CSCE 221 Cover Page
Homework Assignment

First Name: Kirsten Last Name: Madina UIN: 626003641

User Name: kirsten.madina E-mail address: kirsten.madina@gmail.com

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, read more in the Aggie Honor System Office http://aggiehonor.tamu.edu/

Type of sources					
People	Posts on Piazza				
Web pages (provide URL)	Stack Overflow	Cplusplus.com	GeeksforGeeks		
Printed material	textbook				
Other Sources	N/A				

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

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

Your Name (signature) Kirsten Madina Date 04/28/2019

1 Description of Data Structures

The data structure implemented in this program was a Graph using an adjacency matrix built with a vector of arrays of integers. The class Graph has 3 private data members and 8 public member functions. The data members include **int** vertices, which is equal to the number of vertices, **int** edges, which is equal to the number of edges, and **vector<int>** adj[100], which is a vector of 100 integer arrays used as the adjacency matrix.

The functions of this data structure include **int** get$_v ertices$, **int**$get_e dges()$, $textbf voidaddEdge(inta, intb)$, **void**$print()$; **boo**

2 Necessary and Sufficient conditions for drawing one-stroke pictures

I used a Euler algorithm to deterine whether or not a graph can be drawn in one stroke. The Euler algorithm checks to see if the specific graph has more than two vertices with an odd number of edges. In my implementation of this data structure, the Euler algorithm is implemented in the draw_in_one_line() boolean function. This algorithm uses a for look to check how many odd vertices are on each vertex, then uses the conditions previously stated to return either true or false. Under these conditions, given graphs 3 and 6 cannot be done using one stroke.

3 Description of Algorithms and Run Times

INSERT: The insert function is implemented through the **addEdge** function, which inputs to integers and and b. The value of 'b' is then pushed back into adj[a] whilst the value of a is pushed back into adj[b]. For this reason, the insert function is constant time or *O(1)*.

BUILD: Building a graph is implemented in the main using data from an input file. The first two numbers in the file correspond to the number of vertices and number of edges and are called in the graph constructor. The rest of the numbers in the input file are called two at a time using a for loop and are then inputs for the addEdge function. The for loop increments one at a time, while the addEdge function is constant, and therefore Building a graph is **O(n)**.

DELETE: Deleting an edge from a graph is implemented using the function delEdge(int a, int b) which inputs two integers corresponding to vertices with the goal of removing the edge between them. In my implementation of this function, this is done using the std::remove() function included in the <algorithms> header. This remove function takes in to iterators corresponding to the range in which the value removed can be found as well as the value being removed. An iterator then iteratates through the vector until that value is found and then removes it. The pop$_b ackfunctioniscalledoncethevalueisremoved.Thisprovessiscalledtwiceforthearrayatbandthearra$
$removehasaruntimeofO(n)andpop_b ack()isconstant, theruntimeformydelEdgefunctionisO(e1+e2), orsimplyO(n).$

SEARCH : $Searchingforapathinthisimplementationiscalledin theSearch(inti)function.Thisisarecursivelycalledfunctio$

4 Testing