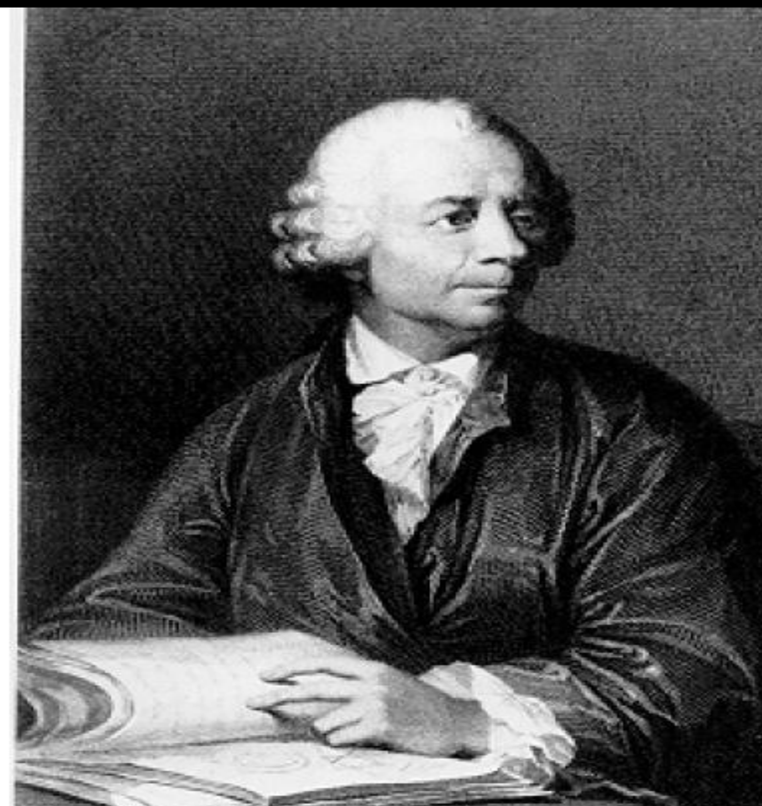
Eulerian Paths and Cycles

What is a Eulerian Path

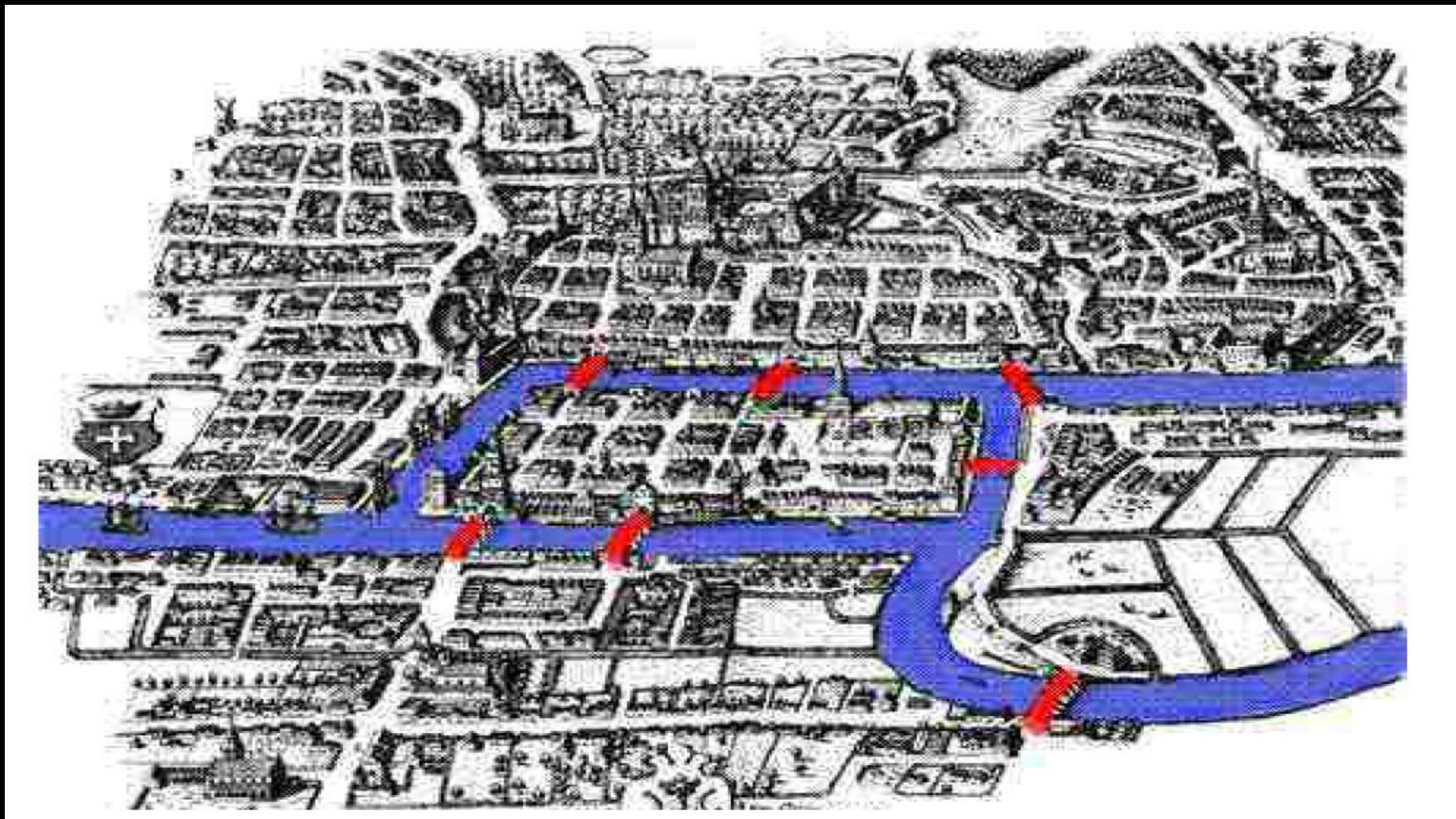
- Given an graph.
- Find a path which uses every edge exactly once.
- This path is called an Eulerian Path.
