```java
package prog4;
/* Ali Mojarrad
 * Comp282 Mon-Wed
 * Assignment 4
 * 05/6/2015
 * BFS graph with DFS EXTRA */

import java.io.*; // for BufferedReader
import java.util.*; // for StringTokenizer

class Edge_Node {
	Vertex_Node target;
	Edge_Node next;

	public Edge_Node(Vertex_Node t, Edge_Node e) {
		target = t;
		next = e;
	}

	public Vertex_Node GetTarget() {
		return target;
	}

	public Edge_Node GetNext() {
		return next;
	}
}
class Vertex_Node {
	String name;
	Edge_Node edge_head;
	int distance;
	Vertex_Node next, parent;
	boolean visited;

	public Vertex_Node(String s, Vertex_Node v) {
		name = s;
		next = v;
		distance = -1;
		parent = null;
		visited = false;

	}

	public String GetName() {
		return name;
	}

	public int GetDistance() {
		return distance;
	}

	public void SetDistance(int d) {
		distance = d;
	}

	public Edge_Node GetNbrList() {
		return edge_head;
```

```
	}

	public Vertex_Node GetNext() {
		return next;
	}

	public Vertex_Node GetParent() {
		return parent;
	}

	public void SetParent(Vertex_Node parent) {
		this.parent = parent;
	}

	public void SetVisited(boolean visited) {
		this.visited = visited;
	}

	public boolean GetVisited() {
		return visited;
	}

}

class Graph {
	Vertex_Node head;
	int size;

	public Graph() {
		head = null;
		size = 0;
	}

	public void clearDist() {
		Vertex_Node pt = head;
		while (pt != null) {
			pt.distance = -1;
			pt = pt.next;
		}
	}
//same functionality as ClearDist for marked visitations -made for clarity
purposes
	public void clearVisits() {
		Vertex_Node pt = head;
		while (pt != null) {
			pt.SetVisited(false);
			pt = pt.next;
		}
	}

	public Vertex_Node findVertex(String s) {
		Vertex_Node pt = head;
		while (pt != null && s.compareTo(pt.name) != 0)
			pt = pt.next;
		return pt;
	}
```

```
	public Vertex_Node input(String fileName) throws IOException {
		String inputLine, sourceName, targetName;
		Vertex_Node source = null, target;
		Edge_Node e;
		StringTokenizer input;
	BufferedReader inFile = new BufferedReader(new   FileReader(fileName));
		inputLine = inFile.readLine();
		while (inputLine != null) {
			input = new StringTokenizer(inputLine);
			sourceName = input.nextToken();
			source = findVertex(sourceName);
			if (source == null) {
				head = new Vertex_Node(sourceName, head);
				source = head;
				size++;
			}
			if (input.hasMoreTokens()) {
				targetName = input.nextToken();
				target = findVertex(targetName);
				if (target == null) {
					head = new Vertex_Node(targetName, head);
					target = head;
					size++;
				}
			// put edge in one direction -- after checking for repeat
				e = source.edge_head;
				while (e != null) {
					if (e.target == target) {
		System.out.print("Multiple edges from " + source.name+ "to");
						System.out.println(target.name + ".");
						break;
					}
					e = e.next;
				}
			source.edge_head = new Edge_Node(target, source.edge_head);
				// put edge in the other direction
				e = target.edge_head;
				while (e != null) {
					if (e.target == source) {
		System.out.print("Multiple edges from " + target.name + " to ");
						System.out.println(source.name + ".");
						break;
					}
					e = e.next;
				}
				target.edge_head = new Edge_Node(source,
target.edge_head);
			}
			inputLine = inFile.readLine();
		}
		inFile.close();
		return source;
	}

	public void output() {
```

```
        Vertex_Node v = head;
        Edge_Node e;
        while (v != null) {
                System.out.print(v.name + ": ");
                e = v.edge_head;
                while (e != null) {
                        System.out.print(e.target.name + " ");
                        e = e.next;
                }
                System.out.println();
                v = v.next;
        }
}

//prints given vertex info
        public void printNode(Vertex_Node v) {
                //checks to see if parent is null and prints accordingly
                if (v.GetParent() == null)
        System.out.println(v.GetName() + " , " + v.GetDistance() + " , "
                                + v.GetParent());
                else
        System.out.println(v.GetName() + " , " + v.GetDistance() + " , "
                                + v.GetParent().GetName());
        }

public void output_bfs(Vertex_Node s) {
        //create a queue to hold given vertex - one vertex at a time
        Queue<Vertex_Node> q = new LinkedList<Vertex_Node>();
        //go through all vertex nodes from s to end
        while (s != null) {
                // if its not been visited set it as parent
                if (s.GetVisited() == false) {
                        s.SetParent(null);
                        s.SetDistance(0);
                        s.SetVisited(true);
                        //add it to q
                        q.add(s);
                        //tracks its edgelist and print it in another method
                        output_bfs(q);
                }
                s = s.GetNext();
        }

//another loop to start from the head to cover the missing vertex nodes
        s = head;
        while (s != null) {
                if (s.GetVisited() == false) {
                        s.SetParent(null);
                        s.SetDistance(0);
                        s.SetVisited(true);
                        q.add(s);
                        output_bfs(q);
                }
                s = s.GetNext();
        }
        //clear visits and distances for DFS-search of same inputs
        clearVisits();
```

```
		clearDist();

	}


	public void output_bfs(Queue<Vertex_Node> q) {
		//while q is not empty
		while (!q.isEmpty()) {
			// start from the edgehead
			Edge_Node nextE = q.peek().GetNbrList();
			//go through all edges
			while (nextE != null) {
				// if not visited before update and add to queue
				if (nextE.GetTarget().GetDistance() == -1) {
					nextE.GetTarget().SetParent(q.peek());

	nextE.GetTarget().SetDistance(q.peek().GetDistance() + 1);
					nextE.GetTarget().SetVisited(true);
					q.add(nextE.GetTarget());
				}
				nextE = nextE.GetNext();

			}
		//at the end of every edgelist check print the first in the queue
			//starting from parent
			printNode(q.remove());
		}

	}

	public void output_dfs(Vertex_Node s) {
		// create a stack
		Stack<Vertex_Node> stack = new Stack<Vertex_Node>();
		// go through all vertex nodes
		while (s != null) {
			// if not visited set info as parent
			if (s.GetVisited() == false) {
				s.SetParent(null);
				s.SetDistance(0);
				s.SetVisited(true);
				// push to stack and call helper method
				stack.push(s);
				output_dfs(stack);
			}
			s = s.GetNext();
		}

		// loop to cover the missing nodes from head to s
		s = head;
		while (s != null) {
			if (s.GetVisited() == false) {
				s.SetParent(null);
				s.SetDistance(0);
				s.SetVisited(true);

				stack.push(s);
				output_dfs(stack);
```

```
                }
                s = s.GetNext();
            }
        //clear visits and dists for assurance (if DFS is run before BFS case)
            clearVisits();
            clearDist();

        }

        public void output_dfs(Stack<Vertex_Node> stack) {
            // go through stack
            while (!stack.isEmpty()) {
                //set v to be the parent node
                Vertex_Node v = stack.peek();
                // look at its edgehead
                Edge_Node nextE = v.GetNbrList();

                //print and pop the parent
                printNode(stack.pop());
                // go through all edges
                while (nextE != null) {
                    // if not visited  update
                    if (nextE.GetTarget().GetDistance() == -1) {
                        nextE.GetTarget().SetParent(v);
                    nextE.GetTarget().SetDistance(v.GetDistance() + 1);
                        nextE.GetTarget().SetVisited(true);
                        // add to stack
                        stack.push(nextE.GetTarget());
                        // recursively call it as a parent
                        output_dfs(stack);
                    }
                    nextE = nextE.GetNext();
                }

            }
        }

}
```

