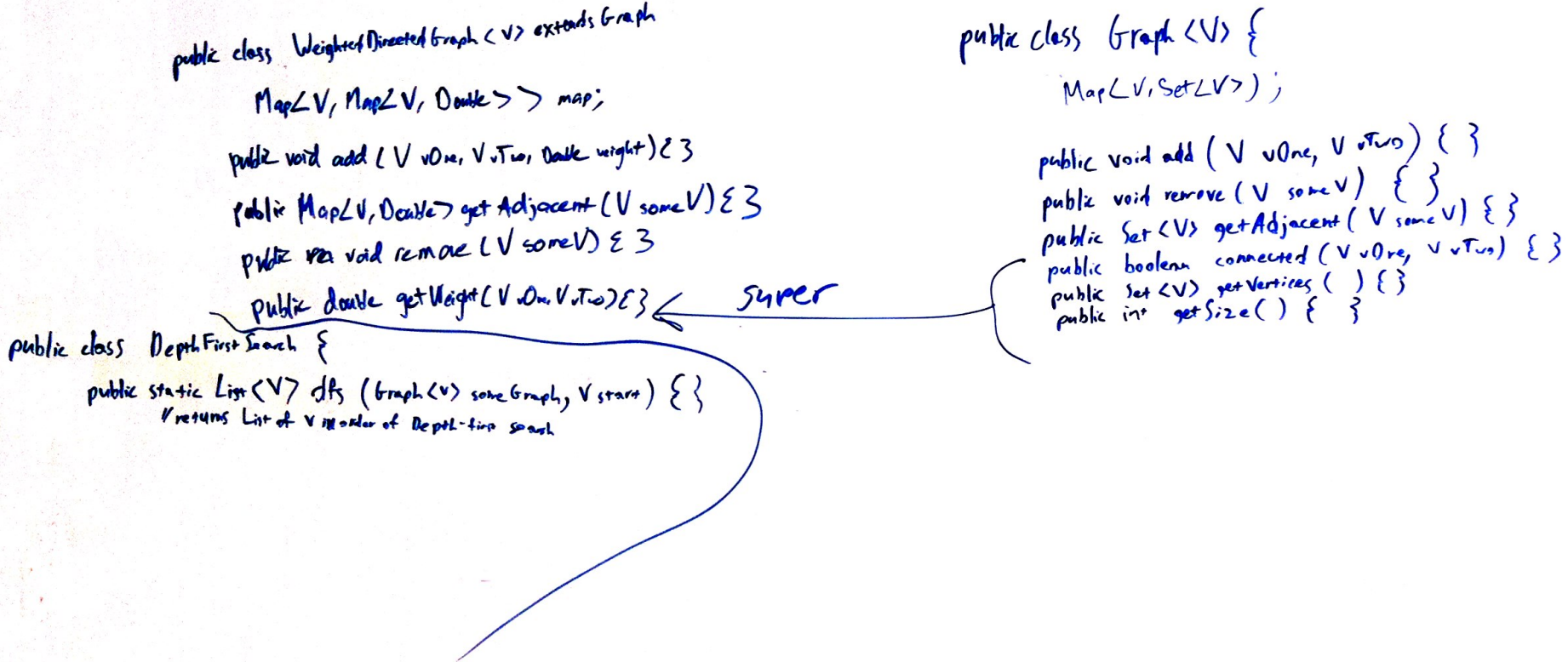
containsVertex →

hasEdge(Vertex 1, Vertex 2) →

parent class

WeightedDirectedGraph<E>

HashSet<Vertex>


public class WeightedDirectedGraph<V> extends Graph
Map<V, Map<V, Double>> map;
public void add(V vOne, V vTwo, Double weight){}
public Map<V, Double> getAdjacent(V someV){}
public void remove(V someV){}
public double getWeight(V vOne, V vTwo){}
public class DepthFirstSearch {
public static List<V> dfs(Graph<V> someGraph, V start) {}
//returns List of V in order of Depth-first search
super
public class Graph<V> {
Map<V, Set<V>>;
public void add(V vOne, V vTwo) { }
public void remove(V someV) { }
public Set<V> getAdjacent(V someV) { }
public boolean connected(V vOne, V vTwo) { }
public Set<V> getVertices() { }
public int getSize() { }

Weighted

Map<V, Map<V, double>>

boolean addEdge(V0, V1, w)

Check V0

Map.put(V0, (new Map(V1, w))

Map.get(V0).put(V1, w)

List<V> getVertTo(V v)

List<V> getVertFrom(V v)

boolean removeEdge(V v1, V v2)

boolean removeVert(V v)

int getNumVert

boolean isEdge

int getWeight

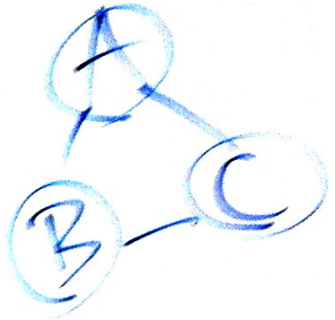
Un weight

weighted Graph

addEdge(V1, V2)

graph.addEdge(V1, V2, 1)

graph.addEdge(V2, V1, 1)



getNeighbor(V) list<V>

return graph.getVertexTo(V);

remove