```
CS 6301.002.  Implementation of advanced data structures and algorithms
Spring 2016;  Wed, Jan 27.
Long Project 0: Finding Euler tours

Ver 1.0: Initial description (Jan 27, 1:00 PM).
Ver 1.1: Added Euler path output (Jan 28, 11:00 AM).
Ver 1.2: 2nd deadline added (Feb 1, 10 AM).
Ver 1.3: added Vertex start to verifyTour for convenience.
Ver 1.4: changed 2nd deadline to Mar 13 (from Mar 6).
Ver 2.0: (Mar 4) Challenge for 2nd deadline (worth EC = 0.2):
         Write code that is clean, object-oriented, well documented, that
         solves the problem in O(E) time, with the following restrictions:
         use only basic types, and Java's LinkedList class and its iterator,
         no recursion, no indexing, no stacks/queues/deques, no hashing,
         no custom implementation of lists or simulating them with arrays.

This is an OPTIONAL project.  Max excellence credits: 1.0

Due: 11:59 PM, Sun, Feb 21 (1st deadline),  Sun, Mar 13 (2nd deadline).
```

Project Description

A graph G is called Eulerian if it is connected and the degree of every vertex is an even number. It is known that such graphs aways have a tour (a cycle that may not be simple) that goes through every edge of the graph exactly once. Such a tour (sometimes called a circuit) is called an Euler tour. If the graph is connected, and it has exactly 2 nodes of odd degree, then it has an Euler Path connecting these two nodes, that includes all the edges of the graph exactly once. In this project, write code that finds an Euler tour or an Euler Path in a given graph. Initial code base: Edge.java, Vertex.java, Graph.java, CheckBipartiteNew.java.

The algorithm that you need to implement is known as Hierholzer's algorithm, and you can get additional details from https://en.wikipedia.org/wiki/Eulerian_path. Your algorithm should run in O(|E|) time. Your program must include the following 2 functions:

```
static List<Edge> findEulerTour(Graph g)  // Return an Euler tour of g
static boolean verifyTour(Graph g, List<Edge> tour, Vertex start)  // verify tour is a valid Euler tour
```

In addition, write a driver program that reads the input graph, finds a tour in it and prints out the tour using the format given below.

Input format

If the program is started with a command line parameter, then args[0] is the name of a file from which input is read. Otherwise, read the input from the standard input (console). Use Graph.readGraph(in, false) to read the input graph. You can assume that the graph is simple. No error handling is required to check for input format errors.

Output format

The output must be printed to standard output (console). If the graph has no Euler tour or Euler path, your program outputs the message "Graph is not Eulerian" by executing the line:

```
System.out.println("Graph is not Eulerian");
```

Otherwise, if the graph has an Euler tour, then it prints the edges of an Euler tour, in correct order, and the tour must start at node 1. If the graph has only an Euler path, then it prints the edges of an Euler path, in correct order, starting at the smaller numbered node of odd degree. Use the following code for printing output:

```
for(Edge e: tour) { System.out.println(e); }
```

Do not print anything else in the output. Note that a graph can have many different Euler tours. Your program needs to find just one of the solutions.

Sample input

```
6 10
1 2 1
1 3 1
1 4 1
1 6 1
2 3 1
3 6 1
```

```
3 4 1
4 5 1
4 6 1
5 6 1
```

Sample output

```
(1,2)
(2,3)
(3,6)
(4,6)
(4,5)
(5,6)
(1,6)
(1,3)
(3,4)
(1,4)
```

Explanation

The actual tour is 1->2->3->6->4->5->6->1->3->4->1. If the edge (1,4) did not exist, then the graph has an Euler path between 1 and 4, and the output is same as above, except for the last line is not there. In this case, the algorithm outputs the path starting at node 1, which is the smaller node of 1 and 4. Additional input files will be provided later.

Instructions for submitting project.