Eulerian Tour

Sample Problem: Riding The Fences

Farmer John owns a large number of fences, which he must periodically check for integrity. Farmer John keeps track of his fences by maintaining a list of their intersection points, along with the fences which end at each point. Each fence has two end points, each at an intersection point, although the intersection point may be the end point of only a single fence. Of course, more than two fences might share an endpoint.

Given the fence layout, calculate if there is a way for Farmer John to ride his horse to all of his fences without riding along a fence more than once. Farmer John can start and end anywhere, but cannot cut across his fields (the only way he can travel between intersection points is along a fence). If there is a way, find one way.

The Abstraction

Given: An undirected graph

Find a path which uses every edge exactly once. This is called an Eulerian tour. If the path begins and ends at the same vertex, it is called a Eulerian circuit.

The Algorithm

Detecting whether a graph has an Eulerian tour or circuit is actually easy; two different rules apply.

- A graph has an Eulerian circuit if and only if it is connected (once you throw out all nodes of degree 0) and every node has `even degree'.
- A graph has an Eulerian path if and only if it is connected and every node except two has even degree.
- In the second case, one of the two nodes which has odd degree must be the start node, while the other is the end node.

The basic idea of the algorithm is to start at some node the graph and determine a circuit back to that same node. Now, as the circuit is added (in reverse order, as it turns out), the algorithm ensures that all the edges of all the nodes along that path have been used. If there is some node along that path which has an edge that has not been used, then the algorithm finds a circuit starting at that node which uses that edge and splices this new circuit into the current one. This continues until all the edges of every node in the original circuit have been used, which, since the graph is connected, implies that all the edges have been used, so the resulting circuit is Eulerian.

More formally, to determine a Eulerian circuit of a graph which has one, pick a starting node and recurse on it. At each recursive step:

- Pick a starting node and recurse on that node. At each step:

- If the node has no neighbors, then append the node to the circuit and return
- If the node has a neighbor, then make a list of the neighbors and process them (which includes deleting them from the list of nodes on which to work) until the node has no more neighbors
- To process a node, delete the edge between the current node and its neighbor, recurse on the neighbor, and postpend the current node to the circuit.

And here's the pseudocode:

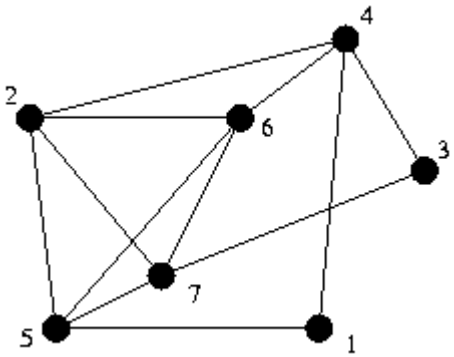
```
# circuit is a global array
   find_euler_circuit
     circuitpos = 0
     find_circuit(node 1)

# nextnode and visited is a local array
# the path will be found in reverse order
  find_circuit(node i)

    if node i has no neighbors then
      circuit(circuitpos) = node i
      circuitpos = circuitpos + 1
    else
      while (node i has neighbors)
          pick a random neighbor node j of node i
          delete_edges (node j, node i)
          find_circuit (node j)
      circuit(circuitpos) = node i
      circuitpos = circuitpos + 1
```

To find an Eulerian tour, simply find one of the nodes which has odd degree and call `find_circuit` with it.

Both of these algorithms run in O(m + n) time, where m is the number of edges and n is the number of nodes, if you store the graph in adjacency list form. With larger graphs, there's a danger of overflowing the run-time stack, so you might have to use your own stack.

