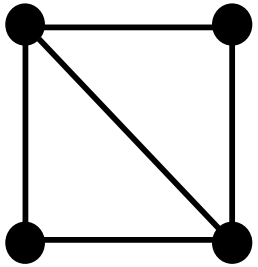
CSE 331 Project 3
Eulerian Cycles

TA: Yongqi Han

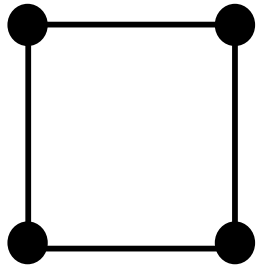
Office Hours: Thursday 6:20pm-7:50pm & Friday 4:00pm-6:00pm

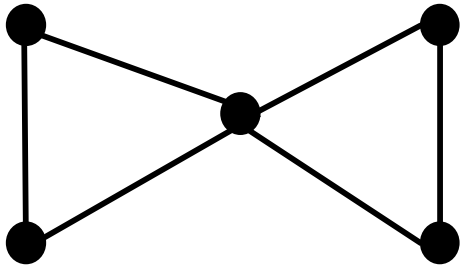
What is an Eulerian Cycle?

- Eulerian Cycle: A cycle traversing all the edges of the graph exactly once
- Eulerian Path: A path (not a cycle) traversing all the edges of the graph exactly once

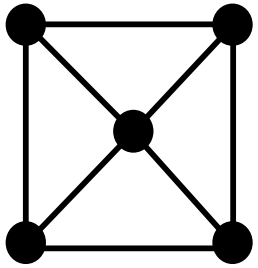


Has an Eulerian Path





Has an Eulerian Cycle



Has no Eulerian Path
or Cycle

Project 3 Specifications

- The goal of project 3 is to find Eulerian cycles in graphs
- Starter code is provided in Mimir with empty function stubs
  - IsConnected()
  - IsEulerian()
  - FindEulerianCycle()
  - VerifyEulerCycle()
  - You can add any additional functions that you need to solve the problem
- The starter code completely handles reading in the input files and creating the adjacency list of the graph
- The input files for the visible test cases can be seen on piazza or in the output of the mimir test cases

IsConnected()

- In order for a graph to contain an Euler cycle, it must be a connected graph
- The IsConnected function returns true if the graph is connected, and false otherwise
- It is recommended that you implement a depth or breadth first search to determine if the graph is connected

IsEulerian()

- For a graph to contain an Euler cycle it must be “Eulerian”
- For a graph to be Eulerian, each node of the graph must have an even degree
- IsEulerian returns true if the graph is Eulerian, and false otherwise
- As a note, this function does not find the actual Euler cycle, it only determines if the graph contains an Euler cycle
- You can assume that any graph input to this function is connected

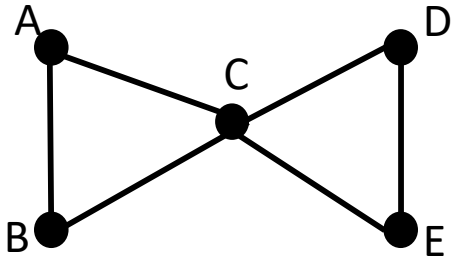
FindEulerianCycle()

- This function finds an Eulerian cycle in the graph
  - It only needs to find one Eulerian cycle, as in many cases there are multiple in a single graph
- FindEulerianCycle returns a list of ints of the discovered Euler cycle
- You can assume that any graph input into this function is Eulerian

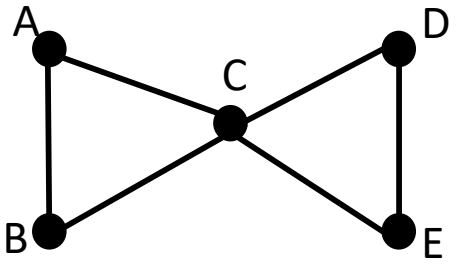
FindEulerianCycle()

- Algorithm description:
  1. Start at any node, then take any edge to visit a neighboring node
  2. Mark the edge traversed as visited
  3. Continue traversing the graph by choosing any unvisited edge until you return to the starting node, completing a cycle
  4. Once you have completed a cycle, create a new subgraph that contains any edges from the original graph that have not been visited
  5. Find any node that is present in both the current cycle and the subgraph of unvisited edges
  6. Start at the common node and continue traversing edges until you complete a new cycle
  7. Splice the new cycle into the original cycle
  8. Repeat steps 5-7 until all edges have been visited
- Example will be shown on the next slide

