Eulerian tours

What is eulerian circuits
and paths and how to find it

3rd training camp 2008
Schalk-Willem Krüger

Paths and circuits

- Hamiltonian Circuit:
  Starts at a *vertex*, passes through every *vertex* once, and returns to the starting *vertex*
- Eulerian Path:
  Go through every edge once – starts and ends on different vertex
- Eulerian Circuit:
  Go through every edge once – starts and ends on same vertex

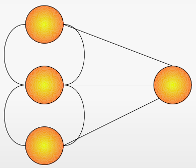
Sample problem

- 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.

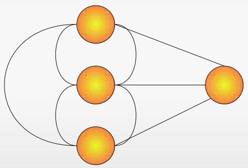
- This is equivalent to finding a Eulerian circuit in a directed graph. The cities is the vertices and the flight the edges.

Euler Paths and Circuit algorithms

- At least one Eulerian circuit exists in a graph with no vertices of odd degree
- NO Eulerian circuit but at least one Eulerian path exists in a graph with 2 vertices of odd degree
- NO Eulerian circuit exists in a graph with more than two vertices of odd degree
- Eulerian paths and circuits only exists at connected graphs



This graph can't have an Eulerian path



This graph can have an Eulerian path

The algorithm

- Pick a starting node and recurse on that node. At each step:
  - If the node has no neighbours, append the node to the circuit and return.
  - If the node has a neighbour, make a list of the neighbours 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 neighbour, and postpend the current node to the circuit.

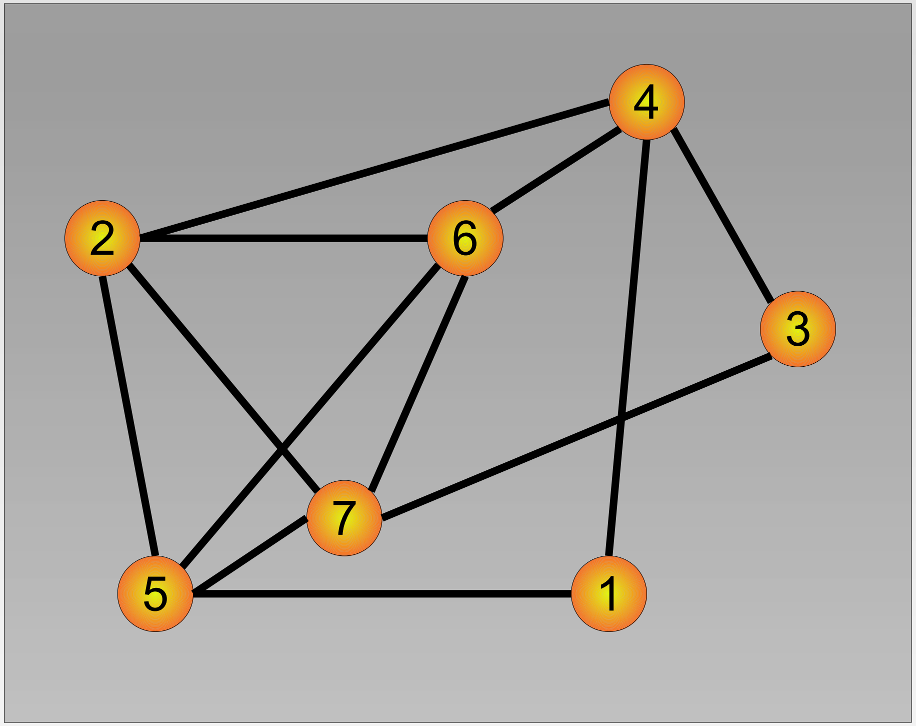
The algorithm - pseudocode

Find node with odd degree and run `find_circuit` with it.

```
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
```

Runtime: If you store in adjacency list form: O(m + n)
m = number of edges
n = number of nodes
Use a stack for larger graphs

Sample run



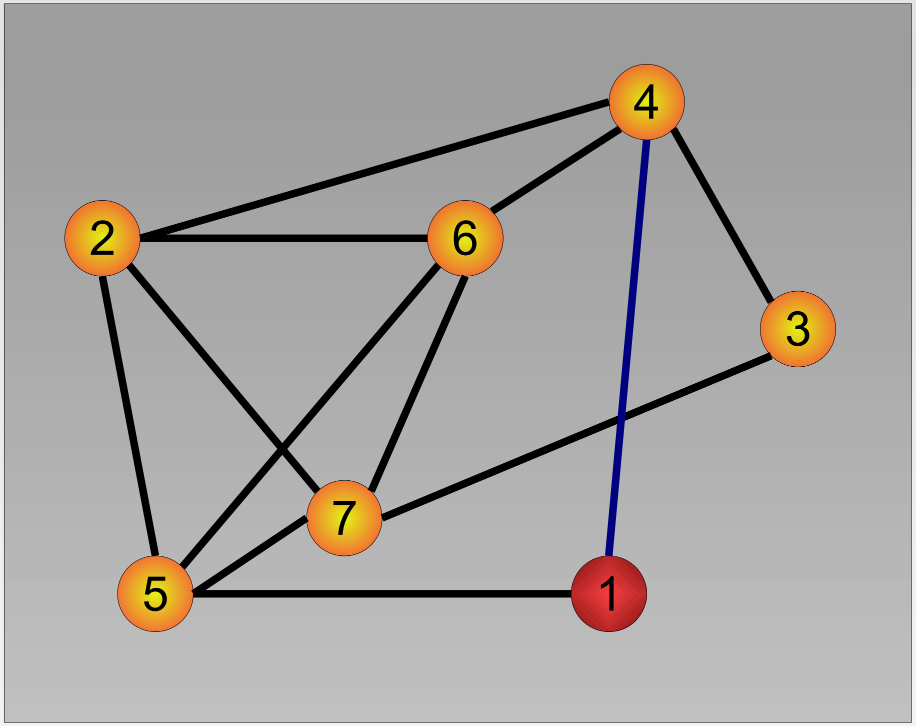
Stack:
Can be runtime stack of recursive function

Current location: current node

Circuit:
Path of Eulerian circuit found so far

- Find eulerian circuit

Sample run



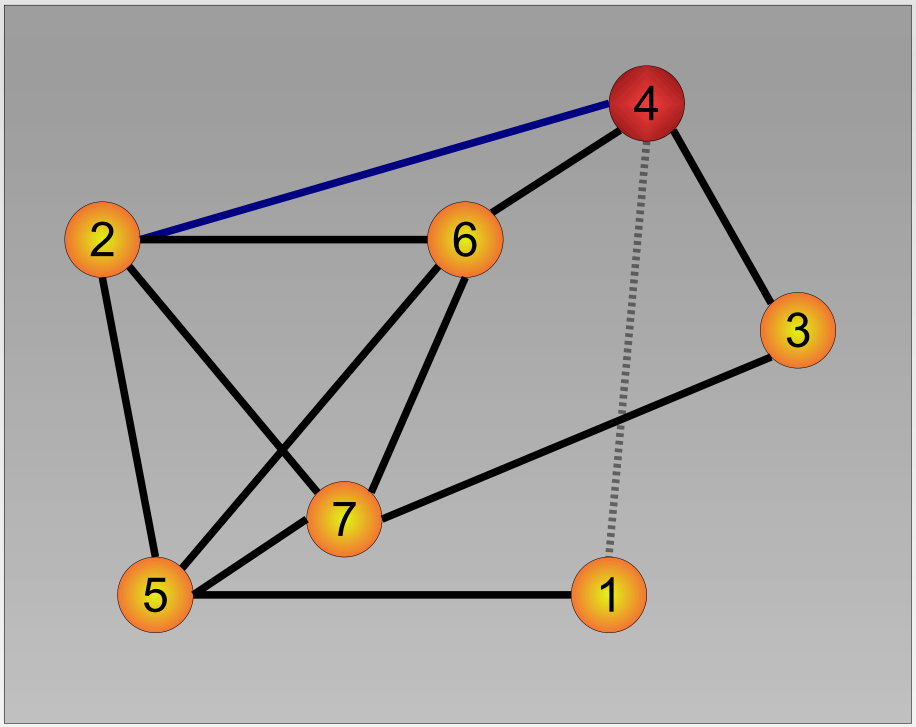
Stack:

Current location: 1

Circuit:

- Select node to begin with: Node 1
- Process first neigbour of node 1: Node 4

Sample run



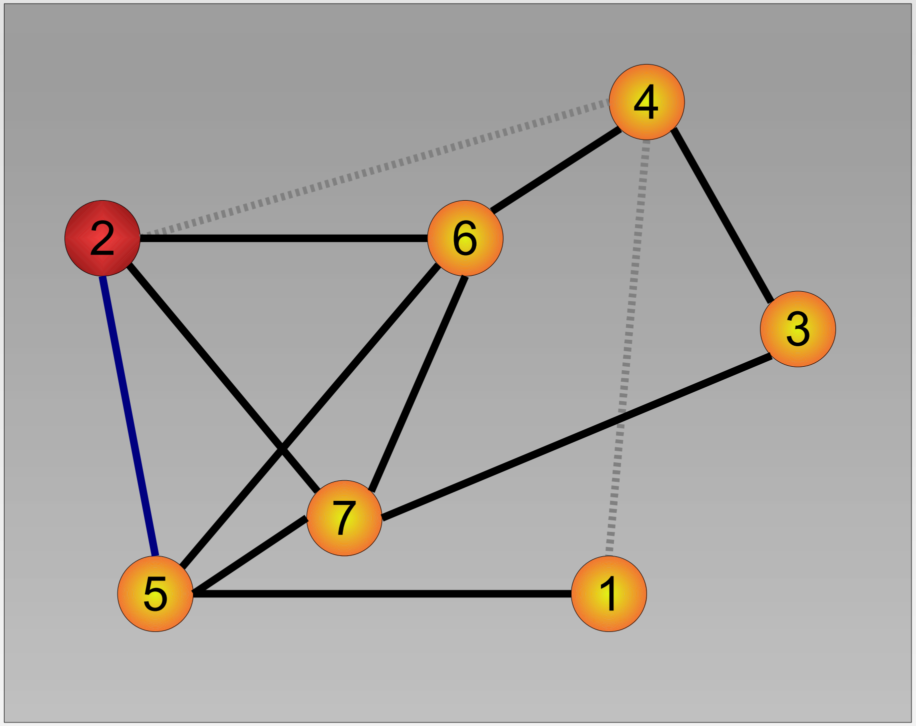
Stack: 1

Current location: 4

Circuit:

- Process node 4 – neigbour of node 1
- Delete edge between node 4 and node 1
- Process first neigbour of node 4: Node 2

Sample run



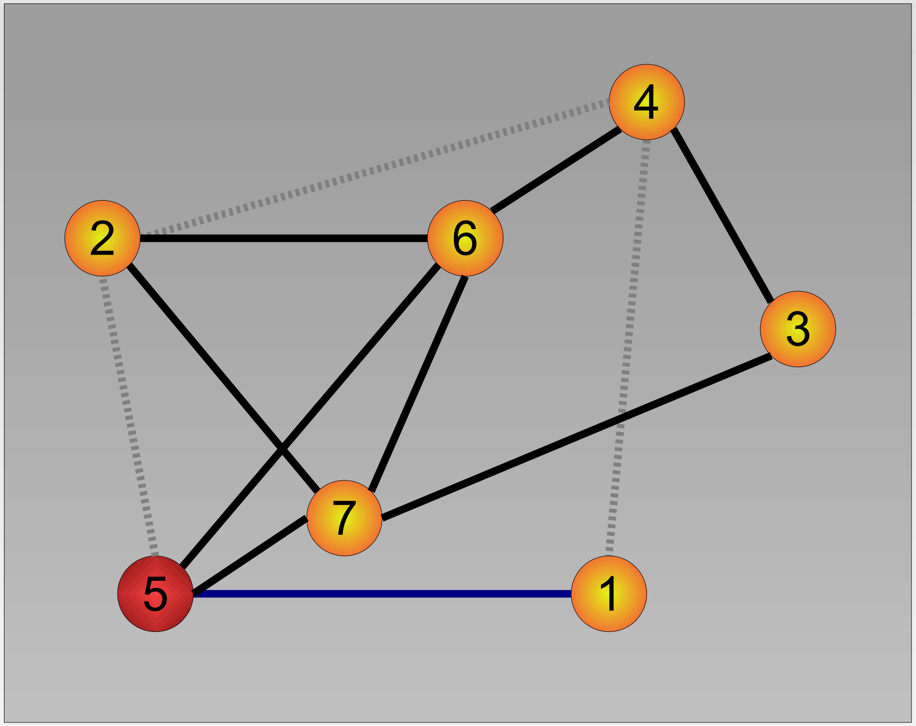
Stack: 1 4

Current location: 2

Circuit:

- Process node 2 – neigbour of node 4
- Delete edge between node 2 and node 4
- Process first neigbour of node 2: Node 5

Sample run



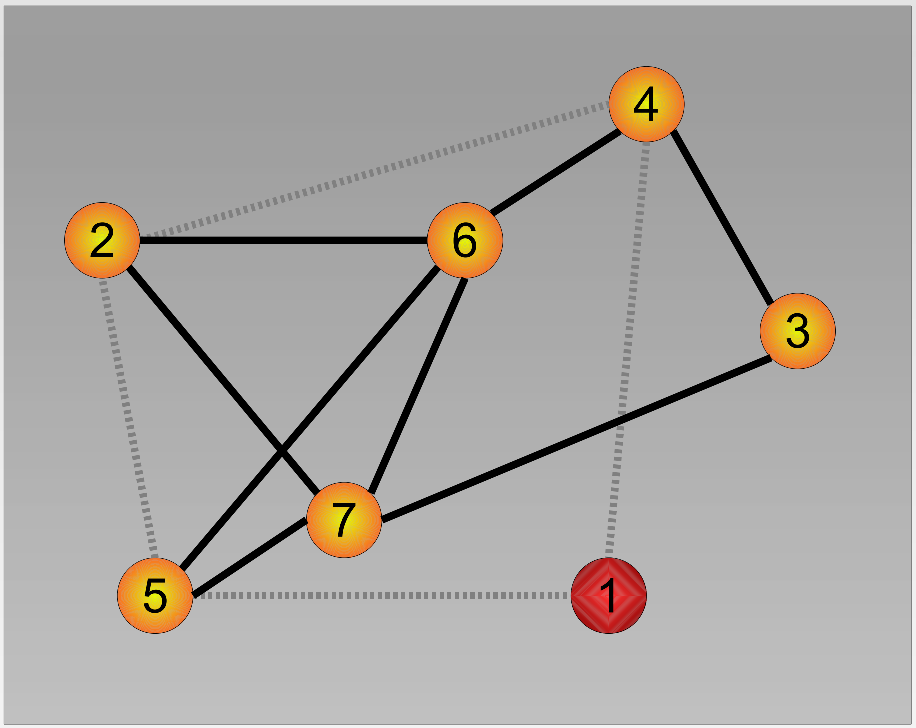
Stack: 1 4 2

Current location: 5

Circuit:

- Process node 5 – neigbour of node 2
- Delete edge between node 5 and node 2
- Process first neigbour of node 5: Node 1

