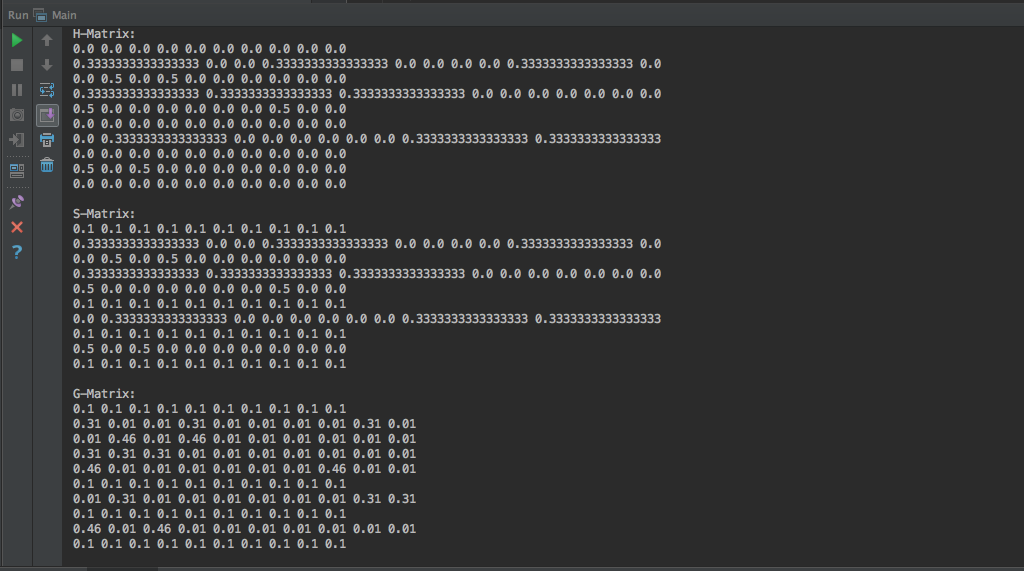
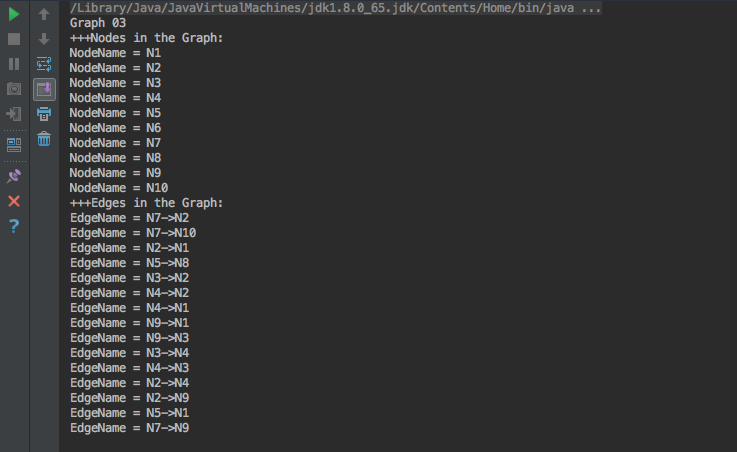
Justin Larkin

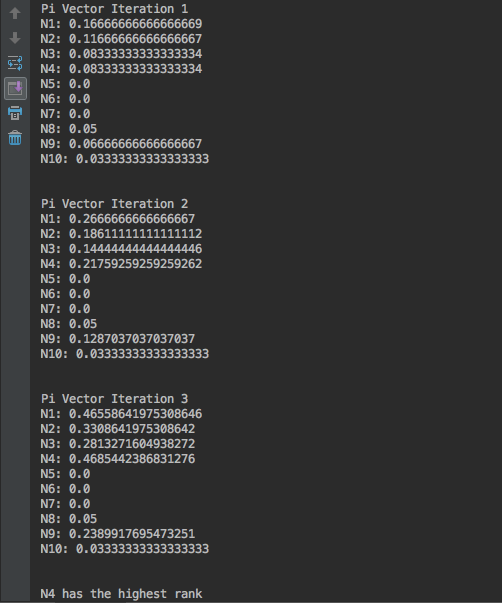
CSCI 5330

Final Project: Page Rank

05/04/16

The purpose of this project was to find the page rank of nodes in a certain graph. We built a graph using nodes and edges imported from a text file and represented the graph as an adjacency matrix (H-Matrix). We transformed this matrix into a matrix (G-Matrix) that is used by Google to calculate page rank based on the pi vector of each node.





