CSCE 221 Cover Page

Programming Assignment #6

First Name: Adrian Last Name: Gamez-Rodriguez UIN:126009409

User Name: adriangamez E-mail address: adriangamez@tamu.edu

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources, you can get a lower number of points or even zero. According to the University Regulations, Section 42, scholastic dishonesty are including: acquiring answers from any unauthorized source, working with another person when not specifically permitted, observing the work of other students during any exam, providing answers when not specifically authorized to do so, informing any person of the contents of an exam prior to the exam, and failing to credit sources used. Disciplinary actions range from grade penalties to expulsion, read more: Aggie Honor System Office

Type of Sources			
People	TA and PT in Lab		
Web Pages	Geeksforgeeks.org	Stackoverflow.com	
Printed Material	Data Structures and Algorithms	Lectures	

I certify that I have listed all the sources that I used to develop the solutions/codes to the submitted work.

"On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work"

Adrian Gamez-Rodriguez Date:4/28/2019

Description of data structure:

To implement the Graph data structure, I used an adjacency list to represent the graph. The container I used was a vector of lists. Each index of the vector corresponded to a numbered vertex. At each index of the vector there are list that contain the adjacent vertices. Some functions that help utilize the data structure is addEdge(), removeEdge(), findStartingVertex(), canBeDrawnInOneStroke(), printPath(), reachability(), and validEdge(). I also made getters and setter to read in the graph from data files.

The functions mentioned above all help with finding a suitable solution to the problem of finding a Euler path or Euler circuit. The addEdge(), adds edge to the graph, while the removeEdge() function "removes" an edge from the adjacency list. First to be able to determine if a solution is even possible I call the function canBeDrawnInOneStroke(). This function traverses through all the list and determines the number of odd vertices and the number of even vertices. If the number of odd vertices is not equal to 2 or 0 then no solution is possible.

Then the function findStartingVertex() finds the starting vertex by having a vector of all the odd vertices, it picks the first odd vertex. If there are no odd vertices then it will just pick 1 as the starting vertex. The rest of the functions are used collectively to determine a path, which I will discuss in the description algorithm and running time section.

Determine for which group of pictures there are always solutions:

There are always solutions to graphs that have either 2 or 0 odd degree vertices. Thus the pictures 1, 2, 4, 5 have solutions.

Determine for which group of pictures there are no solutions.

There are no solutions to graphs that have more than 2 odd degree vertices or one odd degree vertex. This leads to the pictures 3, and 6 not having any solutions.

Determine for which group of pictures there exist solutions starting at one point and ending at the same point. What kind of point could be selected as the start/end point in such a case?

This solution is called a Euler Circuit. Meaning that all the vertices have even degree and start and end at the same vertex. The kind of vertex to be selected is an even vertex.

Determine for which group of pictures there exist solutions starting at one point and ending at a different point. What kind of points could be selected as the start/end points in such a case?

This solution is called a Euler Path. Meaning the solution starts and ends at different vertices, and those start, and end vertices should be the only 2 odd vertices.

Necessary and Sufficient Conditions:

The graph must either have 2 or 0 odd vertices, anything else then a solution does not exist.

Description of algorithm and running time:

The function printPath(), is the algorithm that determines the path if the possible. I call printPath() to print the path starting with the predetermined starting vertex. However, I only call the function if I know there is a solution. We traverse all adjacent vertices of the starting vertex using iterators, if there is only one adjacent vertex, I immediately consider it. If there are more than one adjacent vertices in the list, then we have to check if the edge (u,v) is a bridge or not. A "bridge" is an edge that allows us to come back to that vertex. The idea is to no take the bridge because then we won't be able to come back. We determine if an edge is a bridge or not by counting the number of vertices reachable from u. We remove edge (u,v) and again count the number of reachable vertices from u. If number of reachable vertices are reduced, then edge u-v is a bridge. To see whether all vertices are reachable from u, I try a DFS traversal from vertex u. The function reachability(u) returns number of vertices reachable from u. Finally, when an edge is used, it is removed from the graph. To remove the edge, I replace the corresponding entry in the adjacency list with 0. Since the function is recursive I didn't want the call to crash if the size of the adjacency list changed.