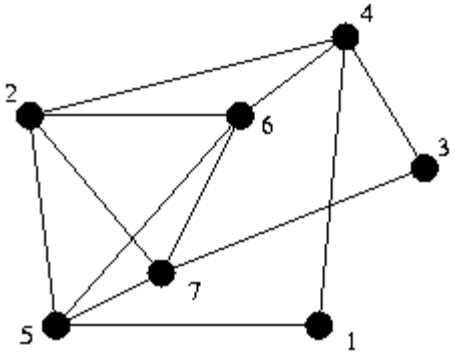
Execution Example

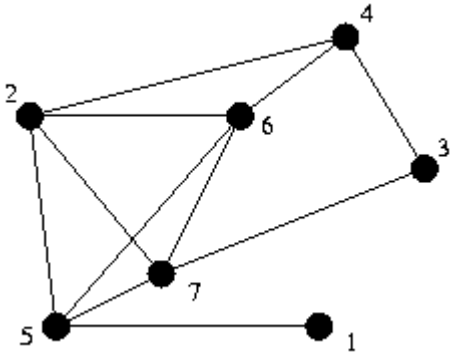
Consider the following graph:



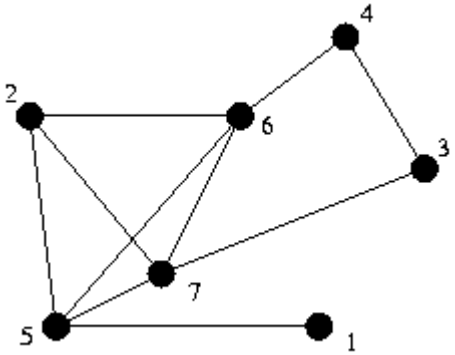
Assume that selecting a random neighbor yields the lowest numbered neighbor, the execution goes as follows:

Stack:
Location: 1
Circuit:

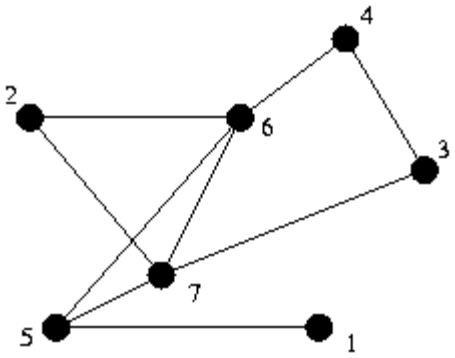




Stack: 1
Location: 4
Circuit:

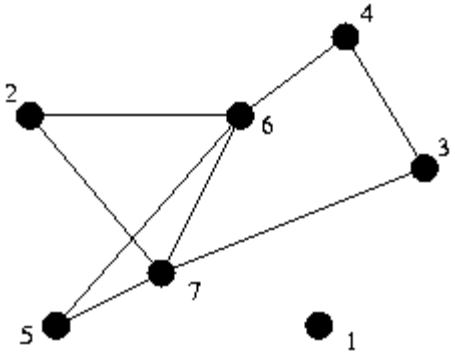


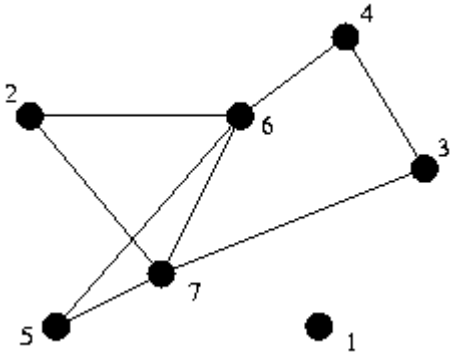
Stack: 1 4
Location: 2
Circuit:



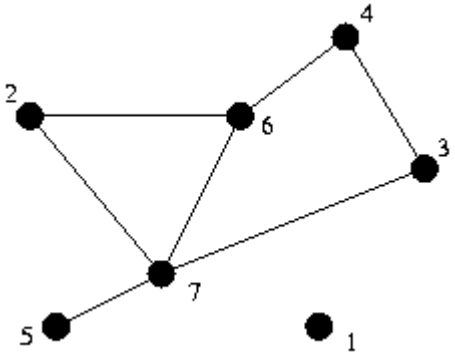
Stack: 1 4 2
Location: 5
Circuit:

Stack: 1 4 2 5
Location: 1
Circuit:

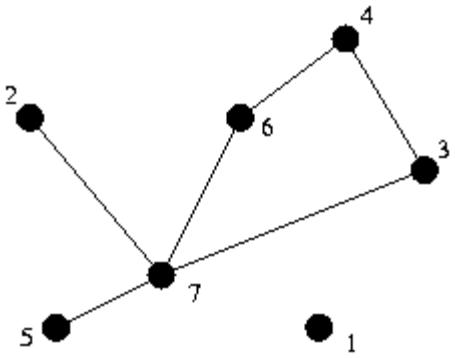




Stack: 1 4 2
Location: 5
Circuit: 1

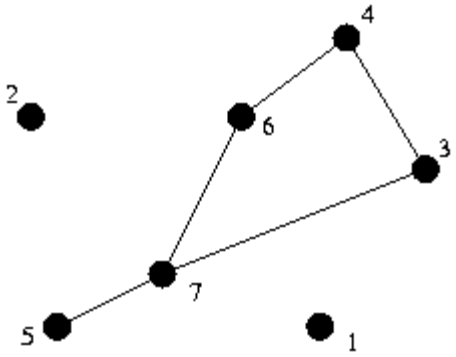


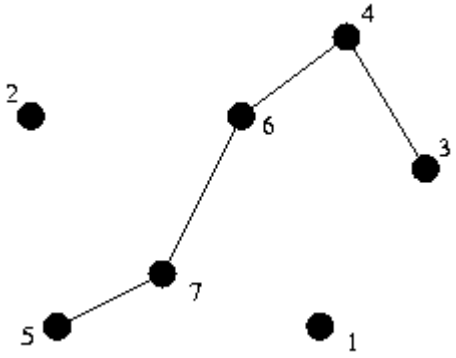
Stack: 1 4 2 5
Location: 6
Circuit: 1



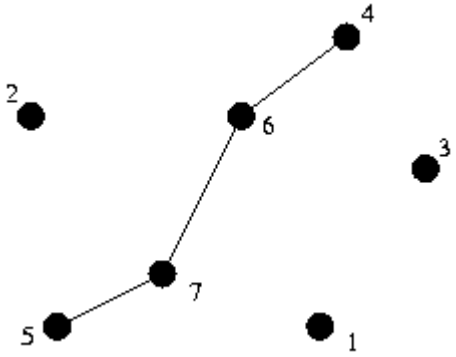
Stack: 1 4 2 5 6
Location: 2
Circuit: 1

Stack: 1 4 2 5 6 2
Location: 7
Circuit: 1

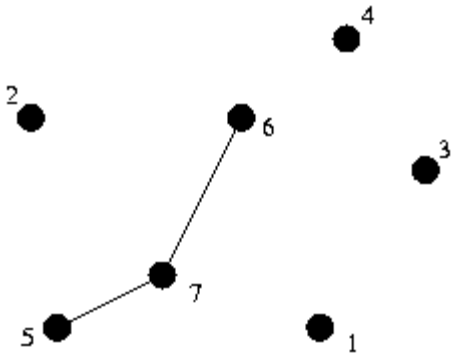




Stack: 1 4 2 5 6 2 7
Location: 3
Circuit: 1

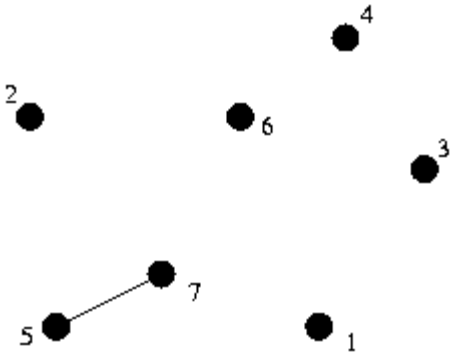


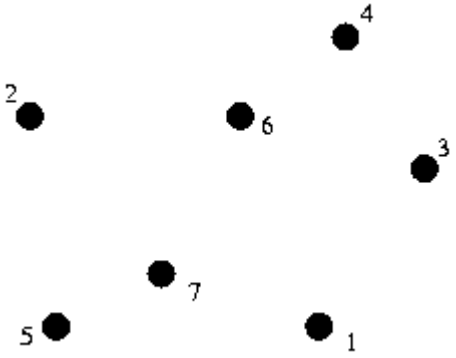
Stack: 1 4 2 5 6 2 7 3
Location: 4
Circuit: 1