*/\*\*  
 \* Created by jlarkin9 on 4/25/16.  
 \*/*public class Node {  
 String name;  
 int id;  
  
 public Node(String name) {  
 this.name = name;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
}

*/\*\*  
 \* Created by jlarkin9 on 4/25/16.  
 \*/*public class Edge {  
 String name;  
 int id;  
 boolean directed;  
 Node fromNode;  
 Node toNode;  
  
 public Edge(String name) {  
 this.name = name;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 //Getters and Setters  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public Node getToNode() {  
 return toNode;  
 }  
  
 public void setToNode(Node toNode) {  
 this.toNode = toNode;  
 }  
  
 public Node getFromNode() {  
 return fromNode;  
 }  
  
 public void setFromNode(Node fromNode) {  
 this.fromNode = fromNode;  
 }  
  
 public boolean isDirected() {  
 return directed;  
 }  
  
 public void setDirected(boolean directed) {  
 this.directed = directed;  
 }  
}

import java.util.HashMap;  
import java.util.Map;  
  
*/\*\*  
 \* Created by jlarkin9 on 4/25/16.  
 \*/*public class Set {  
  
 int count = 0;  
 HashMap<Integer, Object> map = new HashMap<Integer, Object>();  
  
  
 public void add(Object o) {  
 map.put(count, o);  
 count++;  
  
 }  
  
  
 public void remove(Object o) {  
 map.remove(o);  
 }  
  
  
 public int find(Object o) {  
 int key = 0;  
 for (int i = 0; i < map.size(); i++) {  
 if (o.equals(map.get(i)))  
 key = i;  
 }  
 return key;  
 }  
  
  
}

import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.List;  
  
*/\*\*  
 \* Created by jlarkin9 on 4/25/16.  
 \*/*public class NodeSet extends Set {  
  
 int count = 0;  
 HashMap<Integer, Node> map = new HashMap<Integer, Node>();  
  
 public NodeSet() {  
  
 }  
  
  
 public void add(Node node) {  
 map.put(count, node);  
 //node.setId(count);  
 count++;  
  
 }  
  
  
 public void remove(String name) {  
 map.remove(find(name));  
 }  
  
  
 public Node find(String name) {  
 int key = 0;  
 for (int i = 0; i < map.size(); i++) {  
 if (name.equals(map.get(i).getName()))  
 key = i;  
 }  
 return map.get(key);  
 }  
  
  
 public Node get(int index) {  
 return map.get(index);  
 }  
  
  
}

import java.util.HashMap;  
  
*/\*\*  
 \* Created by jlarkin9 on 4/26/16.  
 \*/*public class EdgeSet extends Set {  
 HashMap<Integer, Edge> map = new HashMap<Integer, Edge>();  
  
  
  
 int count = 0;  
  
// public EdgeSet() {  
// HashMap<Integer, Edge> map = new HashMap<Integer, Edge>();  
// }  
  
  
 public void add(Edge Edge) {  
 map.put(count, Edge);  
 count++;  
 }  
  
  
 public void remove(Edge Edge) {  
 map.remove(find(Edge));  
 }  
  
  
 public int find(Edge Edge) {  
 int key = 0;  
 for (int i = 0; i < map.size(); i++) {  
 if (Edge.equals(map.get(i)))  
 key = i;  
 }  
 return key;  
 }  
  
  
 public Edge get(int index) {  
 return map.get(index);  
 }  
  
 public int getCount() {  
 return count;  
 }  
  
}

import java.io.BufferedReader;  
import java.io.File;  
import java.io.FileReader;  
import java.io.IOException;  
import java.text.DecimalFormat;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Arrays;  
  