- If the path begins and ends at the same vertex, it is called a Eulerian Cycle.

Where did all start: Koningsberg



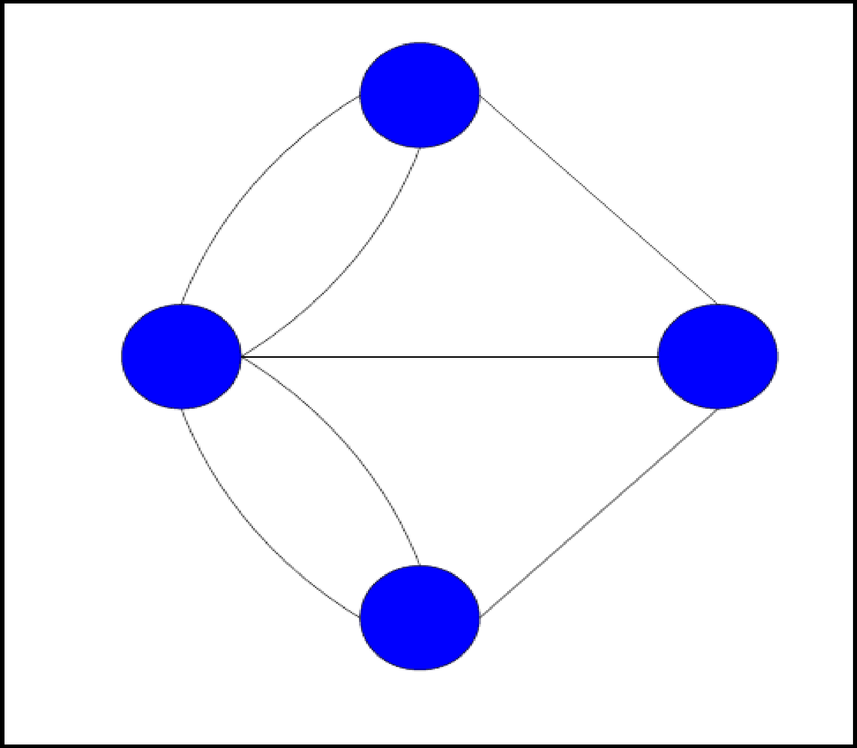
Koningsberg

Find a route which crosses each bridge exactly once?