Sample run



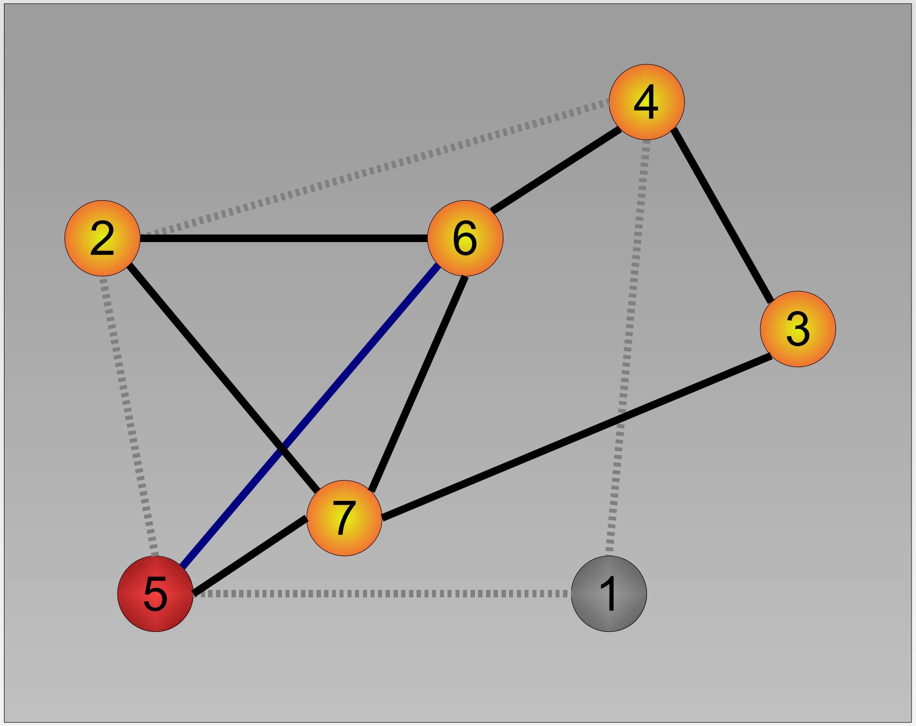
Stack: 1 4 2 5

Current location: 1

Circuit:

- Process node 1 – neigbour of node 5
- Delete edge between node 1 and node 5
- Node 1 has no more neighbours (and edges)
- Append node to circuit

Sample run



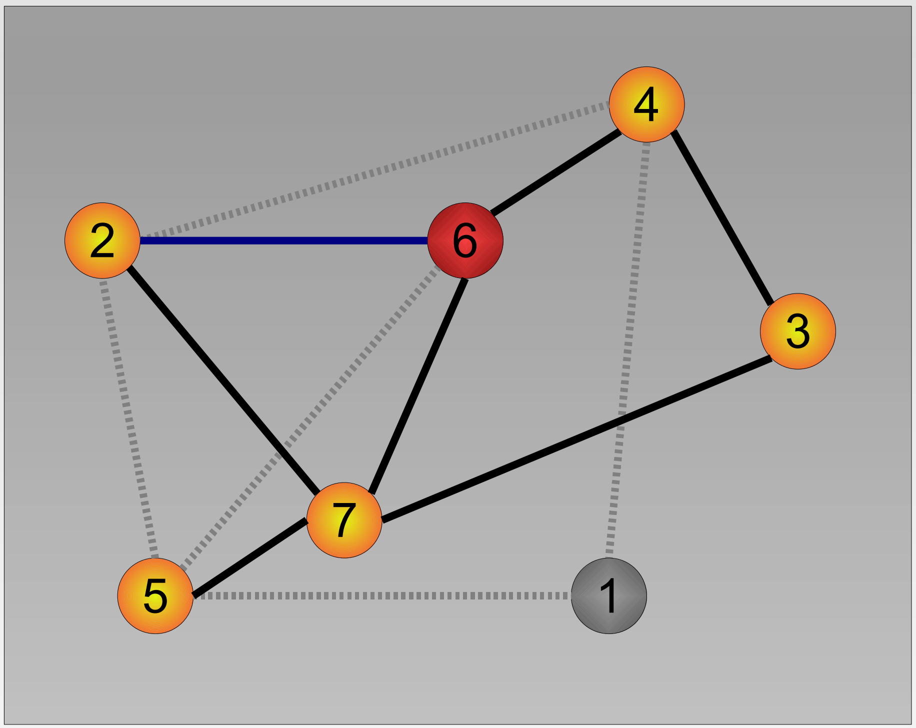
Stack: 1 4 2

Current location: 5

Circuit: 1

- Append node 1 to circuit
- Go back to node 5 – last one on stack
- Process second neigbour of node 5: Node 6

Sample run



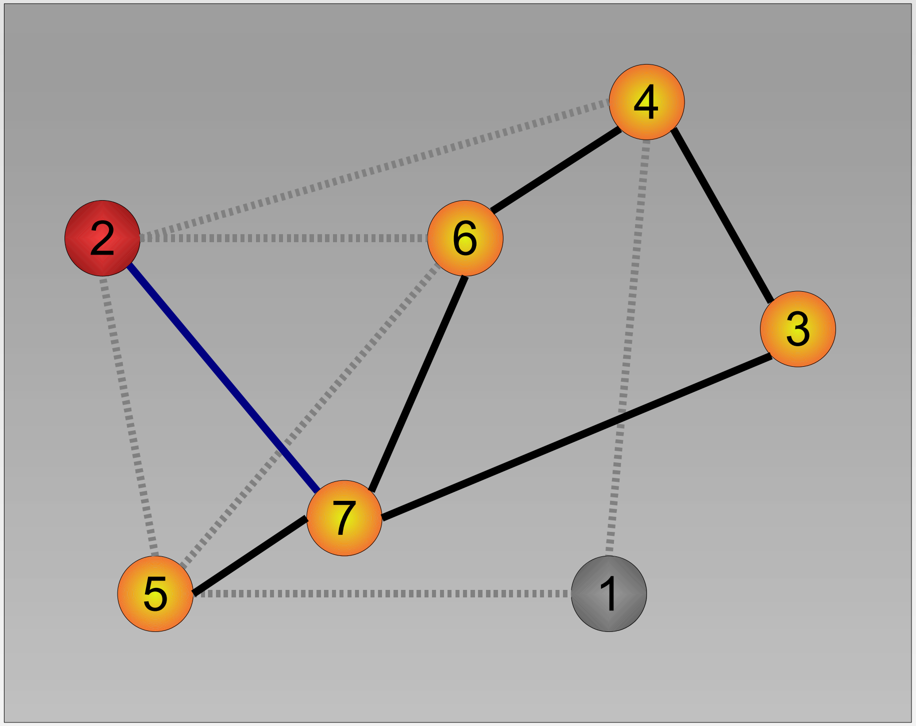
Stack: 1 4 2 5

Current location: 6

Circuit: 1

- Process node 6 – neigbour of node 5
- Delete edge between node 6 and node 5
- Process first neigbour of node 6: Node 2

Sample run



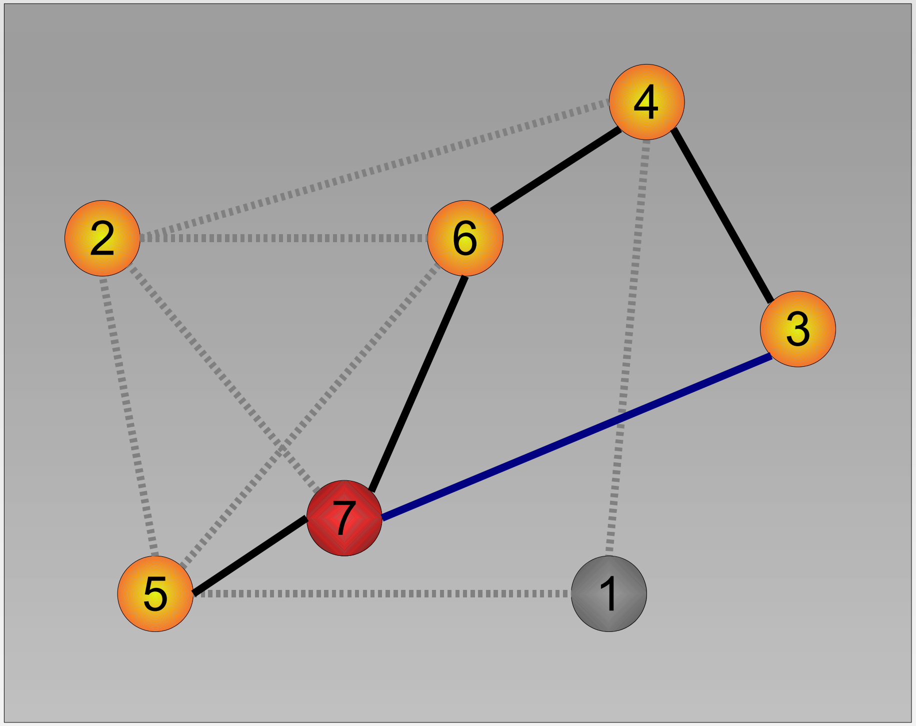
Stack: 1 4 2 5 6

Current location: 2

Circuit: 1

- Process node 2 – neigbour of node 6
- Delete edge between node 2 and node 6
- Process first neigbour of node 2: Node 7