*/\*\*  
 \* Created by jlarkin9 on 4/25/16.  
 \*/*public class Graph {  
 private String name;  
 private Double [][] hMatrix;  
 private Double [][] sMatrix;  
 private Double [][] gMatrix;  
 private Double [] piVector;  
 private Double [] a;  
  
  
 NodeSet nodes = new NodeSet();  
 EdgeSet edges = new EdgeSet();  
  
  
  
 public void createGraphFromFile(String file) throws IOException {  
  
 // read text file into array list of strings  
 BufferedReader in = new BufferedReader(new FileReader(file));  
 String str;  
  
 List<String> list = new ArrayList<String>();  
 while((str = in.readLine()) != null){  
 list.add(str);  
 }  
  
 String[] rawFromFile = list.toArray(new String[0]);  
 String trimmed;  
 for(String line : rawFromFile) {  
  
 //add nodes and edges to respective sets  
 if (line.contains("NodeName")) {  
 trimmed = line.substring(line.lastIndexOf("=") + 2);  
 Node node = new Node(trimmed);  
 nodes.add(node);  
 int nodeID = Integer.*parseInt*(trimmed.substring(1));  
 node.setId(nodeID);  
 }  
  
 if (line.contains("EdgeName")) {  
 trimmed = line.substring(line.lastIndexOf("=") + 2);  
 Edge edge = new Edge(trimmed);  
 edges.add(edge);  
  
 }  
 }  
 //set to and from nodes for each edge  
 for(int i = 0; i < edges.count; i++) {  
 String n = edges.get(i).getName();  
 String fromNode = n.substring(0, n.lastIndexOf("-"));  
 int fromID = Integer.*parseInt*(fromNode.substring(1));  
 String toNode = n.substring(n.lastIndexOf(">") + 1);  
 int toID = Integer.*parseInt*(toNode.substring(1));  
  
 edges.get(i).setFromNode(nodes.find(fromNode));  
 edges.get(i).setToNode(nodes.find(toNode));  
 edges.get(i).getFromNode().setId(fromID);  
 edges.get(i).getToNode().setId(toID);  
  
 }  
  
 }  
  
 public void describeGraph() {  
  
 System.*out*.println("+++Nodes in the Graph:");  
 for(int i = 0; i < nodes.count; i++) {  
 System.*out*.println("NodeName = " + nodes.get(i).getName());  
  
 }  
  
 System.*out*.println("+++Edges in the Graph:");  
 for(int i = 0; i < edges.count; i++) {  
 System.*out*.println("EdgeName = " + edges.get(i).getName());  
 }  
 }  
  
 public void create\_hMatrix() {  
  
 //create matrix and fill with 0  
 hMatrix = new Double[nodes.count][nodes.count];  
 for(int j = 0; j < nodes.count;j++) {  
 for (int n = 0; n < nodes.count;n++)  
 hMatrix[j][n] = 0.0;  
 }  
  
 //count occurrences of outlink  
 int [] outLinkCounter = new int [nodes.count];  
 for (int i = 0; i < nodes.count; i++) {  
 for (int j = 0; j < edges.count; j++) {  
 if (edges.get(j).getFromNode() == nodes.get(i))  
 outLinkCounter[i]++;  
 }  
 }  
  
 //fill matrices with calculated numbers  
 for(int i = 0; i < edges.count; i++ ) {  
 int k = edges.get(i).fromNode.getId() - 1;  
 int l = edges.get(i).toNode.getId() - 1;  
 hMatrix[k][l] = (1.0 / outLinkCounter[k]);  
 }  
 printMatrix(hMatrix, "\nH-Matrix: ");  
  
