**Lab 11 Report**

**CS303L Algorithms and Data Structures**

**Sam Lazrak**

**Spring Semester 2018**

**Objectives:**

* Implement Depth First Search algorithm on graphs.
* Implement Topological Sort

**In-class Assignment:**

1. Write a class DepthFirstPaths.java to implement a Depth First Search algorithm using the pseudocode given below. Alternatively, you can download the file DepthFirstPaths.java and implement all the unimplemented methods of the class so that it performs Depth First Search on graph G.
2. Write a driver program, which reads input file mediumG.txt as an undirected graph and runs the Depth First Search algorithm to find paths to all the other vertices considering 0 as the source. This driver program should display the paths in the following manner:

0 to v: “list of all the vertices traversed to go to v from 0, separated by ‘,’ ”.

The following source code was written:

package Lab11;

public class Graph {

public static void main(String[] args) throws IOException {

BufferedReader r = new BufferedReader(new FileReader("mediumG.txt"));

UndirectedGraph undirected = new UndirectedGraph(r);

System.out.println("0 to " + V + ":");

undirected.DFS(undirected, 0);

}

public static int V;

public static int E;

public LinkedList<Integer>[] adj;

public Graph()

{

V = 0;

E = 0;

}

public Graph(BufferedReader reader) throws IOException

{

String line;

line = reader.readLine();

V = Integer.parseInt(line);

line = reader.readLine();

E = Integer.parseInt(line);

adj = new LinkedList[V];

for (int v = 0; v < V; v++) {

adj[v] = new LinkedList<Integer>();

}

while ((line = reader.readLine()) != null) {

int tempV1, tempV2;

StringTokenizer st = new StringTokenizer(line, " ");

tempV1 = Integer.parseInt(st.nextToken());

tempV2 = Integer.parseInt(st.nextToken());

addEdge(tempV1, tempV2);

}

}

public void addEdge(int v, int w) {

//override

}

public String tostring()

{

String s = new String();

s = "There are "+V+" vertices and "+E+" edges\n";

for(int i=0;i<V;i++)

{

s = s+i+": ";

for(int j = 0; j<adj[i].size();j++)

{

s = s+adj[i].get(j)+" ";

}

s = s+"\n";

}

return s;

}

public void DFS(Graph G, int s) {

boolean visited[] = new boolean[G.V];

DFSVisit(G, s, visited);

}

void DFSVisit(Graph G, int s, boolean visited[]) {

visited[s] = true;

System.out.print(s+", ");

Iterator<Integer> i = G.adj[s].listIterator();

while (i.hasNext())

{

int n = i.next();

if (!visited[n])

DFSVisit(G, n, visited);

}

}

}

public class DirectedGraph extends Graph {

public DirectedGraph(){

super();

}

public DirectedGraph(BufferedReader reader) throws IOException {

super(reader);

}

public void addEdge(int v, int w) {

super.adj[v].add(w);

}

}

public class UndirectedGraph extends Graph {

public UndirectedGraph(){

super();

}

public UndirectedGraph(BufferedReader reader) throws IOException {

super(reader);

}

public void addEdge(int v, int w) {

super.adj[v].add(w);

super.adj[w].add(v);

}

}

The output was:

0 to 250:

0, 15, 24, 39, 66, 149, 163, 202, 204, 222, 225, 211, 209, 206, 114, 176, 191, 231, 248, 232, 217, 201, 216, 95, 115, 153, 228, 241, 76, 45, 48, 50, 59, 80, 97, 144, 168, 187, 208, 226, 151, 138, 188, 230, 197, 184, 181, 196, 205, 207, 210, 212, 214, 219, 221, 156, 139, 122, 110, 108, 135, 141, 94, 198, 223, 242, 182, 170, 229, 249, 203, 200, 189, 220, 247, 190, 169, 177, 186, 161, 120, 145, 146, 218, 224, 167, 117, 140, 147, 162, 192, 243, 193, 179, 244, 246, 175, 143, 152, 85, 82, 43, 30, 70, 79, 84, 100, 103, 174, 19, 13, 129, 133, 166, 236, 178, 98, 88, 41, 81, 119, 134, 137, 183, 215, 126, 109, 91, 227, 64, 47, 29, 74, 38, 53, 56, 73, 34, 22, 17, 121, 158, 113, 90, 173, 128, 136, 159, 234, 130, 194, 164, 150, 72, 107, 69, 1, 40, 75, 89, 116, 20, 127, 61, 87, 111, 60, 63, 96, 199, 237, 25, 112, 239, 240, 233, 71, 148, 157, 125, 172, 180, 213, 235, 238, 245, 195, 154, 171, 165, 155, 142, 124, 118, 57, 65, 27, 62, 78, 77, 102, 52, 93, 160, 68, 58, 33, 23, 9, 44, 49, 185, 104, 83, 67, 55, 26, 5, 32, 4, 3, 37, 31, 21, 7, 42, 86, 51, 18, 35, 36, 12, 28, 14, 2, 101, 132, 92, 16, 54, 99, 6, 106, 123, 105, 131, 10, 11, 8, 46