Sample run



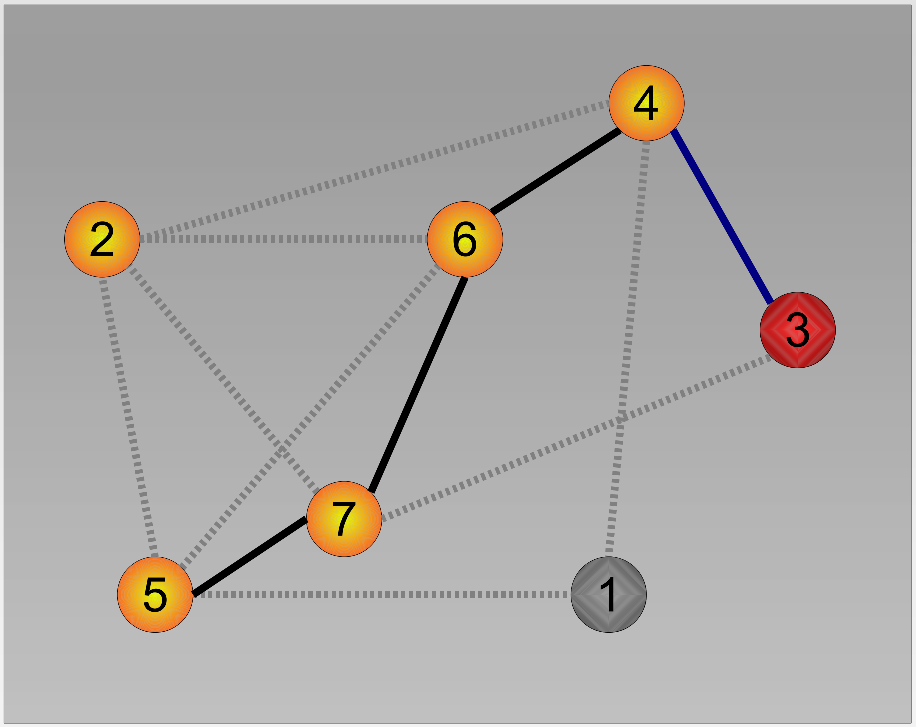
Stack: 1 4 2 5 6 2

Current location: 7

Circuit: 1

- Process node 7 – neigbour of node 2
- Delete edge between node 7 and node 2
- Process first neigbour of node 7: Node 3

Sample run



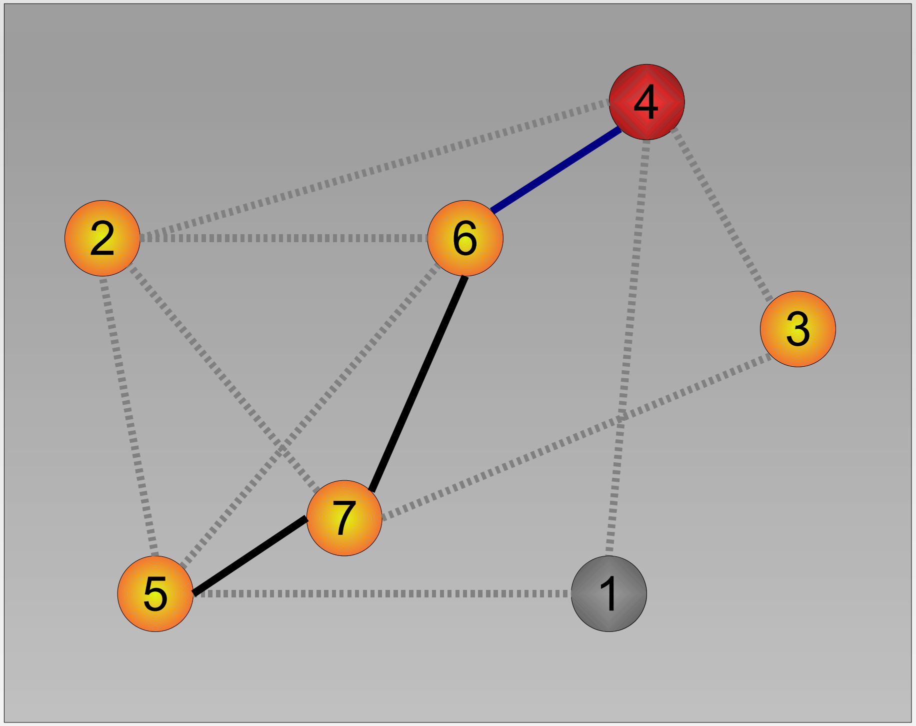
Stack: 1 4 2 5 6 2 7

Current location: 3

Circuit: 1

- Process node 3 – neigbour of node 7
- Delete edge between node 3 and node 7
- Process first neigbour of node 3: Node 4

Sample run



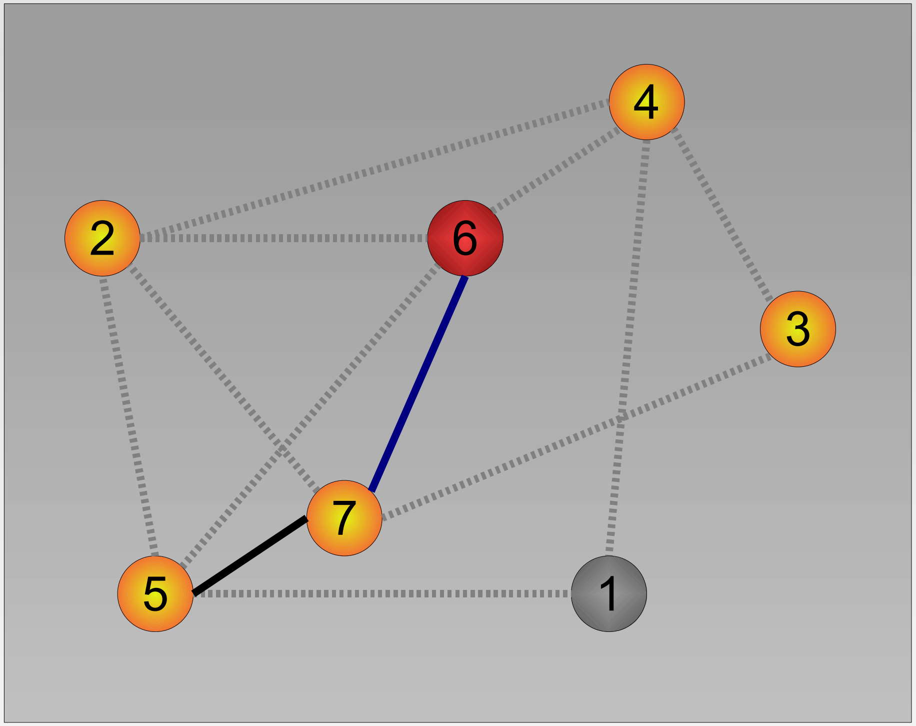
Stack: 1 4 2 5 6 2 7 3

Current location: 4

Circuit: 1

- Process node 4 – neigbour of node 3
- Delete edge between node 4 and node 3
- Process first neigbour of node 4: Node 6