The time complexity of this algorithm depends on two things, the number of vertices and the number of edges in the graph. First the printPath(), is recursive and it searches the adjacent vertices of the starting vertex, it also calls the validEdge() function which also calls the reachability() function which in itself does a DFS, to find reachable vertices. The DFS of a graph represented by an adjacency list is O(V+E). So I believe the total time complexity of the algorithm is O(V+E).

Evidence of testing:

Graph 1:

```
[adriangamez]@linux2 ~/CSCE221/Assignment6> (21:06:05 04/28/19)
:: ./main graph1.data
Data file entered: graph1.data
Corresponding output:
AdjMatrix:
0 1 1 0 0
1 0 1 1 1
1 1 0 0 1
0 1 0 0 1
0 1 1 1 0

AdjList:
1 -> 2 -> 3
2 -> 1 -> 3 -> 4 -> 5
3 -> 1 -> 2 -> 5
4 -> 2 -> 5
5 -> 2 -> 3 -> 4

Edges:
(1,2)(2,3)(3,1)(2,4)(4,5)(5,2)(5,3)
Number of Odd Vertices: 0
Number of Even Vertices: 0
Can this graph be drawn in one stroke? (0= no, 1= yes): 1
Starting Vertex : 3
Path:
(3, 1) (1, 2) (2, 3) (3, 5) (5, 2) (2, 4) (4, 5)
[adriangamez]@linux2 ~/CSCE221/Assignment6> (21:06:07 04/28/19)
```

Graph 2:

```
[adriangamez]@linux2 ~/CSCE221/Assignment6> (21:06:07 04/28/19)
:: ./main graph2.data
Data file entered: graph2.data
Corresponding output:
AdjMatrix:
0 1 1 0 0
1 0 1 1 1
1 1 0 0 0
0 1 0 0 1
0 1 0 1 0

AdjList:
1 -> 2 -> 3
2 -> 1 -> 3 -> 4 -> 5
3 -> 1 -> 2
4 -> 2 -> 5
5 -> 2 -> 4

Edges:
(1,2)(1,3)(2,3)(2,4)(2,5)(4,5)
Number of Odd Vertices: 0
Number of Even Vertices: 0
Can this graph be drawn in one stroke? (0= no, 1= yes): 1
Starting Vertex : 1
Path:
(1, 2) (2, 4) (4, 5) (5, 2) (2, 3) (3, 1)
[adriangamez]@linux2 ~/CSCE221/Assignment6> (21:06:49 04/28/19)
```

Graph 3:

```
[adriangamez]@linux2 ~/CSCE221/Assignment6> (21:07:23 04/28/19)
:: ./main graph3.data
Data file entered: graph3.data
Corresponding output:
AdjMatrix:
0 1 1 1 0 0 0 0 0 0 0 0
1 0 0 0 1 1 0 0 0 0 0 0
1 0 0 1 0 0 1 0 0 0 0 0
1 0 1 0 0 0 0 0 1 0 0 0
0 1 0 0 0 1 0 0 0 0 0 0
0 1 0 0 1 0 0 0 0 1 0 0
0 0 1 0 0 0 0 1 0 0 1 0
0 0 0 0 0 0 1 0 0 0 1 0
0 0 0 1 0 0 0 0 0 1 0 1
0 0 0 0 0 1 0 0 1 0 0 1
0 0 0 0 0 0 1 1 0 0 0 1
0 0 0 0 0 0 0 0 1 1 1 0

AdjList:
1 -> 2 -> 3 -> 4
2 -> 1 -> 5 -> 6
3 -> 1 -> 4 -> 7
4 -> 1 -> 3 -> 9
5 -> 2 -> 6
6 -> 2 -> 5 -> 10
7 -> 3 -> 8 -> 11
8 -> 7 -> 11
9 -> 4 -> 10 -> 12
10 -> 6 -> 9 -> 12
11 -> 7 -> 8 -> 12
12 -> 9 -> 10 -> 11

Edges:
(1,2)(1,3)(1,4)(2,5)(2,6)(3,4)(3,7)(4,9)(5,6)(6,10)(7,8)(7,11)(8,11)(9,10)(9,12)(10,12)(11,12
)
Number of Odd Vertices: 0
Number of Even Vertices: 0
Can this graph be drawn in one stroke? (0= no, 1= yes): 0
No path found.

[adriangamez]@linux2 ~/CSCE221/Assignment6> (21:07:27 04/28/19)
```