**Homework Assignment:**

1. Implement the Topological Sort algorithm for directed graphs and run the algorithm on the directed graph that is provided in the tinyDG.txt file.
2. Write a report containing your observations on Breadth First Search, Depth First Search, and Topological Sorting covering such concepts as programming difficulty, running time, etc.

The following source code was written:

public class Graph {

public static void main(String[] args) throws IOException {

BufferedReader read = new BufferedReader(new FileReader("mediumG.txt"));

DirectedGraph directed = new DirectedGraph(read);

System.out.println("Breadth-First Search: ");

System.out.println("0 to " + V + ":");

long umTime = System.nanoTime();

directed.BFS(directed, 0);

System.out.println("\n" + "BFS on a directed graph with medium input took: " + (System.nanoTime() - umTime) + " nanoseconds.");

System.out.println("\n" + "Depth-First Search: ");

System.out.println("0 to " + V + ":");

long dfsTime = System.nanoTime();

directed.DFS(directed, 0);

System.out.println("\n" + "DFS on a directed graph with medium input took: " + (System.nanoTime() - dfsTime) + " nanoseconds.");

System.out.println("\n" + "Topological Sort: ");

System.out.println("0 to " + V + ":");

long sortTime = System.nanoTime();

directed.topologicalSort();

System.out.println("\n" + "Topological Sort on a directed graph with medium input took: " + (System.nanoTime() - sortTime) + " nanoseconds.");

BufferedReader r = new BufferedReader(new FileReader("tinyDG.txt"));

DirectedGraph direct = new DirectedGraph(r);

System.out.println("\n" + "Topological Sort on a directed graph with tiny input: ");

System.out.println("0 to " + V + ":");

direct.topologicalSort();

}

public static int V;

public static int E;

public LinkedList<Integer>[] adj;

public Graph()

{

V = 0;

E = 0;

}

public Graph(BufferedReader reader) throws IOException

{

String line;

line = reader.readLine();

V = Integer.parseInt(line);

line = reader.readLine();

E = Integer.parseInt(line);

adj = new LinkedList[V];

for (int v = 0; v < V; v++) {

adj[v] = new LinkedList<Integer>();

}

while ((line = reader.readLine()) != null) {

int tempV1, tempV2;

StringTokenizer st = new StringTokenizer(line, " ");

tempV1 = Integer.parseInt(st.nextToken());

tempV2 = Integer.parseInt(st.nextToken());

addEdge(tempV1, tempV2);

}

}

public void addEdge(int v, int w) {

//override

}

public String tostring()

{

String s = new String();

s = "There are "+V+" vertices and "+E+" edges\n";

for(int i=0;i<V;i++)

{

s = s+i+": ";

for(int j = 0; j<adj[i].size();j++)

{

s = s+adj[i].get(j)+" ";

}

s = s+"\n";

}

return s;

}

public void DFS(Graph G, int s) {

boolean visited[] = new boolean[G.V];

DFSVisit(G, s, visited);

}

void DFSVisit(Graph G, int s, boolean visited[]) {

visited[s] = true;

System.out.print(s+", ");

Iterator<Integer> i = G.adj[s].listIterator();

while (i.hasNext())

{

int n = i.next();

if (!visited[n])

DFSVisit(G, n, visited);

}

}

public void topological(int v, boolean visited[], Stack<Integer> stack) {

visited[v] = true;

Integer i;

Iterator<Integer> it = adj[v].iterator();

while (it.hasNext())

{

i = it.next();

if (!visited[i])

topological(i, visited, stack);

}

stack.push(new Integer(v));

}

public void topologicalSort()

{

Stack<Integer> stack = new Stack<Integer>();

boolean visited[] = new boolean[V];

for (int i = 0; i < V; i++)

visited[i] = false;

for (int i = 0; i < V; i++)

if (visited[i] == false)

topological(i, visited, stack);

while (stack.empty()==false)

System.out.print(stack.pop() + ", ");