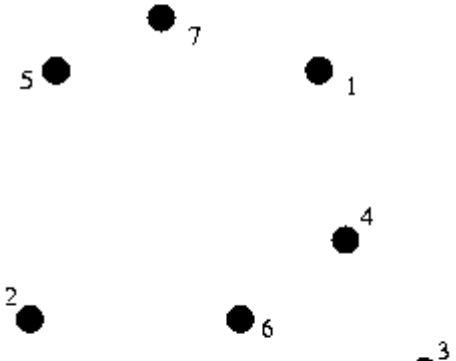
Stack: 1 4 2 5 6 2 7 3 4
Location: 6
Circuit: 1

Stack: 1 4 2 5 6 2 7 3 4 6
Location: 7
Circuit: 1

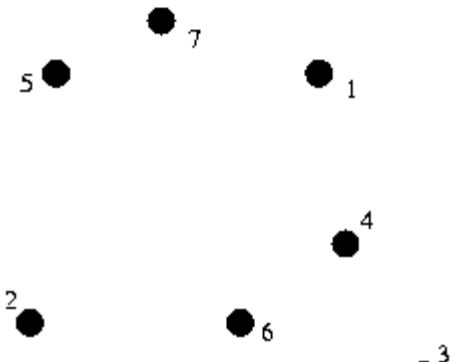




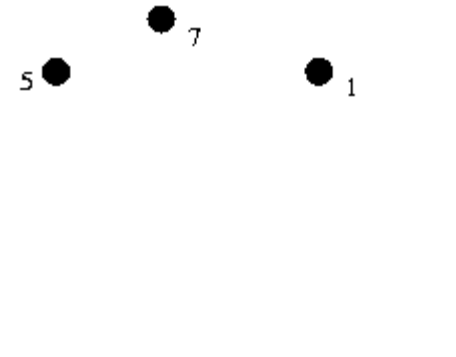
Stack: 1 4 2 5 6 2 7 3 4 6 7
Location: 5
Circuit: 1



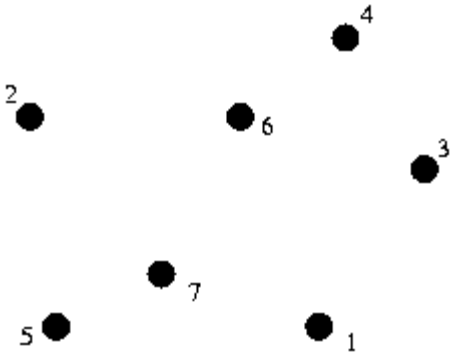
Stack: 1 4 2 5 6 2 7 3 4 6
Location: 7
Circuit: 1 5