Graph 4:

```
[adriangamez]@linux2 ~/CSCE221/Assignment6> (21:08:19 04/28/19)
:: ./main graph4.data
Data file entered: graph4.data
Corresponding output:
AdjMatrix:
0 1 1 0 0 0 0
1 0 0 1 0 0 1
1 0 0 1 1 1 0
0 1 1 0 1 1 0
0 0 1 1 0 1 0
0 0 1 1 1 0 1
0 1 0 0 0 1 0

AdjList:
1 -> 2 -> 3
2 -> 1 -> 4 -> 7
3 -> 1 -> 4 -> 5 -> 6
4 -> 2 -> 3 -> 5 -> 6
5 -> 3 -> 4 -> 6
6 -> 3 -> 4 -> 5 -> 7
7 -> 2 -> 6

Edges:
(1,2)(1,3)(2,4)(2,7)(3,4)(3,5)(3,6)(4,5)(4,6)(5,6)(6,7)
Number of Odd Vertices: 0
Number of Even Vertices: 0
Can this graph be drawn in one stroke? (0= no, 1= yes): 1
Starting Vertex : 2
Path:
(2, 1) (1, 3) (3, 4) (4, 2) (2, 7) (7, 6) (6, 3) (3, 5) (5, 4) (4, 6) (6, 5)
[adriangamez]@linux2 ~/CSCE221/Assignment6> (21:08:23 04/28/19)
```

Graph 5:

```
[adriangamez]@linux2 ~/CSCE221/Assignment6> (21:08:53 04/28/19)
:: ./main graph5.data
Data file entered: graph5.data
Corresponding output:
AdjMatrix:
0 1 1 0 0
1 0 1 1 1
1 1 0 0 0
0 1 0 0 1
0 1 0 1 0

AdjList:
1 -> 2 -> 3
2 -> 1 -> 3 -> 4 -> 5
3 -> 1 -> 2
4 -> 2 -> 5
5 -> 2 -> 4

Edges:
(1,2)(2,3)(3,1)(2,4)(4,5)(5,2)
Number of Odd Vertices: 0
Number of Even Vertices: 0
Can this graph be drawn in one stroke? (0= no, 1= yes): 1
Starting Vertex : 1
Path:
(1, 2) (2, 4) (4, 5) (5, 2) (2, 3) (3, 1)
[adriangamez]@linux2 ~/CSCE221/Assignment6> (21:08:59 04/28/19)
```

Graph 6:

```
[adriangamez]@linux2 ~/CSCE221/Assignment6> (21:09:37 04/28/19)
:: ./main graph6.data
Data file entered: graph6.data
Corresponding output:
AdjMatrix:
0 1 1 1 0
1 0 1 1 1
1 1 0 0 1
1 1 0 0 1
0 1 1 1 0

AdjList:
1 -> 2 -> 3 -> 4
2 -> 1 -> 3 -> 4 -> 5
3 -> 1 -> 2 -> 5
4 -> 1 -> 2 -> 5
5 -> 2 -> 3 -> 4

Edges:
(1,2)(1,3)(1,4)(2,3)(2,4)(2,5)(3,5)(4,5)
Number of Odd Vertices: 0
Number of Even Vertices: 0
Can this graph be drawn in one stroke? (0= no, 1= yes): 0
No path found.

[adriangamez]@linux2 ~/CSCE221/Assignment6> (21:09:42 04/28/19)
```