}

public void BFS(Graph G, int s){

boolean visited[] = new boolean[G.V];

LinkedList<Integer> queue = new LinkedList<Integer>();

visited[s]=true;

queue.add(s);

while (queue.size() != 0)

{

s = queue.poll();

System.out.print(s+", ");

Iterator<Integer> i = G.adj[s].listIterator();

while (i.hasNext())

{

int n = i.next();

if (!visited[n])

{

visited[n] = true;

queue.add(n);

}

}

}

}

The output was:

Breadth-First Search:

0 to 250:

0, 15, 24, 44, 49, 58, 59, 68, 80, 97, 114, 149, 160, 163, 176, 191, 202, 204, 209, 211, 222, 225, 39, 66, 206, 93, 144, 168, 185, 231, 248, 165, 187, 226, 201, 232, 208, 171, 172, 180, 213, 216, 217, 235, 238, 245, 197, 230

BFS on a directed graph with medium input took: 1349406 nanoseconds.

Depth-First Search:

0 to 250:

0, 15, 24, 39, 66, 149, 163, 202, 204, 222, 225, 231, 248, 209, 211, 206, 80, 97, 144, 168, 187, 208, 226, 185, 201, 216, 232, 217, 160, 176, 191, 114, 49, 59, 93, 58, 68, 165, 171, 172, 180, 213, 235, 238, 245, 197, 230, 44

DFS on a directed graph with medium input took: 1113405 nanoseconds.

Topological Sort:

0 to 250:

92, 132, 69, 61, 87, 46, 38, 74, 31, 29, 47, 64, 91, 109, 126, 25, 60, 111, 63, 96, 199, 237, 22, 34, 53, 56, 73, 21, 27, 62, 90, 20, 40, 75, 89, 127, 116, 17, 13, 19, 12, 28, 113, 10, 105, 9, 23, 33, 8, 11, 30, 43, 82, 85, 7, 57, 118, 124, 142, 155, 154, 195, 65, 71, 6, 16, 54, 99, 4, 5, 32, 52, 26, 77, 102, 55, 78, 138, 151, 3, 45, 67, 112, 128, 136, 159, 239, 48, 83, 50, 104, 37, 76, 95, 115, 153, 228, 241, 2, 42, 101, 125, 148, 157, 14, 129, 18, 51, 86, 108, 135, 181, 184, 188, 233, 240, 70, 79, 110, 122, 139, 156, 196, 205, 84, 106, 131, 143, 152, 207, 210, 123, 175, 100, 133, 103, 174, 179, 212, 244, 246, 214, 219, 221, 193, 35, 94, 141, 36, 41, 121, 158, 170, 88, 182, 198, 98, 117, 178, 167, 140, 147, 166, 236, 162, 192, 243, 81, 119, 134, 137, 183, 215, 120, 161, 229, 145, 146, 218, 227, 224, 1, 72, 150, 169, 177, 186, 189, 164, 190, 107, 200, 223, 242, 203, 249, 173, 130, 234, 194, 220, 247, 0, 44, 15, 58, 68, 165, 171, 172, 197, 230, 180, 213, 235, 238, 245, 49, 93, 59, 24, 114, 39, 80, 97, 160, 176, 191, 144, 185, 201, 217, 216, 232, 168, 187, 208, 226, 66, 149, 206, 163, 202, 209, 211, 204, 231, 248, 222, 225

Topological Sort on a directed graph with medium input took: 1479533 nanoseconds.

Topological Sort on a directed graph with tiny input:

0 to 13:

7, 6, 9, 11, 10, 12, 8, 0, 5, 4, 2, 3, 1

Analysis:

In this homework assignment, we are comparing the running times of breadth-first search, depth- first search, and topological sort. Breadth- first search is an algorithm that searches a graph and finds the shortest path from s to each reachable vertex. The time complexity of breadth-first search is O(V+E) where V is the number of vertices and E is the number of edges. Depth- first search searches deeper in the graph whenever possible. The time complexity of depth-first search is O(V+E) as well. In depth- first search, we start from a vertex, we first print it and then recursively call DFS for its adjacent vertices. In topological sorting, we use a temporary stack. We don’t print the vertex immediately, we first recursively call topological sorting for all its adjacent vertices, then push it to a stack. Finally, print contents of stack. A vertex is pushed to stack only when all of its adjacent vertices (and their adjacent vertices and so on) are already in stack. The time complexity of topological sort is O(V+E) as well. It is evident in my output that the running time for breadth-first search, depth-first search, and topological sort are similar as in theory.