Import upperbound=newbound; } public double residualcapacityto(int vertex) {

import java.

util.*; class FlowNetworkGraph {private int vertexCount; private int edgeCount; private ArrayList > graph; public FlowNetworkGraph(int vertexCount){this. vertexCount = vertexCount; graph = new ArrayList >(vertexCount); for(int i= 0; i ());}}public void addEdge(FlowEdge edge){int v = edge. from(); int w = edge.

to(); graph. get(v). add(edge); graph. get(w). add(edge); edgeCount+ +;}public void addVertexPlaceholder(){//When a new vertex needs to be appended to the graphgraph.

add(new ArrayList ()); vertexCount++;}public int vertexCount(){return vertexCount;}public int edgeCount(){return edgeCount;}public Iterable adjacentTo(int vertex){return graph. get(vertex);}public Iterable edges() {ArrayList edges = new ArrayList (vertexCount); for(int i= 0; i

fromVertex = fromVertex; this. toVertex = toVertex; this. capacity = capacity;}public FlowEdge(int fromVertex, int toVertex, double lowerBound, double upperBound){this.

fromVertex = fromVertex; this. toVertex = toVertex; this. lowerBound = lowerBound; this.

upperBound = upperBound;}public int from(){return fromVertex;}public int to(){return toVertex;}public int otherVertex(int vertex){if(vertex== this. fromVertex){return toVertex;}else{return fromVertex;}}public double getCapacity(){return capacity;}public void setCapacity(double capacity) {this. capacity= capacity;}public double flow(){return flow;}public double

getLowerBound(){return lowerBound;}public void setLowerBound(double newBound){lowerBound= newBound;}public double getUpperBound() {return upperBound;}public void setUpperBound(double newBound) {upperBound= newBound;}public double residualCapacityTo(int vertex) {if(vertex== toVertex){return capacity-flow;}else return flow;}public void addResidualFlowTo(int vertex, double changeInFlow){if(vertex== this. toVertex){flow+= changeInFlow;}else{flow-= changeInFlow;}}@Overridepublic String toString(){return “”+fromVertex+”–>”+toVertex+” (capacity=”+capacity+”)”;}}public class CirculationWithDemands {private double maxFlow = 0; private int sumOfDemands = 0; private int sumOfSupplies = 0; private int lowerBoundsAdjustedsumOfDemands= 0; private int lowerBoundsAdjustedsumOfSupplies= 0; private boolean doDemandsMatchSupplies= true; private boolean hasLowerBounds = false; private boolean marked; private FlowEdge edgeTo; public CirculationWithDemands(FlowNetworkGraph graph, ArrayList vertexName, int vertexDemand){ArrayList demandVertices = new ArrayList (); ArrayList supplyVertices = new ArrayList (); for(int vertex= 0; vertex 0) {demandVertices.

add(vertex); sumOfDemands += vertexDemandvertex;}else if(vertexDemandvertex <0){supplyVertices. add(vertex); sumOfSupplies += -vertexDemandvertex;//negative}//If demand= 0 nothing needs to change, vertex is not connected to source or sink}if(sumOfSupplies != sumOfDemands){doDemandsMatchSupplies= false;}if(doDemandsMatchSupplies){//Only continue if supplies/demands are

valid//Process edges and adjust for lower boundsfor(FlowEdge edge : graph. edges()){if(edge. getLowerBound() != 0){//Edges with NO lower bounds have lower bound of 0hasLowerBounds= true;//Subtract lower bounds from capacity & update boundsdouble oldLowerBound = edge.

getLowerBound(); edge. setCapacity(edge. getUpperBound() – oldLowerBound);//lower bound edges initally have no capacityedge. setUpperBound(edge.

