```java
// Project 13.2    Taylor Somma    CS501

/*
13.2 Modify the dfs.java program (Listing 13.1) to use adjacency lists rather than an adjacency matrix. You can obtain a list by adapting the Link and LinkList classes from the linkList2.java program (Listing 5.2) in Chapter 5. Modify the find( ) routine from LinkList to search for an unvisited vertex rather than for a key value.

Lafore, Robert (2002-11-16). Data Structures & Algorithms in Java (Kindle Locations 7318-7321). Pearson HE, Inc.. Kindle Edition.

*/
class graphNode
{
  private String label;
   private adjacencyList neighborList;
  public int numNeighbors;
  private boolean isVisited;

  public graphNode(String newLabel)
  {
    label=newLabel;
    numNeighbors=0;
    isVisited=false;
    neighborList = new adjacencyList();
  }

  public boolean isUnvisitedNeighbors()
  {
    return neighborList.isUnvisitedNeighbors();
  }

  public void setVisited(boolean isVisitedIn)
  {
    isVisited=isVisitedIn;
  }

  public boolean isVisited()
  {
    return isVisited;
  }

  public adjacencyList getNeighborList()
  {
    return neighborList;
  }

  public void printNeighborList()
  {
    neighborList.setFirstNodeAsCurrentNode();
    for (int x=0;x<neighborList.returnNumNeighbors();x++)
    {
      System.out.print(neighborList.getCurrentNode().getLabel()+", ");
      neighborList.getNextNode();
    }
  }

  public graphNode getNextUnvisited()
  {
    neighborList.setFirstNodeAsCurrentNode();
```

```
57      for (int x=0;x<neighborList.returnNumNeighbors();x++)
58      {
59        System.out.println("gNU at "+neighborList.getCurrentNode().getGraphNode().getLabel());
60        if (neighborList.getCurrentNode().getGraphNode().isVisited()==false)
61        {
62          return neighborList.getCurrentNode().getGraphNode();
63        }
64        neighborList.getNextNode();
65      }
66      return null;
67    }
68    public int returnNumNeighbors()
69    {
70      return numNeighbors;
71    }
72
73    public void addNeighbor(graphNode newNeighbor)
74    {
75      neighborList.addNode(newNeighbor);
76      newNeighbor.getNeighborList().addNode(this);
77      newNeighbor.numNeighbors++;
78      numNeighbors++;
79    }
80
81    public String getLabel()
82    {
83      return label;
84    }
85  }
86
87  class graphListNode
88  {
89    private graphNode graphNodeValue;
90    private graphListNode nextNode;
91    private graphListNode previousNode;
92
93    public graphListNode(graphNode newNode)
94    {
95      graphNodeValue = newNode;
96      nextNode=null;
97      previousNode=null;
98    }
99
100   public String getLabel()
101   {
102     return graphNodeValue.getLabel();
103   }
104
105   public graphNode getGraphNode()
106   {
107     return graphNodeValue;
108   }
109
110   public boolean isVisited()
111   {
112     return graphNodeValue.isVisited();
113   }
114
115   public void setAsVisited(boolean isVisitedIn)
116   {
117     graphNodeValue.setVisited(true);
```

```
118   }
119
120   public void setNext(graphListNode neighborNode)
121   {
122     nextNode = neighborNode;
123   }
124
125   public void setPrevious(graphListNode neighborNode)
126   {
127     previousNode = neighborNode;
128   }
129
130   public graphListNode getNext()
131   {
132     return nextNode;
133   }
134
135   public graphListNode getPrevious()
136   {
137     return previousNode;
138   }
139 }
140
141
142 class adjacencyList
143 {
144   private graphListNode firstNode,currentNode;
145   private int numNeighbors;
146
147   public adjacencyList()
148   {
149     firstNode=null;
150     numNeighbors=0;
151   }
152   // Calling this function for an adjacency list belonging to a node adds the new node as a
    neighbor
153   public void addNode(graphNode newNode)
154   {
155     graphListNode newListNode = new graphListNode(newNode);
156     if (firstNode==null)
157     {
158       firstNode=newListNode;
159       currentNode=newListNode;
160       numNeighbors++;
161     }
162     else
163     {
164       while (currentNode.getNext()!=null)
165       {
166         currentNode=currentNode.getNext();
167       }
168       newListNode.setPrevious(currentNode);
169       currentNode.setNext(newListNode);
170       currentNode=currentNode.getNext();
171       numNeighbors++;
172     }
173   }
174
175   public int returnNumNeighbors()
176   {
177     return numNeighbors;
```

```
178   }
179
180   public void setFirstNodeAsCurrentNode()
181   {
182     currentNode=firstNode;
183   }
184
185   public boolean isUnvisitedNeighbors()
186   {
187     currentNode = firstNode;
188     while (currentNode!=null)
189     {
190       if (currentNode.isVisited()==false)
191       {
192         return true;
193       }
194       currentNode=currentNode.getNext();
195     }
196     return false;
197   }
198
199   public boolean isEmpty()
200   {
201     if (firstNode==null)
202       return true;
203     else
204       return false;
205   }
206
207   public graphListNode returnFirstNode()
208   {
209     return firstNode;
210   }
211
212   public graphListNode getCurrentNode()
213   {
214     return currentNode;
215   }
216
217   //Helper Function for printing list of neighbors
218   public void getNextNode()
219   {
220     currentNode=currentNode.getNext();
221   }
222
223   public graphListNode getLastNode()
224   {
225     graphListNode currentNode = firstNode;
226     while(currentNode.getNext()!=null)  //May need to add && currentNode!=null