**Test #1: BFS**
**=======**
a , 0 , null
e , 1 , a
b , 1 , a
i , 2 , e
f , 2 , e
l , 2 , b
c , 2 , b
j , 3 , i
g , 3 , f
k , 3 , l
h , 3 , l
d , 3 , c

**Test #1: DFS**
**=======**
a , 0 , null
e , 1 , a
i , 2 , e
j , 3 , i
k , 4 , j
l , 5 , k
b , 6 , l
c , 7 , b
g , 8 , c
h , 9 , g
d , 10 , h
f , 9 , g

**Test #2: BFS**
**=======**
Folsum , 0 , null
EchoPark , 1 , Folsum
Kobe , 1 , Folsum
Denver , 2 , EchoPark
Jasper , 2 , EchoPark
LAX , 2 , Kobe
Gothum , 3 , Denver
Colinga , 3 , Denver
Io , 3 , Jasper
Barstow , 4 , Gothum
Helena , 4 , Colinga
Albany , 5 , Barstow

**Test #2: DFS**
**=======**
Folsum , 0 , null
EchoPark , 1 , Folsum
Denver , 2 , EchoPark
Gothum , 3 , Denver
Barstow , 4 , Gothum
Colinga , 5 , Barstow
Jasper , 6 , Colinga
Kobe , 7 , Jasper
LAX , 8 , Kobe
Io , 9 , LAX
Helena , 10 , Io
Albany , 11 , Helena