Koningsberg Graph

This graph represents the Koningsburg bridges



When do Eulerian Paths and Cycles exist?

- Euler’s solution
- An Eulerian cycle exists if and only if it is connected and every node has ‘even degree’.
- An Eulerian path exists if and only if it is connected and every node except two has even degree.
- In the Eulerian path the 2 nodes with odd degree have to be the start and end vertices

Proof: a Eulerian graph must have all vertices of even degree

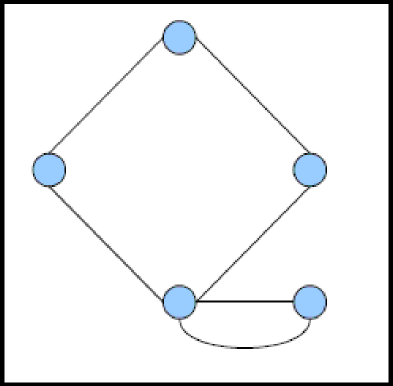
- Let C be an Eulerian cycle of graph G, which starts and ends at vertex u.
- Each time a vertex is included in the cycle C, two edges connected to that vertex are used up.
- Every edge in G is included in the cycle. So every vertex other than u must have even degree.
- The tour starts and ends at u, so it must also have even degree.

Proof: a graph with all vertices of even degree must be Eulerian

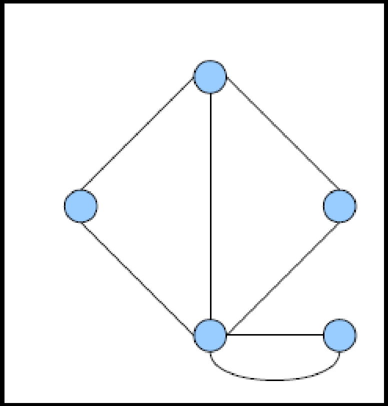
- **Assume the opposite**: G is a non-eulerian graph with all vertices of even degree.
- G must contain a cycle. Let C be the largest possible cycle in the graph.
- Because of our assumption, C must have missed out some of the graph G, call this D.
- C is Eulerian, so has no vertices of odd degree. D therefore also has no vertices of odd degree.
- D must have some cycle E which shares a common vertex with C
- Combination of C and E therefore makes a cycle larger than C, which violates our assumption in (2). **Contradiction**.

Examples

Eulerian Cycle:

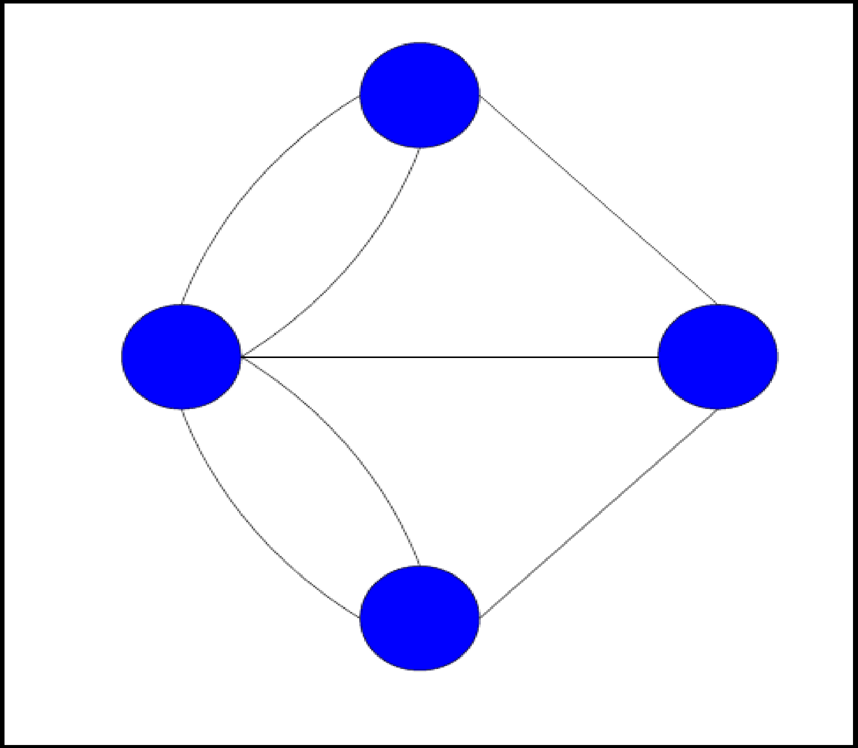


Eulerian Path:



And Koningsburg?