Sample run



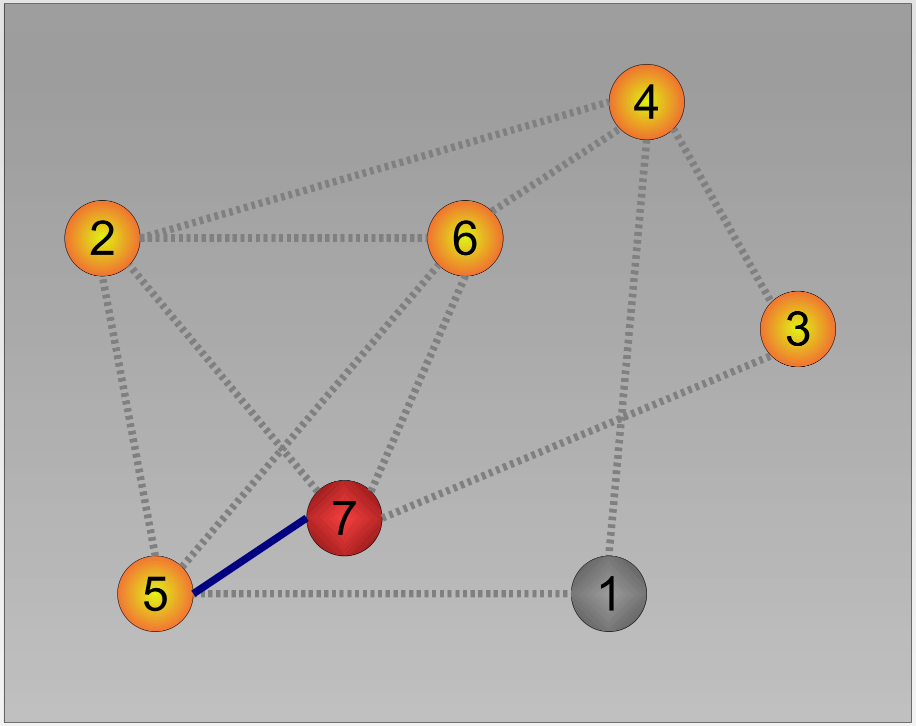
Stack: 1 4 2 5 6 2 7 3 4

Current location: 6

Circuit: 1

- Process node 6 – neigbour of node 4
- Delete edge between node 6 and node 4
- Process first neigbour of node 6: Node 7

Sample run



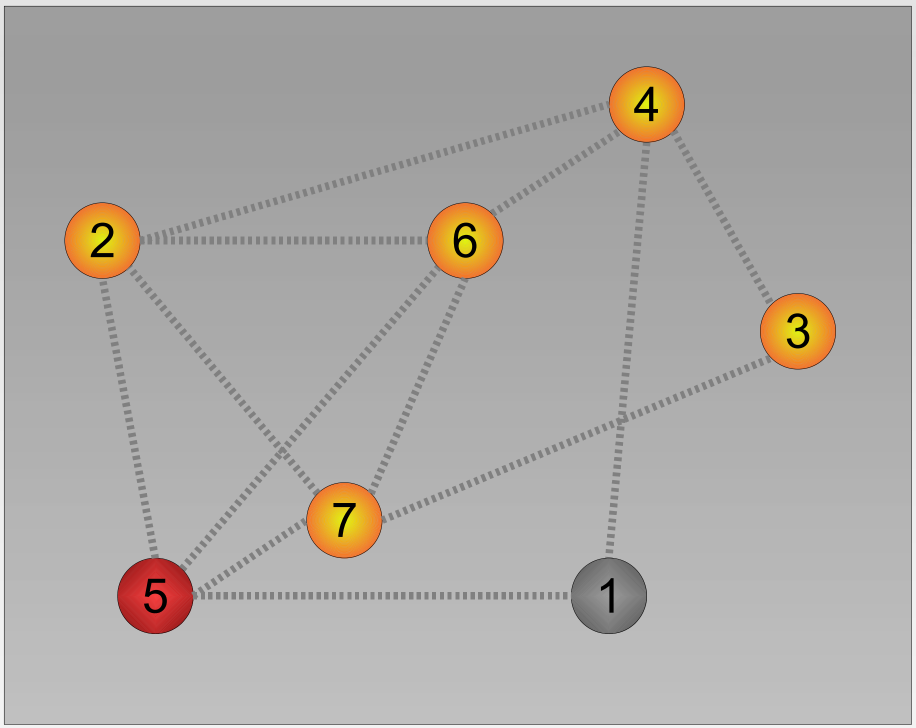
Stack: 1 4 2 5 6 2 7 3 4 6

Current location: 7

Circuit: 1

- Process node 7 – neigbour of node 6
- Delete edge between node 7 and node 6
- Process first neigbour of node 7: Node 5

Sample run



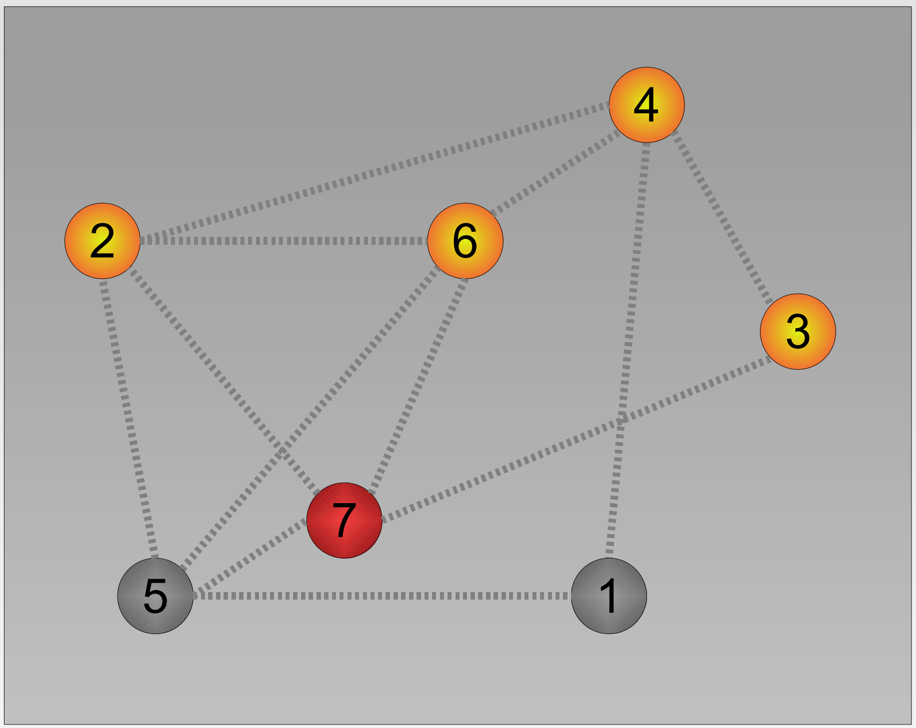
Stack: 1 4 2 5 6 2 7 3 4 6 7

Current location: 5

Circuit: 1

- Process node 5 – neigbour of node 7
- Delete edge between node 5 and node 7
- Node 5 has no more neighbours (and edges)
- Append node to circuit

Sample run



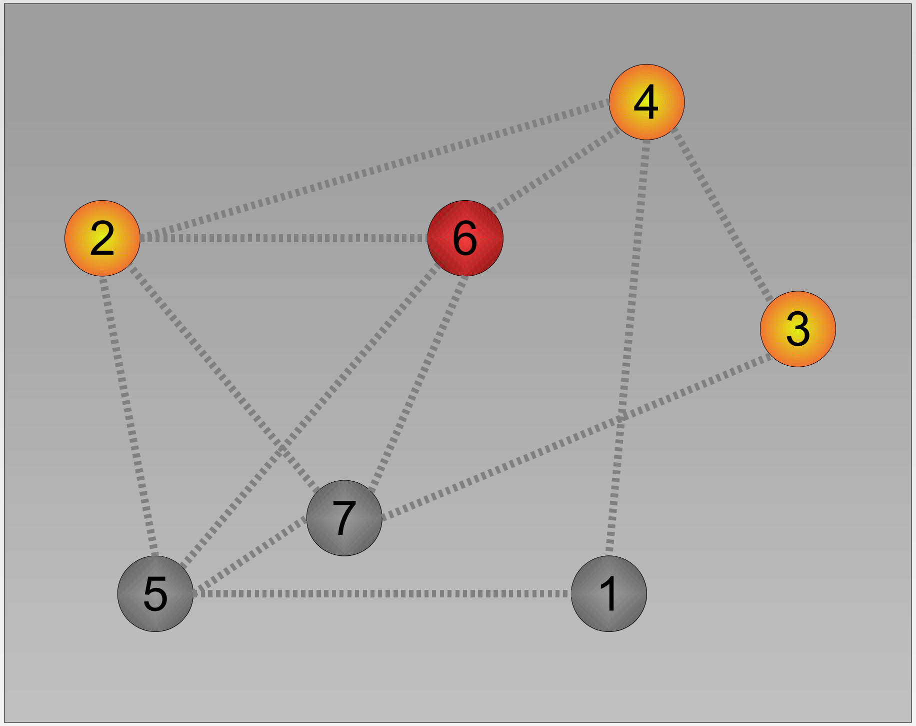
Stack: 1 4 2 5 6 2 7 3 4 6

Current location: 7

Circuit: 1 5

- Append node 5 to circuit
- Go back to node 7 – last one on stack
- Node 7 has no more neighbours (and edges)
- Append node to circuit