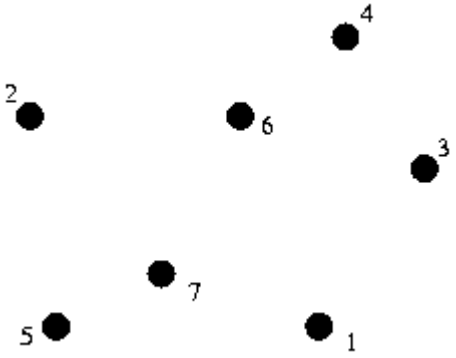


Stack: 1 4 2 5 6 2 7 3 4
Location: 6
Circuit: 1 5 7

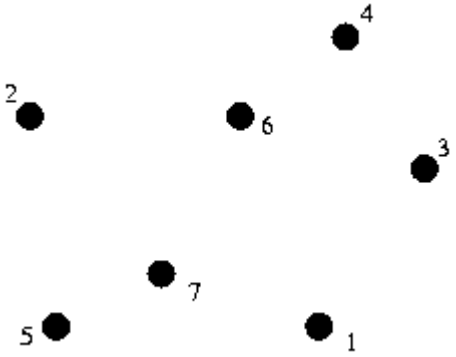


Stack: 1 4 2 5 6 2 7 3
Location: 4
Circuit: 1 5 7 6

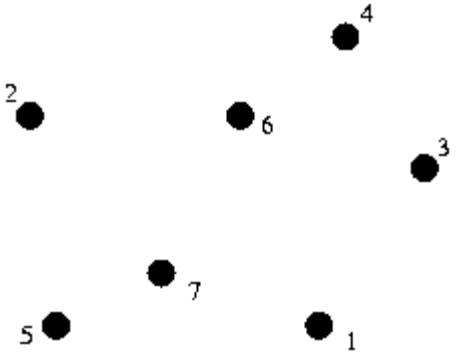




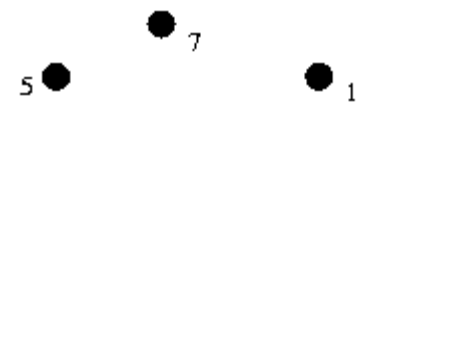
Stack: 1 4 2 5 6 2 7
Location: 3
Circuit: 1 5 7 6 4



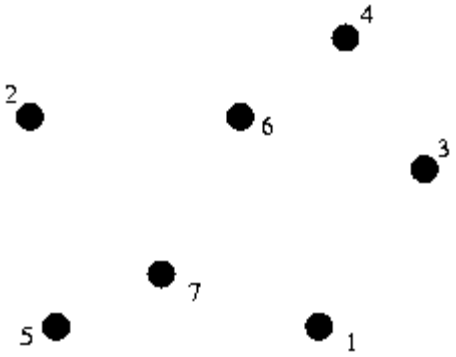
Stack: 1 4 2 5 6 2
Location: 7
Circuit: 1 5 7 6 4 3

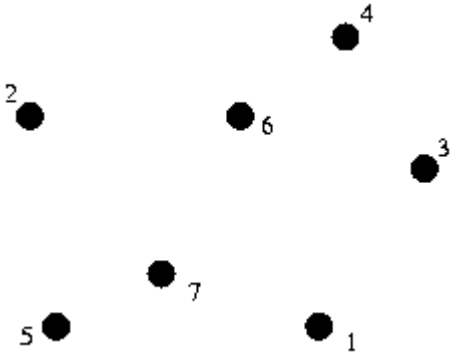


Stack: 1 4 2 5 6
Location: 2
Circuit: 1 5 7 6 4 3 7

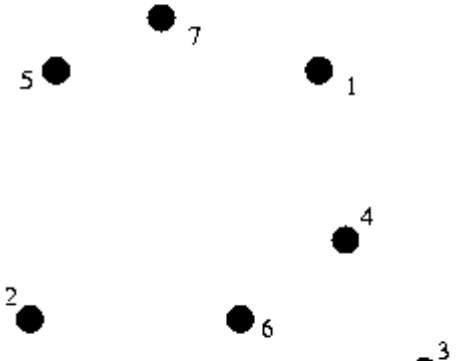


Stack: 1 4 2 5
Location: 6
Circuit: 1 5 7 6 4 3 7 2

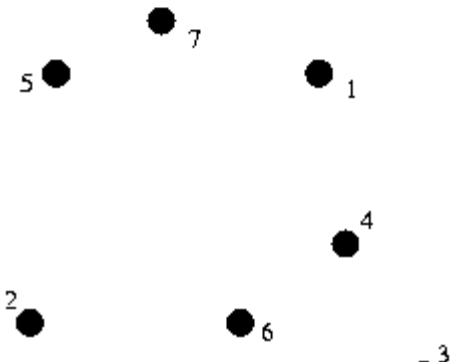




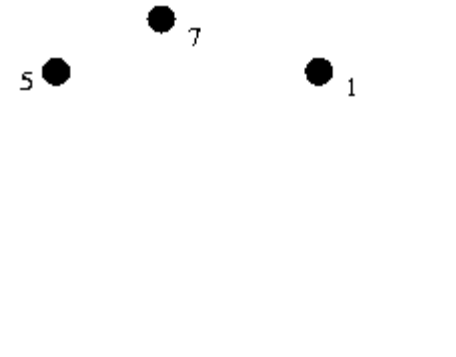
Stack: 1 4 2
Location: 5
Circuit: 1 5 7 6 4 3 7 2 6