getCapacity()); edge. setLowerBound(0);//Adjust supplies/demands for both ends of the edge. Subtract oldLowerBound if vertex is a demand vertex (> 0) & add if it’s a supply vertex (<0)if(vertexDemandedge. from()> 0) {vertexDemandedge. from() -= oldLowerBound;}else{vertexDemandedge. from() += oldLowerBound;}if(vertexDemandedge. to()> 0) {vertexDemandedge. to() -= oldLowerBound;}else{vertexDemandedge. to() += oldLowerBound;}}}//Recalculate Sum of supplies/demands with adjusted boundsif(hasLowerBounds){lowerBoundsAdjustedsumOfDemands= 0; lowerBoundsAdjustedsumOfSupplies= 0; for(int vertex= 0; vertex 0) {lowerBoundsAdjustedsumOfDemands += vertexDemandvertex;}else if(vertexDemandvertex <0){lowerBoundsAdjustedsumOfSupplies += -vertexDemandvertex;//negative}//If demand= 0 nothing needs to change, vertex is not connected to source or sink}if(lowerBoundsAdjustedsumOfSupplies != lowerBoundsAdjustedsumOfDemands){doDemandsMatchSupplies= false;}}if(doDemandsMatchSupplies){//Add S & T, connect to supply/demand verticesint source = graph. vertexCount(); int sink = source + 1; vertexName. add(" S"); vertexName. add(" T"); graph.

addVertexPlaceholder(); graph. addVertexPlaceholder();//Connect demand vertices to sink & source vertex to all supply verticesfor(int vertex : demandVertices){graph. addEdge(new FlowEdge(vertex, sink, vertexDemandvertex));}for(int vertex : supplyVertices){graph. addEdge(new FlowEdge(source, vertex, -vertexDemandvertex));//negative of the demand value to get a positive capacity }//Begin Ford Fulkerson partmaxFlow = 0; while(hasAugmentingPath(graph, source, sink)){ double bottneckFlow = Double. POSITIVE_INFINITY;//Loop backwards over path & find the bottleneck flowArrayList augmentingPathBackwards = new ArrayList ();//save vertices on the path while looping backwardsfor(int v = sink; v!= source; v= edgeTov. otherVertex(v)){augmentingPathBackwards. add(v); bottneckFlow = Math. min(bottneckFlow, edgeTov. residualCapacityTo(v));}//Update residual Capacitiesfor(int v = sink; v!= source; v= edgeTov. otherVertex(v)) {edgeTov. addResidualFlowTo(v, bottneckFlow);}System. out. print(“ Min Edge Weight=”+bottneckFlow); System. out. print(” Augmenting Path: “); System. out. print(vertexName. get(source)); for(int i= augmentingPathBackwards. size()-1; i>= 0; i–){System. out. print(“–>”+vertexName. get(augmentingPathBackwards. get(i)));}System. out. println(); maxFlow += bottneckFlow;}}}displayOutputMessages(graph, vertexName);}//Breadth first searchpublic boolean hasAugmentingPath(FlowNetworkGraph graph, int source, int sink){edgeTo = new FlowEdgegraph. vertexCount(); marked = new booleangraph. vertexCount(); Queue vertexQueue = new LinkedList (); vertexQueue. add(source);//add & visit the source vertexmarkedsource = true; while(! vertexQueue. isEmpty()){int vertex = vertexQueue. poll();//remove vertex from head of queuefor(FlowEdge edge : graph. adjacentTo(vertex)){int

otherVertex = edge. otherVertex(vertex); if(edge. residualCapacityTo(otherVertex)> 0 && ! markedotherVertex){//if vertex has residual capacity & is unvisitededgeTootherVertex = edge;//update the edges leading out of otherVertexmarkedotherVertex = true;//visit the new vertexvertexQueue. add(otherVertex);//and add to queue}}}return markedsink;//did BFS visit the target}public double maxFlow(){return maxFlow;}public int sumOfDemands(){return sumOfDemands;}public int sumOfSupplies(){return sumOfSupplies;}public boolean doDemandsMatchSupplies(){return doDemandsMatchSupplies;}public boolean hasCirculation(){if(! doDemandsMatchSupplies){return false;}else if(hasLowerBounds){if(maxFlow!= lowerBoundsAdjustedsumOfSupplies || maxFlow!= lowerBoundsAdjustedsumOfDemands){return false;}}else if(maxFlow!= sumOfSupplies || maxFlow!= sumOfDemands){return false;}return true;}private void displayOutputMessages(FlowNetworkGraph graph, ArrayList vertexName){if(hasCirculation()){System. out. println(“ Graph has Circulation Maxflow value = “+maxFlow); System. out. println(“ Mincut vertices: “); for(int v= 0; v vertexNameGraph1 = new ArrayList (Arrays. asList(“ A”, “ B”, “ C”, “ D”)); int vertexDemandGraph1 = {-3, -3, 2, 4}; CirculationWithDemands circulationFinderGraph = new CirculationWithDemands(graph1, vertexNameGraph1, vertexDemandGraph1);}}