**Test #3: BFS**
**=======**
a , 0 , null
e , 1 , a
b , 1 , a
f , 2 , e
i , 0 , null
j , 1 , i
k , 2 , j
l , 3 , k
d , 0 , null
c , 1 , d
g , 2 , c
h , 3 , g

**Test #3: DFS**
**=======**
a , 0 , null
e , 1 , a
b , 1 , a
f , 2 , e
i , 0 , null
j , 1 , i
k , 2 , j
l , 3 , k
d , 0 , null
c , 1 , d
g , 2 , c
h , 3 , g

**=======**
a , 0 , null
e , 1 , a
f , 2 , e
b , 3 , f
i , 0 , null
j , 1 , i
k , 2 , j
l , 3 , k
d , 0 , null
c , 1 , d
g , 2 , c
h , 3 , g

**Test #4: BFS**
=======
h , 0 , null
g , 1 , h
d , 1 , h
f , 2 , g
c , 2 , d
e , 3 , f
b , 3 , c
i , 4 , e
a , 4 , b
j , 5 , i
k , 6 , j
l , 7 , k

**Test #4: DFS**
=======
h , 0 , null
g , 1 , h
f , 2 , g
e , 3 , f
i , 4 , e
j , 5 , i
k , 6 , j
l , 7 , k
d , 1 , h
c , 2 , d
b , 3 , c
a , 4 , b

**Test #5: BFS**
=======
a , 0 , null
b , 1 , a
l , 0 , null
k , 1 , l
j , 0 , null
i , 1 , j
h , 0 , null
g , 1 , h
f , 0 , null
e , 1 , f
d , 0 , null
c , 1 , d

**Test #5: DFS**
=======
a , 0 , null
b , 1 , a
l , 0 , null
k , 1 , l
j , 0 , null
i , 1 , j
h , 0 , null
g , 1 , h
f , 0 , null
e , 1 , f
d , 0 , null
c , 1 , d

**Test #6: BFS**
=======
f , 0 , null
e , 1 , f
i , 1 , f
j , 1 , f
k , 1 , f
g , 1 , f
c , 1 , f
b , 1 , f
a , 1 , f

**Test #6: DFS**
=======
f , 0 , null
e , 1 , f
i , 2 , e
j , 3 , i
k , 4 , j
g , 5 , k
c , 6 , g
b , 7 , c
a , 8 , b

**Test #7: BFS**
=======
a , 0 , null
e , 1 , a
i , 2 , e
d , 0 , null
h , 1 , d
l , 2 , h
k , 0 , null
g , 1 , k
c , 2 , g
j , 0 , null
f , 1 , j
b , 2 , f

**Test #7: DFS**
=======
a , 0 , null
e , 1 , a
i , 2 , e
d , 0 , null
h , 1 , d
l , 2 , h
k , 0 , null
g , 1 , k
c , 2 , g
j , 0 , null
f , 1 , j
b , 2 , f

**Test #9: BFS**
=======
b , 0 , null
c , 1 , b
a , 1 , b
d , 2 , c
e , 2 , a
i , 3 , e
f , 3 , e
m , 4 , i
j , 4 , f
g , 4 , f
k , 5 , g
h , 5 , g
o , 6 , k
l , 6 , h
p , 7 , o
n , 7 , o

**Test #9: DFS**
=======
b , 0 , null
c , 1 , b
d , 2 , c
a , 1 , b
e , 2 , a
i , 3 , e
m , 4 , i
f , 3 , e
j , 4 , f
g , 4 , f
k , 5 , g
o , 6 , k
p , 7 , o
n , 7 , o
h , 5 , g
l , 6 , h

**Test #10: BFS**
=======
a , 0 , null
e , 1 , a
b , 1 , a
f , 2 , e
k , 2 , e
c , 2 , b
h , 2 , b
l , 3 , f
q , 3 , k
i , 3 , c
d , 3 , c
n , 3 , h
g , 3 , h
j , 4 , i
m , 4 , n
p , 5 , j
o , 6 , p

**Test #10: DFS**
=======
a , 0 , null
e , 1 , a
f , 2 , e
l , 3 , f
k , 4 , l
q , 5 , k
b , 1 , a
c , 2 , b
i , 3 , c
j , 4 , i
p , 5 , j
o , 6 , p
d , 5 , j
h , 2 , b
n , 3 , h
m , 4 , n
g , 5 , m