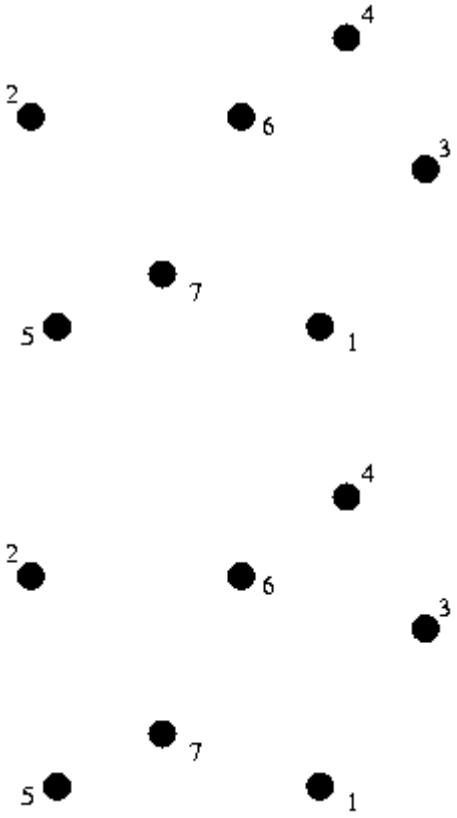
Stack: 1 4
Location: 2
Circuit: 1 5 7 6 4 3 7 2 6 5



Stack: 1
Location: 4
Circuit: 1 5 7 6 4 3 7 2 6 5 2



Stack:
Location: 1
Circuit: 1 5 7 6 4 3 7 2 6 5 2 4



Stack:
Location:
Circuit: 1 5 7 6 4 3 7 2 6 5 2 4 1

Extensions

Multiple edges between nodes can be handled by the exact same algorithm.

Self-loops can be handled by the exact same algorithm as well, if self-loops are considered to add 2 (one in and one out) to the degree of a node.

A directed graph has a Eulerian circuit if it is strongly connected (except for nodes with both in-degree and out-degree of 0) and the indegree of each node equals its outdegree. The algorithm is exactly the same, except that because of the way this code finds the cycle, you must traverse arcs in reverse order.

Finding a Eulerian path in a directed graph is harder. Consult Sedgewick if you are interested.

Example problems

Airplane Hopping

Given a collection of cities, along with the flights between those cities, determine if there is a sequence of flights such that you take every flight exactly once, and end up at the place you started.

Analysis: This is equivalent to finding a Eulerian circuit in a directed graph.

Cows on Parade

Farmer John has two types of cows: black Angus and white Jerseys. While marching 19 of their cows to market the other day, John's wife Farmeress Joanne, noticed that all 16

possibilities of four successive black and white cows (e.g., bbbb, bbbw, bbwb, bbww, ..., wwww) were present. Of course, some of the combinations overlapped others.

Given N (2 <= N <= 15), find the minimum length sequence of cows such that every combination of N successive black and white cows occurs in that sequence.

Analysis: The vertices of the graph are the possibilities of N-1 cows. Being at a node corresponds to the last N-1 cows matching the node in color. That is, for N = 4, if the last 3 cows were *wbw*, then you are at the *wbw* node. Each node has out-degree of 2, corresponding to adding a black or white cow to the end of the sequence. In addition, each node has in-degree of 2, corresponding to whether the cow just before the last N-1 cows is black or white.

The graph is strongly connected, and the in-degree of each node equals its out-degree, so the graph has a Eulerian circuit.

The sequence corresponding to the Eulerian circuit is the sequence of N-1 cows of the first node in the circuit, followed by cows corresponding to the color of the edge.

Back to USACO Training Gateway | Comment or Question