edu.uci.ics.jung.graph

Interface Graph<V,E>

All Superinterfaces:

Hypergraph<V,E>

All Known Subinterfaces:

DirectedGraph<V,E>, Forest<V,E>, KPartiteGraph<V,E>, Tree<V,E>, UndirectedGraph<V,E>

All Known Implementing Classes:

AbstractGraph, AbstractTypedGraph, DelegateForest, DelegateTree, DirectedOrderedSparseMultigraph, DirectedSparseGraph,DirectedSparseMultigraph, GraphDecorator, ObservableGraph, OrderedKAryTree, OrderedSparseMultigraph, SortedSparseMultigraph,SparseGraph, SparseMultigraph, UndirectedOrderedSparseMultigraph, UndirectedSparseGraph, UndirectedSparseMultigraph

---

```
public interface Graph<V,E>
extends Hypergraph<V,E>
```

A graph consisting of a set of vertices of type `V` set and a set of edges of type `E`. Edges of this graph type have exactly two endpoints; whether these endpoints must be distinct depends on the implementation.

This interface permits, but does not enforce, any of the following common variations of graphs:

- directed and undirected edges
- vertices and edges with attributes (for example, weighted edges)
- vertices and edges of different types (for example, bipartite or multimodal graphs)
- parallel edges (multiple edges which connect a single set of vertices)
- representations as matrices or as adjacency lists or adjacency maps

Extensions or implementations of this interface may enforce or disallow any or all of these variations.

Definitions (with respect to a given vertex `v`):

- **incoming edge** of `v`: an edge that can be traversed from a neighbor of `v` to reach `v`
- **outgoing edge** of `v`: an edge that can be traversed from `v` to reach some neighbor of `v`
- **predecessor** of `v`: a vertex at the other end of an incoming edge of `v`
- **successor** of `v`: a vertex at the other end of an outgoing edge of `v`

Method Summary

`boolean`	**`addEdge`**`(E e, V v1, V v2)` Adds edge `e` to this graph such that it connects vertex `v1` to `v2`.
`boolean`	**`addEdge`**`(E e, V v1, V v2, EdgeType edgeType)` Adds edge `e` to this graph such that it connects vertex `v1` to `v2`.
`V`	**`getDest`**`(E directed_edge)` If `directed_edge` is a directed edge in this graph, returns the destination; otherwise returns `null`.

Pair<V>	getEndpoints(E edge) Returns the endpoints of edge as a Pair.
Collection<E>	getInEdges(V vertex) Returns a Collection view of the incoming edges incident to vertex in this graph.
V	getOpposite(V vertex, E edge) Returns the vertex at the other end of edge from vertex.
Collection<E>	getOutEdges(V vertex) Returns a Collection view of the outgoing edges incident to vertex in this graph.
int	getPredecessorCount(V vertex) Returns the number of predecessors that vertex has in this graph.
Collection<V>	getPredecessors(V vertex) Returns a Collection view of the predecessors of vertex in this graph.
V	getSource(E directed_edge) If directed_edge is a directed edge in this graph, returns the source; otherwise returns null.
int	getSuccessorCount(V vertex) Returns the number of successors that vertex has in this graph.
Collection<V>	getSuccessors(V vertex) Returns a Collection view of the successors of vertex in this graph.
int	inDegree(V vertex) Returns the number of incoming edges incident to vertex.
boolean	isDest(V vertex, E edge) Returns true if vertex is the destination of edge.
boolean	isPredecessor(V v1, V v2) Returns true if v1 is a predecessor of v2 in this graph.
boolean	isSource(V vertex, E edge) Returns true if vertex is the source of edge.
boolean	isSuccessor(V v1, V v2) Returns true if v1 is a successor of v2 in this graph.
int	outDegree(V vertex) Returns the number of outgoing edges incident to vertex.

Methods inherited from interface edu.uci.ics.jung.graph.Hypergraph

addEdge, addEdge, addVertex, containsEdge, containsVertex, degree, findEdge, findEdgeSet, getDefaultEdgeType, getEdgeCount, getEdgeCount, getEdges, getEdges, getEdgeType, getIncidentCount, getIncidentEdges, getIncidentVertices, getNeighborCount, getNeighbors, getVertexCount, getVertices, isIncident, isNeighbor, removeEdge, removeVertex