 }  
  
 public void create\_sMatrix() {  
  
 //fill matrix  
 sMatrix = new Double[nodes.count][nodes.count];  
 for(int j = 0; j < nodes.count;j++) {  
 for (int n = 0; n < nodes.count;n++)  
 sMatrix[j][n] = hMatrix[j][n];  
 }  
  
 boolean[] dangle = findDangling();  
 for(int i = 0; i < nodes.count;i++) {  
 for(int j = 0; j < nodes.count;j++) {  
 //if dangling node, value is 1/n  
 if (dangle[i] == true) {  
 sMatrix[i][j] = (1.0/nodes.count);  
 }  
 }  
 }  
 printMatrix(sMatrix, "\nS-Matrix: ");  
  
 }  
  
 public void create\_gMatrix() {  
  
 gMatrix = new Double[nodes.count][nodes.count];  
 boolean[] dangle = findDangling();  
 DecimalFormat df = new DecimalFormat("#.##########");  
  
 for(int j = 0; j < nodes.count;j++) {  
 for (int n = 0; n < nodes.count;n++)  
 //if dangling node, value is the same as S-Matrix  
  
 if(dangle[j] == true) {  
 gMatrix[j][n] = Double.*valueOf*(df.format(sMatrix[j][n]));  
 }  
 else  
 gMatrix[j][n] = Double.*valueOf*(df.format((sMatrix[j][n] \* 0.9) + ((0.1 \* (1.0 / nodes.count)))));  
  
 }  
 printMatrix(gMatrix, "\nG-Matrix: ");  
  
 }  
  
 public void computePageRank() {  
  
 // initialize arrays  
  
 piVector = new Double[nodes.count];  
 a = new Double[nodes.count];  
 for (int x = 0; x < nodes.count;x++) {  
 a[x] = (1.0 / nodes.count);  
 piVector[x] = 0.0;  
 }  
  
 //create piVector and iterate 3 times  
 for(int n = 1; n <= 3; n++) {  
 System.*out*.println("\nPi Vector Iteration " + n);  
 for (int j = 0; j < nodes.count; j++) { // bColumn  
 for (int k = 0; k < nodes.count; k++) { // aColumn  
 piVector[j] += a[k] \* hMatrix[k][j];  
 }  
 System.*out*.println(nodes.get(j).getName() + ": " + piVector[j]);  
  
 }  
 a = piVector;  
 System.*out*.println();  
 }  
  
 //find max  
  
 Double max = piVector[0];  
 int key = 0;  
 for (int i = 1; i < nodes.count; i++) {  
 if (piVector[i] > max) {  
 max = piVector[i];  
 key = i;  
 }  
 }  
 System.*out*.println("\n" + nodes.get(key).getName() + " has the highest rank");  
  
 }  
  
 //find dangling nodes  
  
 public boolean[] findDangling() {  
  
 boolean[] danglingRows = new boolean[nodes.count];  
 for (int i = 0; i < nodes.count; i++) {  
 boolean dangling = true;  
 double first = sMatrix[i][0];  
 for (int j = 1; j < nodes.count; j++) {  
 if (sMatrix[i][j] != first) dangling = false;  
 }  
 if (dangling)  
 danglingRows[i] = true;  
 else  
 danglingRows[i] = false;  
 }  
  
 return danglingRows;  
  
 }  
  
 public void printMatrix(Double[][] matrix, String type) {  
  
 System.*out*.println(type);  
 for (int i = 0; i < nodes.count; i++) {  
 for (int j = 0; j < nodes.count; j++) {  
 System.*out*.print(matrix[i][j] + " ");  
 }  
 System.*out*.print("\n");  
 }  
 }  
  
}

import java.io.IOException;  
  
public class Main {  
  
 public static void main(String[] args) throws IOException {  
 Graph graph = new Graph();  
 System.*out*.println("Graph 03");  
 graph.createGraphFromFile("/Users/jlarkin9/Downloads/PageRank\_03-2.txt");  
  
 graph.describeGraph();  
 graph.create\_hMatrix();  
 graph.create\_sMatrix();  
 graph.create\_gMatrix();  
 graph.computePageRank();  
  
 }  
}