- No Eulerian Path or cycle!



Finding Eulerian Cycles

- Start off with a node
- Find a cycle containing that node
- Find a node along that path which has an edge that has not been used
- Find a cycle starting at this node witch uses the unused edge
- Splice this new cycle into the existing cycle
- Continue in this way until no nodes exist with unused edges
- Since the graph is connected this implies we have found a Eulerian Cycle

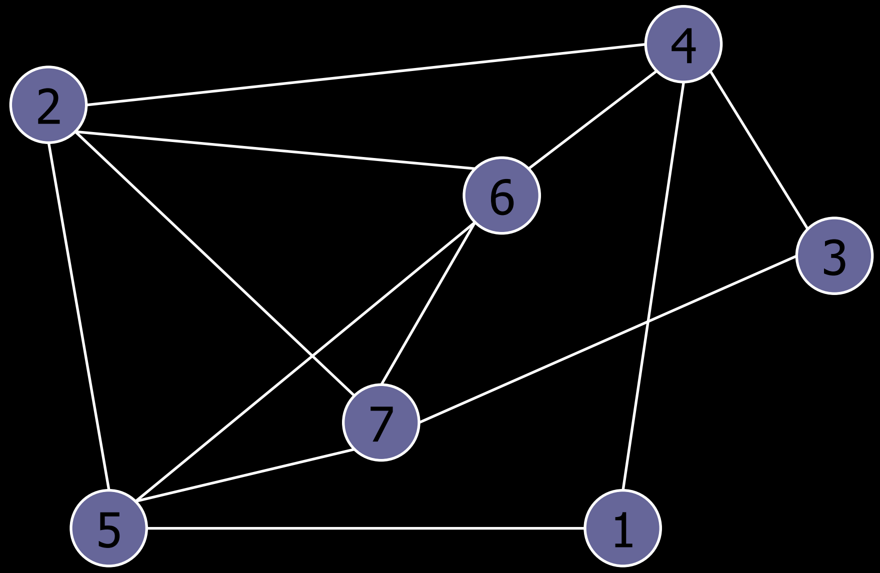
Formal Algorithm

- 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 until the node has no more neighbors
  - To process a neighbour, delete the edge between the current node and its neighbor, recurse on the neighbor
  - After processing all neighbours append current node to the circuit.