227       currentNode=currentNode.getNext();
228     return currentNode;
229   }
230
231   public void visitNextNeighbor()
232   {
233     currentNode.getGraphNode().setVisited(true);
234     currentNode=currentNode.getNext();
235   }
236
237   public graphListNode getFirstNode()
238   {
```

```
239     return firstNode;
240   }
241 }
242
243 class dfsStack
244 {
245   private graphNode[] graphNodeStack = new graphNode[100];
246   private int firstNodeIndex,lastNodeIndex,currentNodeIndex, stackLength;
247
248   public dfsStack()
249   {
250     graphNodeStack[0]=null;
251     lastNodeIndex=0;
252     firstNodeIndex=0;
253     stackLength=0;
254   }
255
256   public void addNode(graphNode newNode)
257   {
258     graphNodeStack[lastNodeIndex]=newNode;
259     lastNodeIndex++;
260     stackLength++;
261   }
262
263   public boolean isEmpty()
264   {
265     if (stackLength<=0)
266       return true;
267     else
268       return false;
269   }
270
271   public graphNode popNode()
272   {
273     if (stackLength!=0)
274     {
275       graphNode returnNode = graphNodeStack[lastNodeIndex-1];
276       stackLength--;
277       lastNodeIndex--;
278       System.out.println("Popping node: "+returnNode.getLabel());
279       return returnNode;
280     }
281     else
282     {
283       System.out.println("The stack is empty");
284       return null;
285     }
286   }
287
288   public graphNode peakNode()
289   {
290     if (stackLength!=0)
291     {
292       graphNode returnNode = graphNodeStack[lastNodeIndex-1];
293       return returnNode;
294     }
295     else
296     {
297       System.out.println("The stack is empty");
298       return null;
299     }
```

```
300   }
301 }
302
303 public class depthFirstSearch
304 {
305
306   public static void dfs(graphNode nodeIn, dfsStack stackIn)
307   {
308     if (nodeIn==null)
309     {
310       return;
311     }
312     stackIn.addNode(nodeIn);
313     nodeIn.setVisited(true);
314     while (stackIn.isEmpty()==false)
315     {
316       //System.out.println("Entering while loop");
317       if (nodeIn.isUnvisitedNeighbors()==false)
318       {
319         //System.out.println("Found NO unvisited nodes");
320         stackIn.popNode();
321       }
322       else
323       {
324         //System.out.println("Found unvisited for "+nodeIn.getLabel());
325         graphNode visitingNode = nodeIn.getNextUnvisited();
326         dfs(visitingNode,stackIn);
327       }
328       //System.out.println("In while loop ");
329     }
330     //System.out.println("End of dfs");
331     //System.out.println("Returning due to empty stack");
332     return;
333   }
334   public static void main(String[] args)
335   {
336     //Graph node contains all nodes in graph
337     //GraphListNode is the data structure that represents each node in an
338     //adjacency list. The graphListNode contains a graphNode as its primary value. The
    adjacencyList data structure has a graphListNode as its first,last and curent data points.
    Each graphListNode then has a previous and next node which can be used to traverse the
    adjacency list. Get neighbors by calling the adjacecylist.first and iterate through next.
339
340     //Sample text Boston,NY,Chicago,LA,Miami,Houston, Honolulu
341
342     System.out.println("Test setup");
343     graphNode gn1 = new graphNode("Boston");
344     graphNode gn2 = new graphNode("New York");
345     graphNode gn3 = new graphNode("Chicago");
346     graphNode gn4 = new graphNode("Los Angeles");
347     graphNode gn5 = new graphNode("Miami");
348     graphNode gn6 = new graphNode("Houston");
349     graphNode gn7 = new graphNode("Honolulu");
350
351     dfsStack stackForDFS = new dfsStack();
352
353     gn1.addNeighbor(gn2);
354     System.out.println(gn1.isUnvisitedNeighbors());
355     System.out.println(gn3.isUnvisitedNeighbors());
356
357     gn1.addNeighbor(gn3);
```

```
358     gn1.addNeighbor(gn4);
359     gn2.addNeighbor(gn5);
360     gn2.addNeighbor(gn6);
361     gn4.addNeighbor(gn7);
362     System.out.println("GN1 has: "+gn1.returnNumNeighbors()+" Neighbors");
363     System.out.println("GN2 has: "+gn2.returnNumNeighbors()+" Neighbors");
364     System.out.println("GN3 has: "+gn3.returnNumNeighbors()+" Neighbors");
365     System.out.println("GN4 has: "+gn4.returnNumNeighbors()+" Neighbors");
366     System.out.println("GN5 has: "+gn5.returnNumNeighbors()+" Neighbors");
367     System.out.println("GN6 has: "+gn6.returnNumNeighbors()+" Neighbors");
368     System.out.println("GN7 has: "+gn7.returnNumNeighbors()+" Neighbors");
369     System.out.println(gn5.isUnvisitedNeighbors());
370 
371     System.out.println(gn1.getLabel()+" Neighbors Are: ");
372     gn1.printNeighborList();
373     System.out.println("\n");
374     System.out.println(gn2.getLabel()+" Neighbors Are: ");
375     gn2.printNeighborList();
376     System.out.println("\n");
377     System.out.println(gn3.getLabel()+" Neighbors Are: ");
378     gn3.printNeighborList();
379     System.out.println("\n");
380     System.out.println(gn4.getLabel()+" Neighbors Are: ");
381     gn4.printNeighborList();
382     System.out.println("\n");
383     System.out.println(gn5.getLabel()+" Neighbors Are: ");
384     gn5.printNeighborList();
385     System.out.println("\n");
386     System.out.println(gn6.getLabel()+" Neighbors Are: ");
387     gn6.printNeighborList();
388     System.out.println("\n");
389 
390     dfs(gn1,stackForDFS);
391 
392 
393   }
394 }
```