Sample run



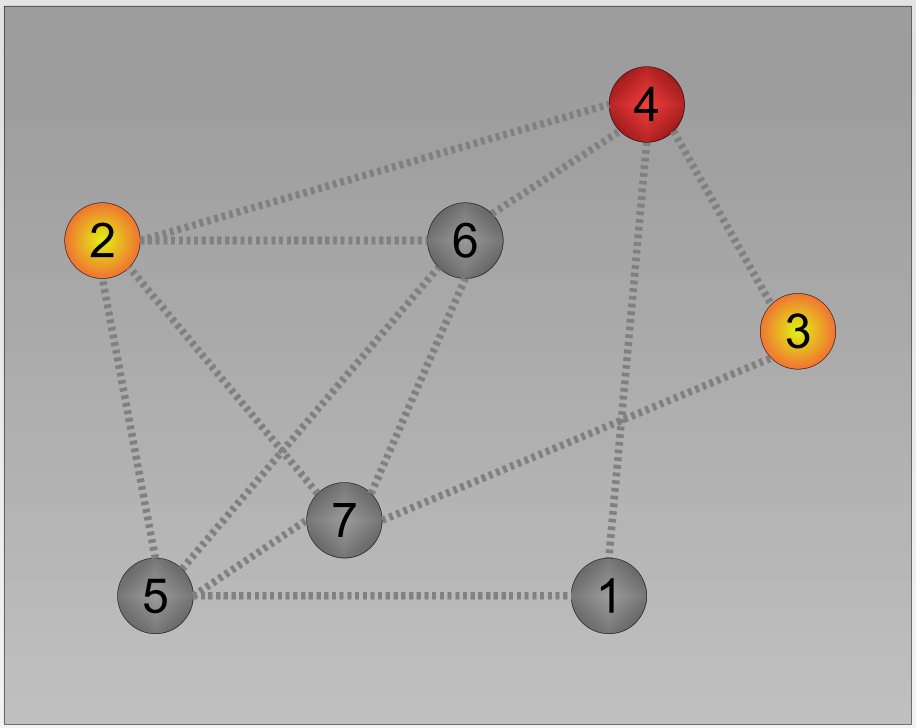
Stack: 1 4 2 5 6 2 7 3 4

Current location: 6

Circuit: 1 5 7

- Append node 7 to circuit
- Go back to node 6 – last one on stack
- Node 6 has no more neighbours (and edges)
- Append node to circuit

Sample run



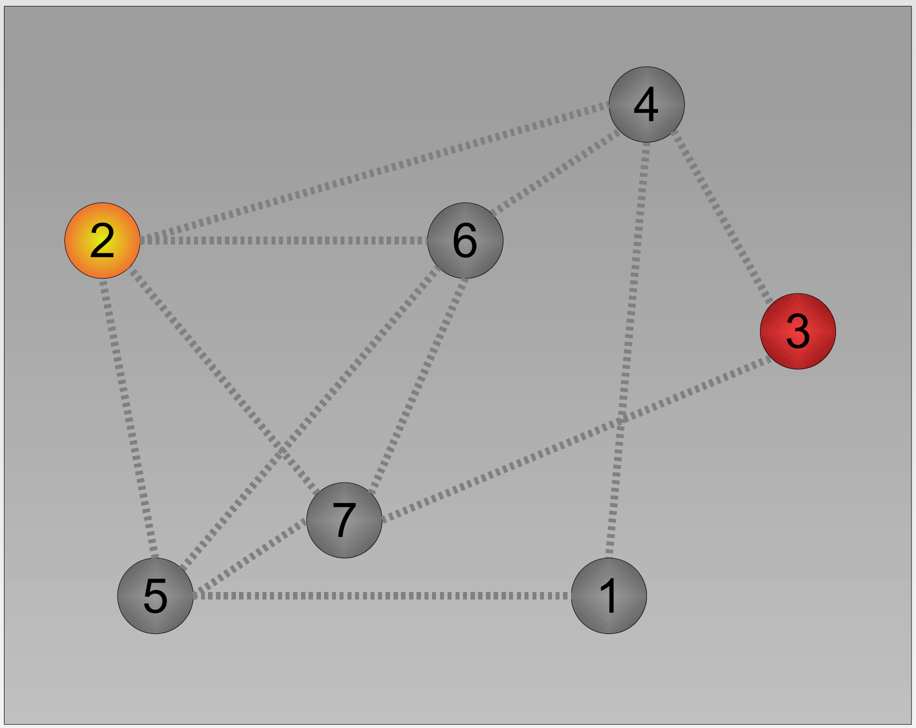
Stack: 1 4 2 5 6 2 7 3

Current location: 4

Circuit: 1 5 7 6

- Append node 6 to circuit
- Go back to node 4 – last one on stack
- Node 4 has no more neighbours (and edges)
- Append node to circuit

Sample run



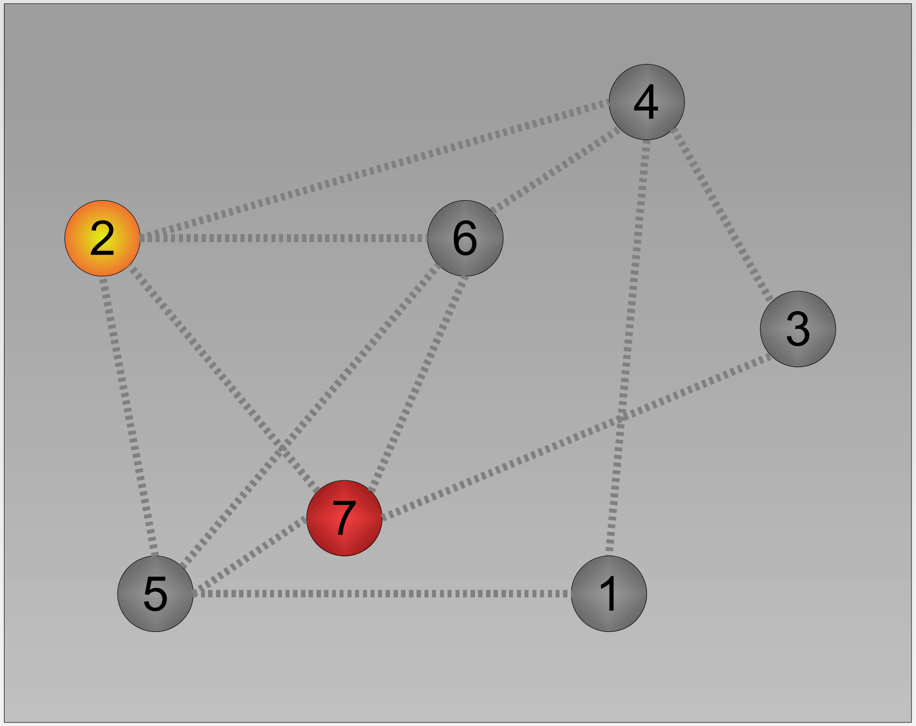
Stack: 1 4 2 5 6 2 7

Current location: 3

Circuit: 1 5 7 6 4

- Append node 4 to circuit
- Go back to node 3 – last one on stack
- Node 3 has no more neighbours (and edges)
- Append node to circuit

