

Directed Graph - Documentation

Specification

We define a class name DirectedGraph representing a directed graph

For Java we have an additional class called Pair<T1,T2> with the following public fields:

- first: The first element of the pair of type T1
- second: The second element of the pair of type T2

For both implementation we have the following functions:

C++	Java
<pre>DirectedGraph read_from_file(std::string filename);</pre>	<pre>public static DirectedGraph read_from_file(String filename)</pre>
<p>Reads from a file the graph in the following format: First line: n m Following m lines: v1 v2 w Where n is the number of vertices, m the number of edges, v1,v2 vertices and w the weight between v1 and v2 The vertices are number from 0 to n-1</p>	
<pre>void write_to_file(const DirectedGraph& g, std::string filename);</pre>	<pre>public static void write_to_file(DirectedGraph graph, String filename)</pre>
<p>Writes to a file the graph from memory reindexed in the following format: First line: n m Following m lines: v1 v2 w Where n is the number of vertices, m the number of edges, v1,v2 vertices and w the weight between v1 and v2 The vertices are number from 0 to n-1</p>	
<pre>DirectedGraph random_graph(int num_vertices, int num_edges);</pre> <p>Generates a random graph with the corresponding number of vertices and edges</p>	<pre>public static DirectedGraph random_graph(Integer num_vertices, Integer num_edges)</pre>

The class DirectedGraph provides the following public methods:

C++	Java
<code>int numVertices() const;</code>	<code>public Integer numVertices();</code>
Returns the number of vertices in the graph	
<code>int numEdges() const;</code>	<code>public Integer numEdges()</code>
Returns the number of edges in the graph	
<code>std::unordered_set<int> vertices() const;</code>	<code>public Iterator<Integer> vertices();</code>
Returns an iterator object of all vertices	
<code>std::unordered_map<std::pair<int,int>,int> edges() const;</code>	<code>public Iterator<Pair<Pair<Integer,Integer>,Integer >> edges();</code>
Returns an iterator object of all edges	
<code>bool isEdge(int v1, int v2) const;</code>	<code>public boolean isEdge(Integer v1, Integer v2);</code>
Returns true if there exists an edge between v1 and v2, false otherwise Throws an exception if v1 or v2 are not a valid vertices	
<code>int inDegree(int v) const;</code>	<code>public int inDegree(Integer v)</code>
Returns the in degree of the vertex v Throws an exception if v is not a valid vertex	
<code>int outDegree(int v) const;</code>	<code>public int outDegree(Integer v);</code>
Returns the out degree of the vertex v Throws an exception if v is not a valid vertex	
<code>std::unordered_set<int> inbound(int v) const;</code>	<code>public Iterator<Integer> inbound(Integer v);</code>
Returns an iterator object of all inbound vertices of v Throws an exception if v is not a valid vertex	
<code>std::unordered_set<int> outbound(int v) const;</code>	<code>Iterator<Integer> outbound(Integer v);</code>
Returns an iterator object of all outbound vertices of v Throws an exception if v is not a valid vertex	

<pre>int weight(int v1, int v2) const;</pre>	<pre>public Integer weight(Integer v1, Integer v2);</pre>
<p>Returns the weight of the edge between v1 and v2 Throws an exception if v1 or v2 are not valid vertices or there exists no edge between v1 and v2</p>	
<pre>void weight(int v1, int v2, int w);</pre>	<pre>public void weight(Integer v1, Integer v2, Integer w);</pre>
<p>Sets the weight of the edge between v1 and v2 Throws an exception if v1 or v2 are not valid vertices or there exists no edge between v1 and v2</p>	
<pre>void addVertex(int v);</pre>	<pre>public void addVertex(Integer v);</pre>
<p>Adds a vertex to the graph Throws an exception if v is already in the graph</p>	
<pre>void removeVertex(int v);</pre>	<pre>public void removeVertex(Integer v);</pre>
<p>Removes a vertex from the graph Throws an exception if v is not in the graph</p>	
<pre>void addEdge(int v1, int v2, int w);</pre>	<pre>public void addEdge(Integer v1, Integer v2, Integer w);</pre>
<p>Adds an edge between v1 and v2 with weight w Throws an exception if v1 or v2 are not valid vertices or there exists already an edge between them</p>	
<pre>void removeEdge(int v1, int v2);</pre>	<pre>public void removeEdge(Integer v1, Integer v2);</pre>
<p>Removes the edge between v1 and v2 Throws an exception if v1 or v2 are not valid vertices or if no edge exists between v1 and v2</p>	
<pre>DirectedGraph reindex() const;</pre>	<pre>public DirectedGraph reindex();</pre>
<p>Returns an reindexed copy of the graph with vertices having values between 0 and n-1 where n is the number of vertices</p>	

Implementation

The class DirectedGraph has the following data members:

C++	Java
<pre>std::unordered_set<int> _vertices;</pre> <p>Represents a vertices from the graph</p>	<pre>private HashSet<Integer> _vertices = new HashSet<Integer>();</pre>
<pre>std::unordered_map<int,std::unordered_s et<int>> _inbounds;</pre> <p>Represents a map of the inbounds of all vertices of the graph</p>	<pre>private HashMap<Integer, HashSet<Integer>> _inbounds = new HashMap<Integer, HashSet<Integer>>();</pre>
<pre>std::unordered_map<int,std::unordered_s et<int>> _outbounds;</pre> <p>Represents a map of the outbounds of all vertices of the graph</p>	<pre>private HashMap<Integer, HashSet<Integer>> _outbounds = new HashMap<Integer, HashSet<Integer>>();</pre>
<pre>std::unordered_map<std::pair<int,int>,int> _weights;</pre> <p>Represents a map of all pair of edges that has an edge as a key and the weight as a value</p>	<pre>private HashMap<Pair<Integer,Integer>,Integer> _edges = new HashMap<Pair<Integer,Integer>,Integer>(</pre> <p>)</p>
<p>std::unordered_map and std::unordered_set have been used to achieve O(1) operations on the graph (reindex being an exception)</p>	<p>HashMap and HashSethave been used to achieve O(1) operations on the graph (reindex being an exception)</p>