CSE 331 - Project 5
Unweighted Graphs

TA: Sorrachai Yingchareonthawornchai
`yingchar@msu.edu`
Due Date: 11:59 pm Nov 14, 2014

1 Project Description

In this project, you will complete implementation of an efficient algorithm to find Euler Circuits in an unweighted graph. You have been given code to implement a program that will load graph_input.txt file, construct a graph and perform operations related to some properties of the graph. In addition, you have been given a good deal of latitude in the implementation. Make sure that you **understand** algorithms before implementing them.
Let V, E be set of vertices and edges respectively. Your job is to complete the implementation of the following methods in UnweightedGraph class:

1. IsConnected: return true if the graph is connected. Return false otherwise. The running time should be $O(|V| + |E|)$.

2. IsEulerian: return true if the graph is Eulerian. Otherwise, return false. A graph is Eulerian if and only if it contains Euler cycle. Euler cycle is a cycle that visits every edge exactly once. The running time should be $O(|V|)$. You can assume in this function that the graph is connected.

3. FindEulerCycle: return an euler cycle as represented in a list containing the order in which each vertex is visited, and returns the starting vertex again at the end. An example of Euler cycle is given in Figure 1. In this function, you can assume that the graph is connected and Eulerian. The running time should be $O(|V| + |E|)$.

4. VerifyEulerCycle: return true if an input list is a valid Euler cycle. Return false otherwise. This running time should be $O(|V| + |E|)$. **DO NOT** implement this function by calling FindEulerCycle. It is meaningless to do so. Test carefully for this function, i.e., you have to verify your VerifyEulerCycle carefully.

For this project you may use anything from the STL. I recommend using STL list to store your path, that way you can easily use the splice() function to merge paths together. STL list is also a doubly linked list, which means that you can erase items from it in constant time (using iterators), so it may also be ideal for storing adjacency lists. We have provided you with a few graphs to test your projects with, but make sure to come up with your own test cases!

2 Input Test Cases

The program will take one input which is a file contianing exactly n lines. n lines are numbers seperated by space. They are adjacency lists. Your program will then read file and create a graph accordingly.
Your program will build a graph based on the input file. Then, your program will check if the graph is an Eulerian graph. If so, it will find Eulerian cycle and verify the result. As usual, main.cpp does the job for you. You have to complete implementation of UnweightedGraph class. Note: use the CSE black server as a running machine. Other test cases will be executed after the due date for grading.

3 Project Deliverables

The following files must be submitted via Handin no later than 11:59 pm Nov 14, 2014.

1. UnweightedGraph.h - contains your class prototype of unweighted graph.
2. UnweightedGraph.cpp - contains your implementation of unweighted graph class.

4 Written Questions

There is no written question.

5 Hints

The following hints are not the only ways to implement. You may find alternative way to implement, but make sure that your implementation is correct and efficient.

1. IsConnected: you may use depth-first-search (DFS) or breadth-first-search (BFS); the important part is you need to perform DFS or BFS only once using arbitrary vertex as an initial vertex. The graph is connected if and only if all vertices are visited.
2. IsEulerian: when constructing a graph, store degree seqeuence into another vector. A graph is Eulerian if and only if all degrees of vertices are even.

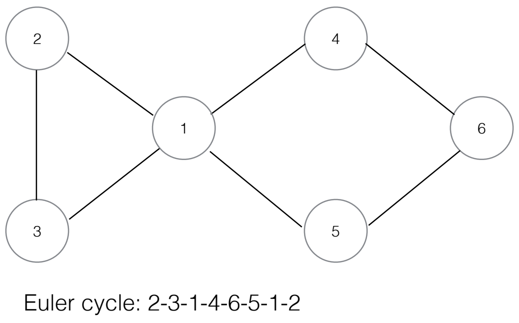

Figure 1: An example of Euler cycle in a graph.

3. findEulerCycle:
    (a) Make sure you understand the algorithm, see tutorial file in the website.
    (b) You may need to use additional appropriate data structure to make this function efficient. For example, when visiting an edge and you want to mark this edge as visited edge. It is inefficient for adjacency list since you have to traverse two times(why two?).
    (c) STL list is useful for this function, especially for splicing paths. You must avoid repetitious scanning of adjacency lists. This can be done by having a pointer to last used edges for each vertex since you need to visit each edge only once.
    (d) Once a path is spliced in, you have to find a new vertex that has unvisited edge. This new vertex can be found in the current cycle. You can store a pointer to the start of the splice point so that you do not have to scan from the beginning. If none of them has unvisited edges, you are done.
4. VerifyEulerCycle: correctness of this function is the most important since findEulerCycle's correctness is based on this function for the testing purpose.