Sample run



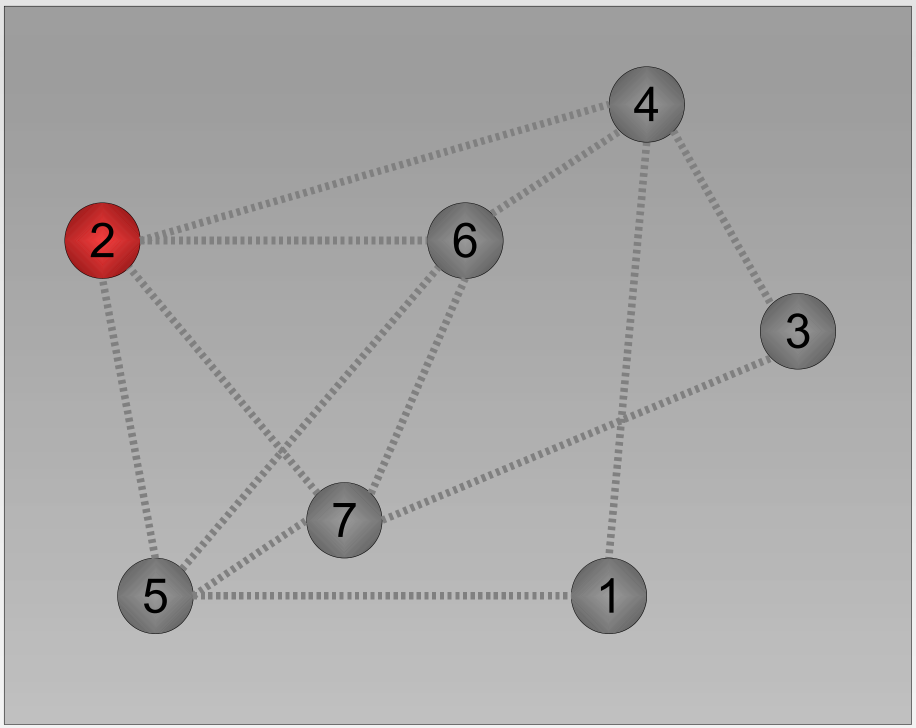
Stack: 1 4 2 5 6 2

Current location: 7

Circuit: 1 5 7 6 4 3

- Append node 3 to circuit
- Go back to node 7 – last one on stack
- Node 7 has no more neighbours (and edges)
- Append node to circuit

Sample run



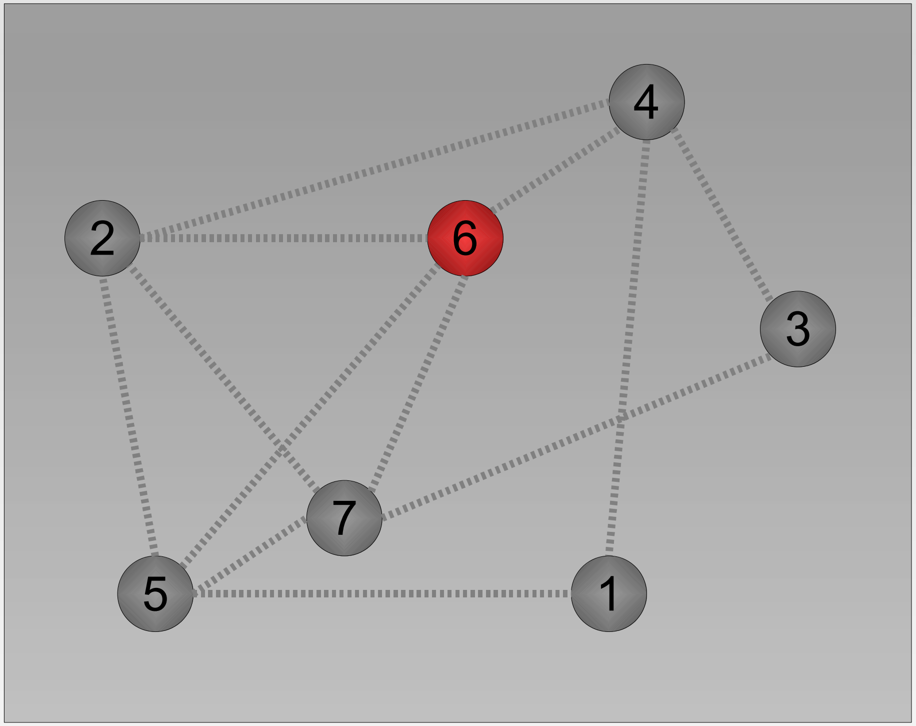
Stack: 1 4 2 5 6

Current location: 2

Circuit: 1 5 7 6 4 3 7

- Append node 7 to circuit
- Go back to node 2 – last one on stack
- Node 2 has no more neighbours (and edges)
- Append node to circuit

Sample run



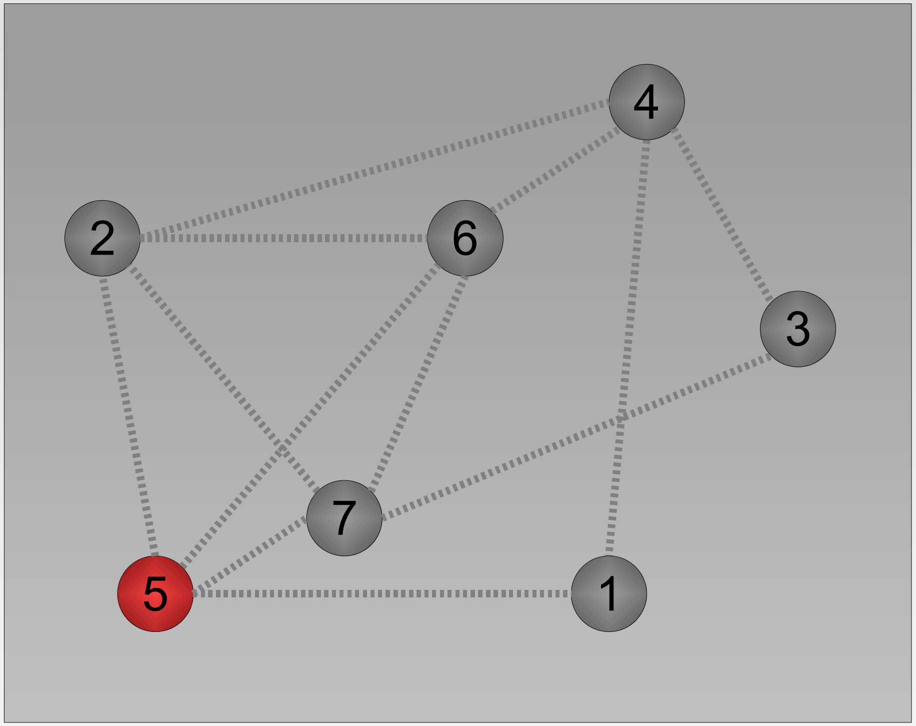
Stack: 1 4 2 5

Current location: 6

Circuit: 1 5 7 6 4 3 7 2

- Append node 2 to circuit
- Go back to node 6 – last one on stack
- Node 6 has no more neighbours (and edges)
- Append node to circuit

Sample run



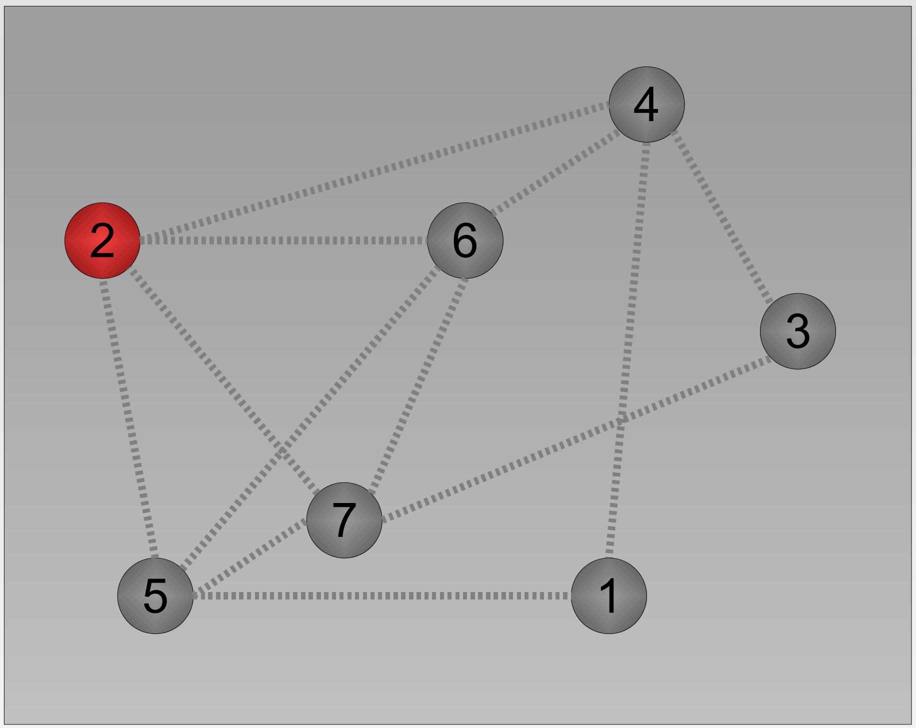
Stack: 1 4 2

Current location: 5

Circuit: 1 5 7 6 4 3 7 2 6

- Append node 6 to circuit
- Go back to node 5 – last one on stack
- Node 5 has no more neighbours (and edges)
- Append node to circuit

