# Java week 11

**public** **class** ReferenceCountTopologicalSort<T> **extends** AdjacencyGraph<T> **implements** TopologicalSort<T>

{

//create our hashmap

HashMap<T, Integer> map = **new** HashMap<T, Integer>();

//create our list for sorted nodes

List<T> sortedNodes = **new** ArrayList<T>();

@Override

**public** List<T> getSort() **throws** GraphError

{

initialise();

setUpReferenceCounts();

addToSort();

System.***out***.println(map);

System.***out***.println(sortedNodes);

**return** sortedNodes;

}

**private** **void** initialise()

{

//add all our empty nodes

**for** (T node : getNodes())

{

map.put(node, 0);

}

}

**private** **void** setUpReferenceCounts() **throws** GraphError

{

//for every object in every node

**for** (T node : getNodes())

{

//for every object in every neighbours node

**for** (T successor : getNeighbours(node))

{

**int** references = map.get(successor);

//if the current neighbour is not null

**if** (map.get(successor) != **null**)

{

//add to our hashmap

map.put(successor, ++references);

}

}

}

}

**public** **void** addToSort() **throws** GraphError

{

//whilst we have not sorted the nodes

**while**(sortedNodes.size() < getNodes().size())

{

**for** (T node : getNodes())

{

//if the node is not null and the value is more than 0

**if** (map.get(node) != **null** &&map.get(node).intValue() == 0)

{

//add to our sorted nodes

sortedNodes.add(node);

//for every neighbour

**for** (T successor : getNeighbours(node))

{

Integer references = map.get(successor);

//if our successor is not null

**if** (references != **null**)

{

//add to our map

map.put(successor, references - 1);

}

}

//remove from our map

map.remove(node);

**break**;

}

}

}

}

**public** **static** **void** main(String[] args) **throws** GraphError

{

ReferenceCountTopologicalSort<Integer> graph = **new** ReferenceCountTopologicalSort<>();

Integer node0 = **new** Integer(0);

Integer node1 = **new** Integer(1);

Integer node2 = **new** Integer(2);

Integer node3 = **new** Integer(3);

Integer node4 = **new** Integer(4);

Integer node5 = **new** Integer(5);

Integer node6 = **new** Integer(6);

Integer node7 = **new** Integer(7);

Integer node8 = **new** Integer(8);

Integer node9 = **new** Integer(9);

graph.add(node0);

graph.add(node1);

graph.add(node2);

graph.add(node3);

graph.add(node4);

graph.add(node5);

graph.add(node6);

graph.add(node7);

graph.add(node8);

graph.add(node9);

graph.add(0, 1);

graph.add(0, 5);

graph.add(1, 7);

graph.add(3, 2);

graph.add(3, 4);

graph.add(3, 8);

graph.add(6, 0);

graph.add(6, 1);

graph.add(6, 2);

graph.add(8, 4);

graph.add(8, 7);

graph.add(9, 4);

graph.getSort();

}

{}

[3, 6, 0, 1, 2, 5, 8, 7, 9, 4]