Pseudo-Code

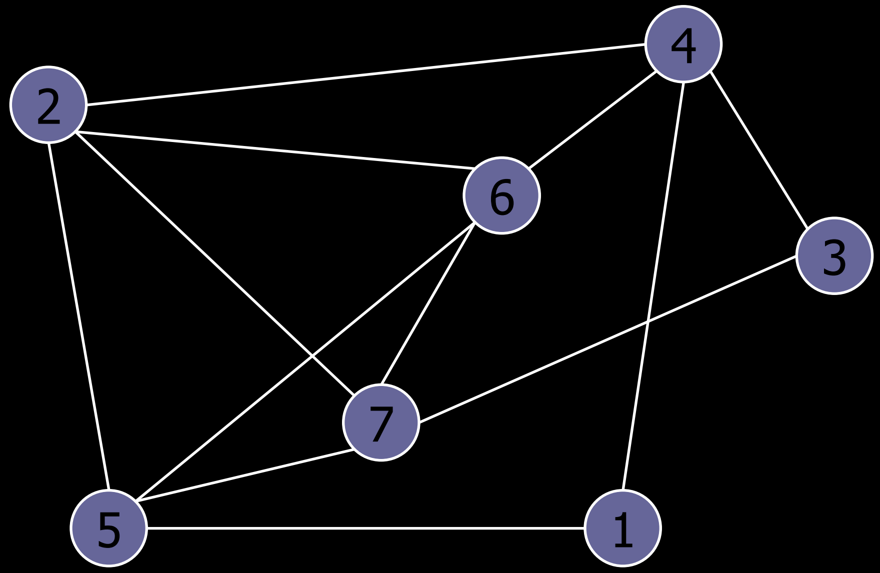
- ```
  find_circuit (node i)
    if node i has no neighbors
      circuit [circuitpos] = node i
      circuitpos++
    else
      while (node i has neighbors)
        pick a neighbor j of node i
        delete_edges (node j, node i)
        find_circuit (node j)