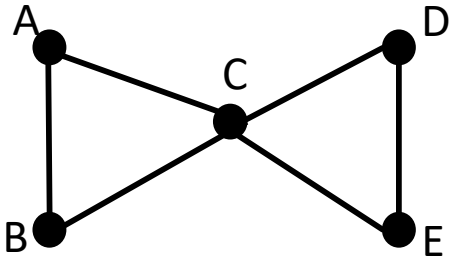
Algorithm Demonstration Part 1



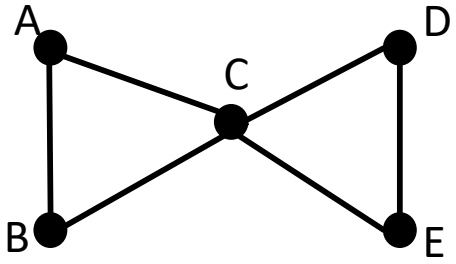
A



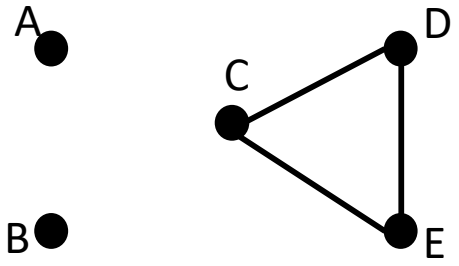
A – C



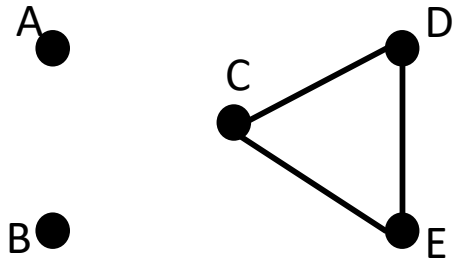
A – C – B



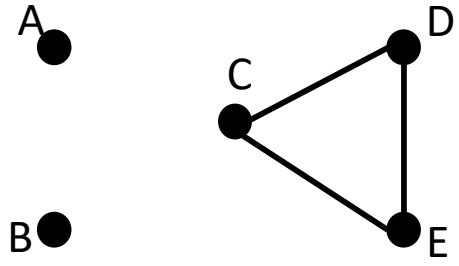
A – C – B – A



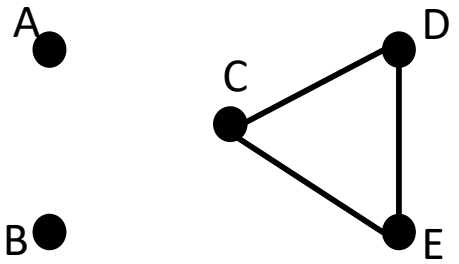
C



C – D

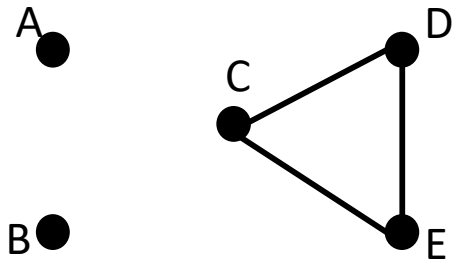


C – D – E

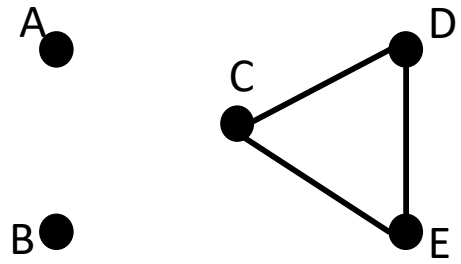


C – D – E – C

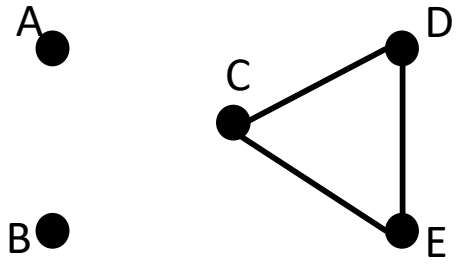
Algorithm Description Part 2



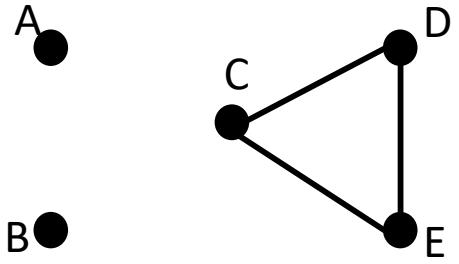
C



C – D



C – D – E



C – D – E – C

Original Cycle was: A – C – B – A, the new cycle is: C – D – E – C, we now splice the second cycle into the first to get: A – C – D – E – C – B – A, which is a valid Eulerian cycle

For this graph the algorithm ends here, however, if there were any unvisited edges remaining we would continue repeating these steps until all the edges are visited

VerifyEulerCycle()

- This function takes a cycle as input, and returns true if the path is an Euler cycle, and false otherwise
- For a cycle to be an Eulerian cycle, it must visit every edge exactly once and end at the starting node
- There is no guarantee that the graph is connected or Eulerian for this function

Questions?