Sample run



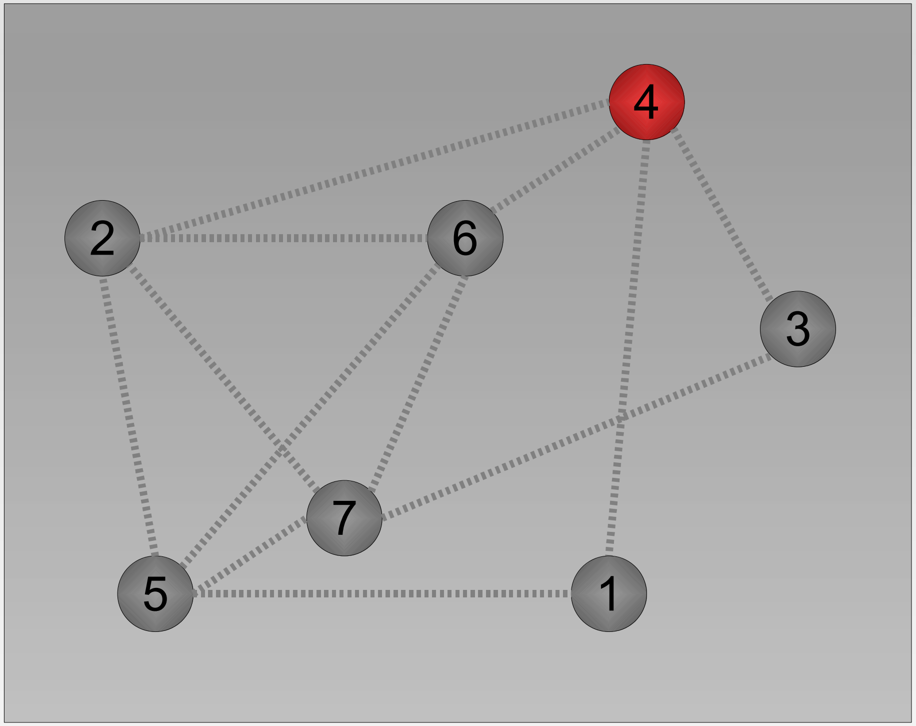
Stack: 1 4

Current location: 2

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

- Append node 5 to circuit
- Go back to node 2 – last one on stack
- Node 2 has no more neighbours (and edges)
- Append node to circuit

Sample run



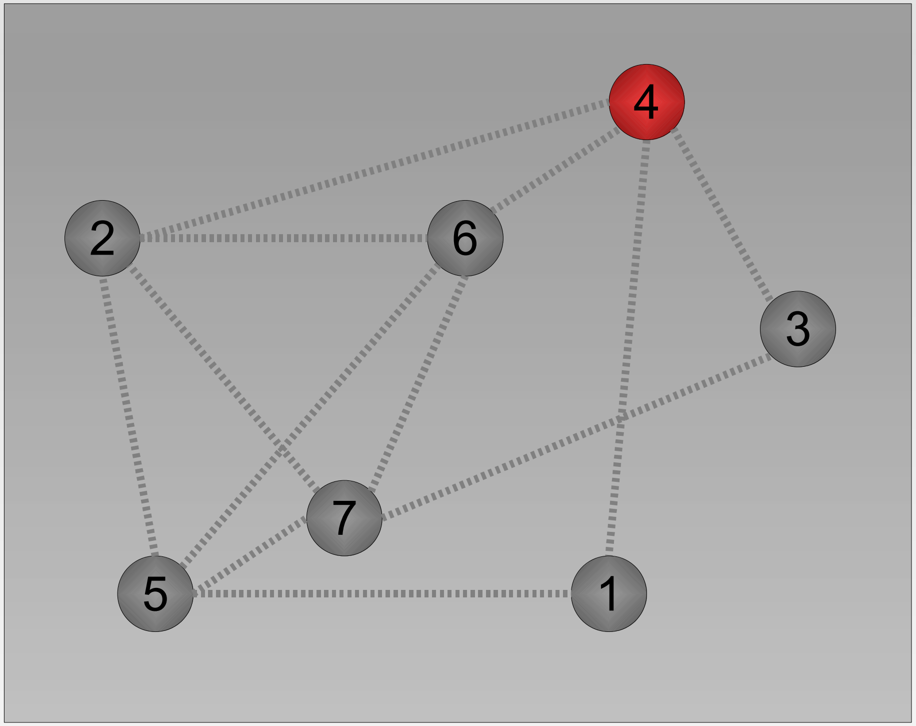
Stack: 1

Current location: 4

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

- Append node 2 to circuit
- Go back to node 4 – last one on stack
- Node 4 has no more neighbours (and edges)
- Append node to circuit

Sample run



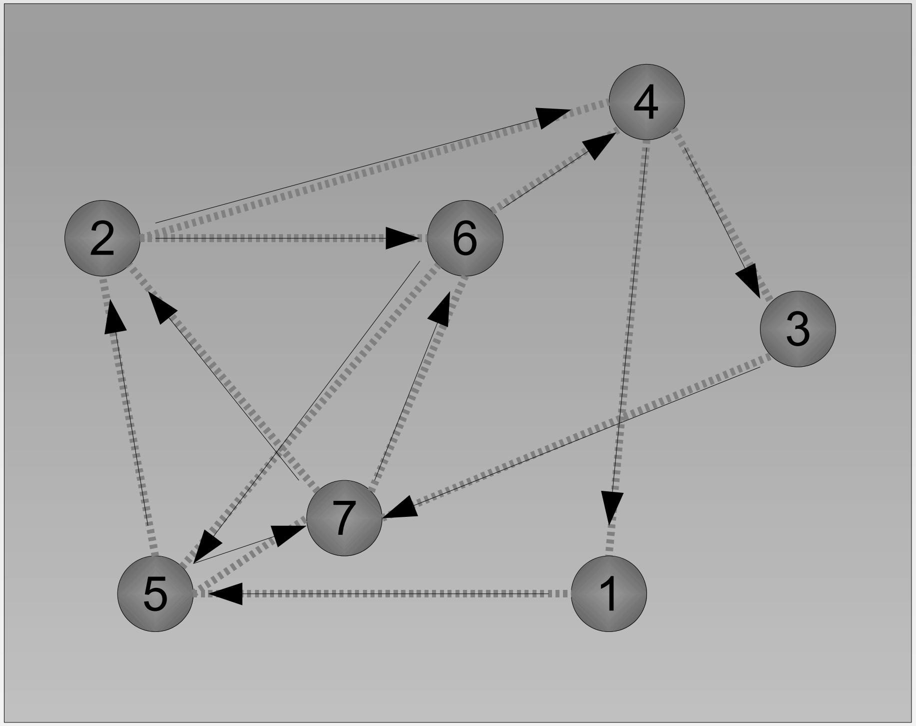
Stack:

Current location: 1

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

- Append node 4 to circuit
- Go back to node 1 – last one on stack
- Node 1 has no more neighbours (and edges)
- Append node to circuit

Sample run



Stack:

Current location:

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

- Append node 1 to curcuit
- Nothing on stack left
- Finished!

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.

Extensions

- 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.

The end

THE END

Any questions?