      circuit [circuitpos] = node i
      circuitpos++
  ```
  ```

Execution Example



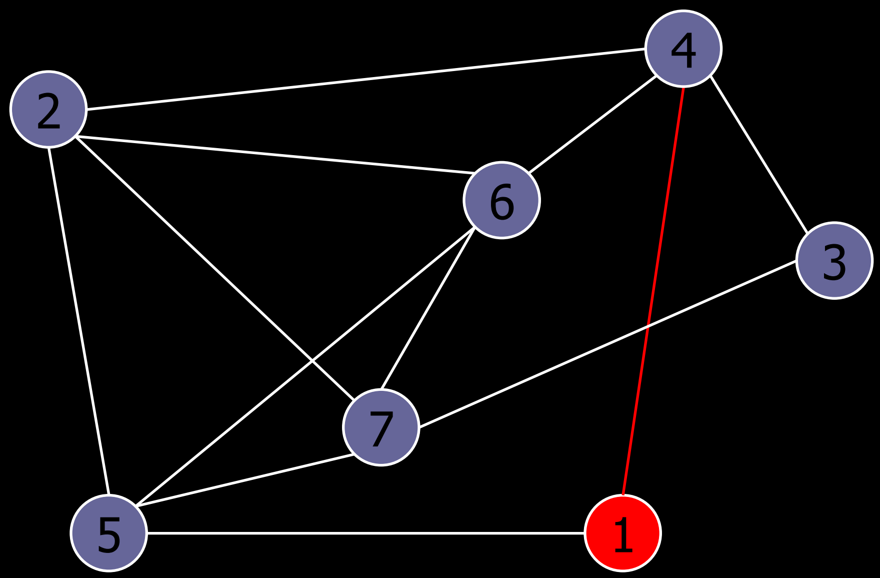
- Stack:
- Location:
- Circuit:

Execution Example



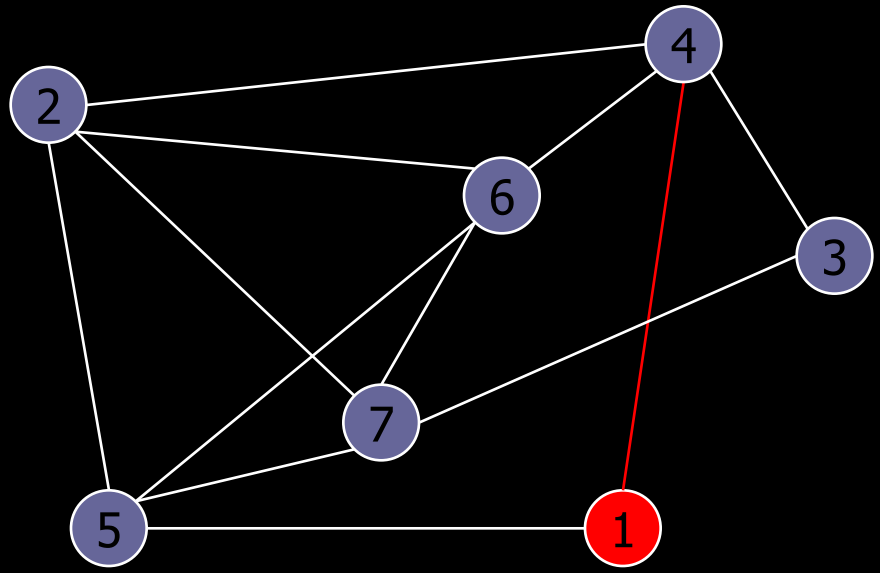
- Stack:
- Location:
- Circuit:

Execution Example



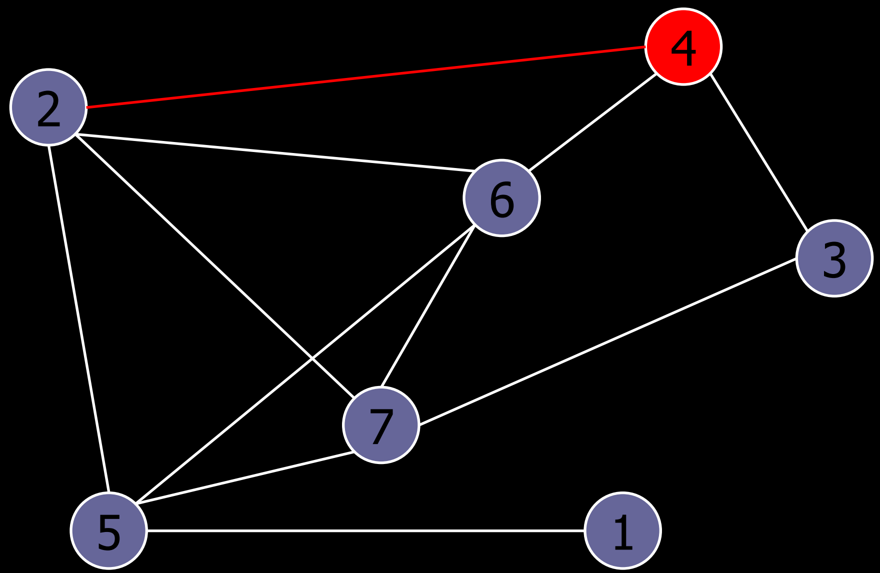
- Stack:
- Location: 1
- Circuit:

Execution Example



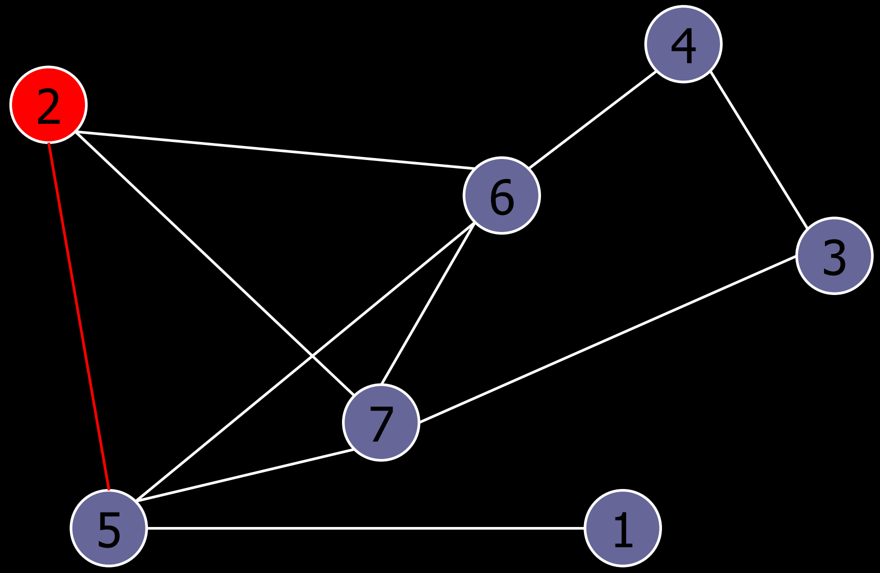
- Stack:
- Location: 1
- Circuit:

Execution Example



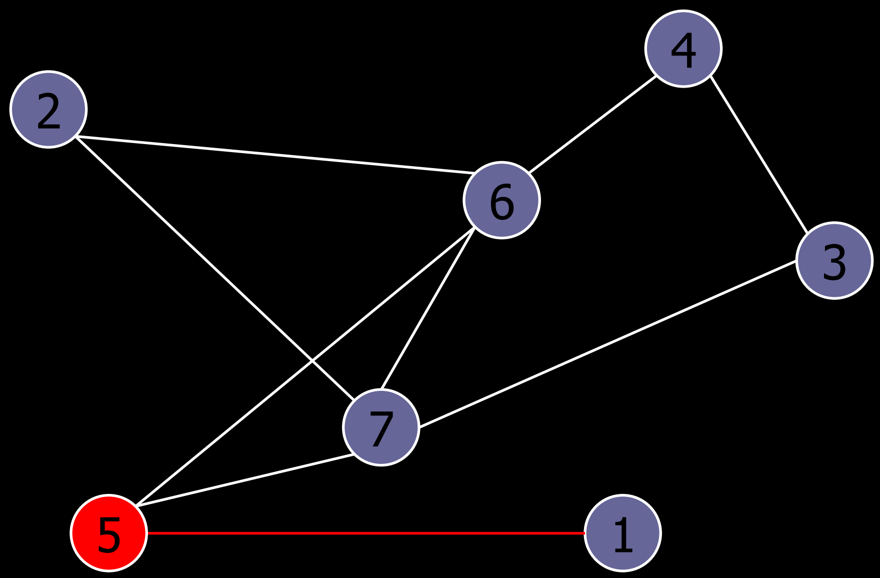
- Stack: 1
- Location: 4
- Circuit:

Execution Example



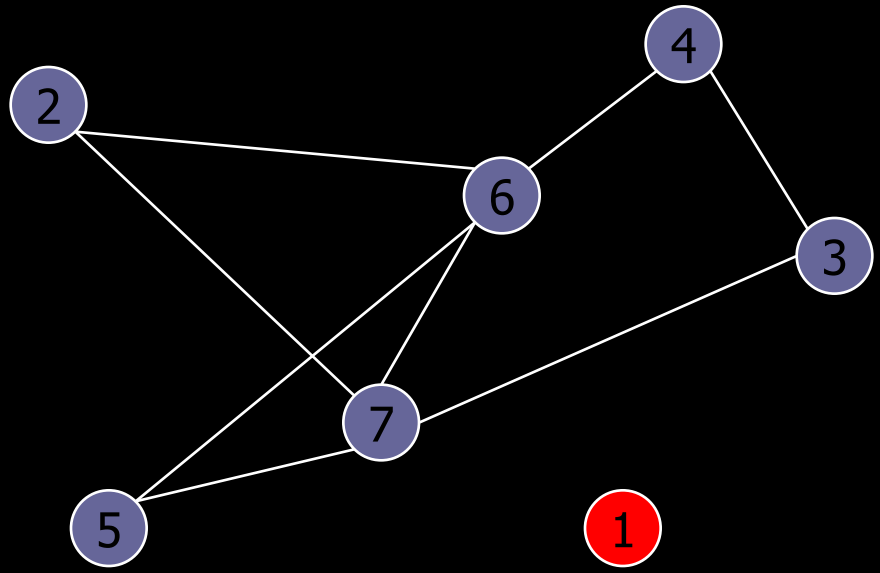
- Stack: 1 4
- Location: 2
- Circuit:

Execution Example



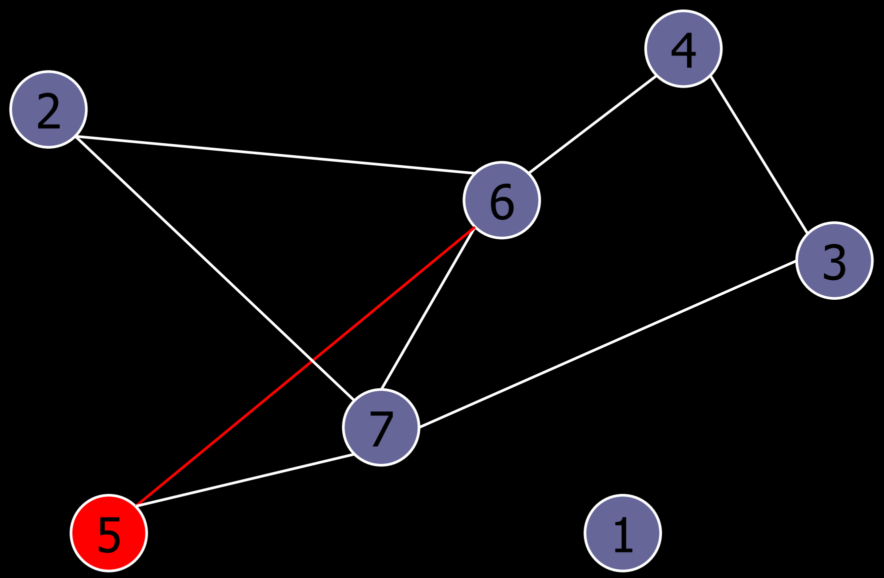
- Stack: 1 4 2
- Location: 5
- Circuit:

Execution Example



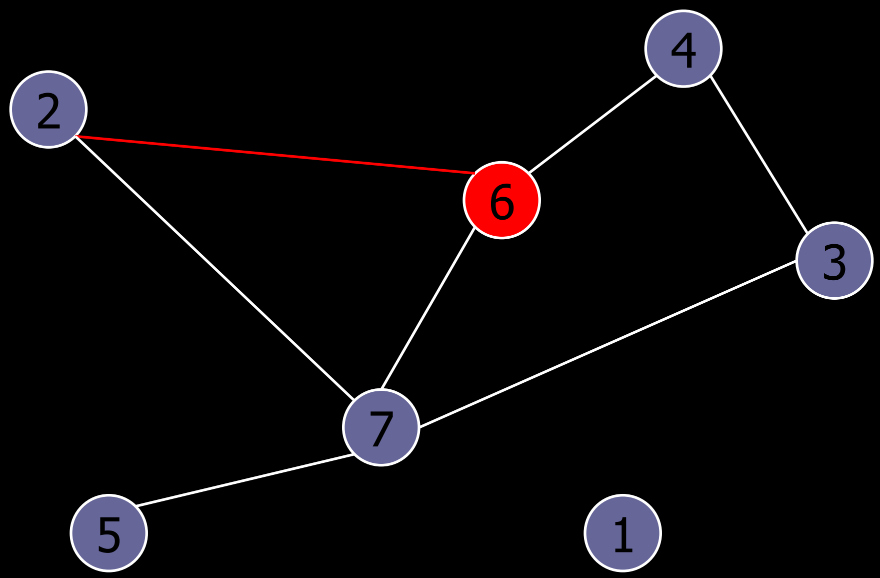
- Stack: 1 4 2 5
- Location: 1
- Circuit:

Execution Example



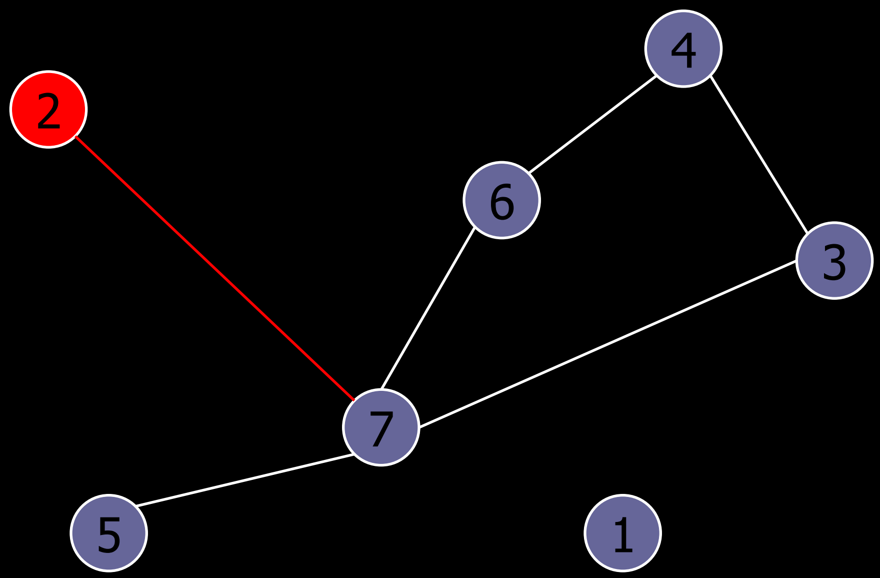
- Stack: 1 4 2
- Location: 5
- Circuit: 1

Execution Example



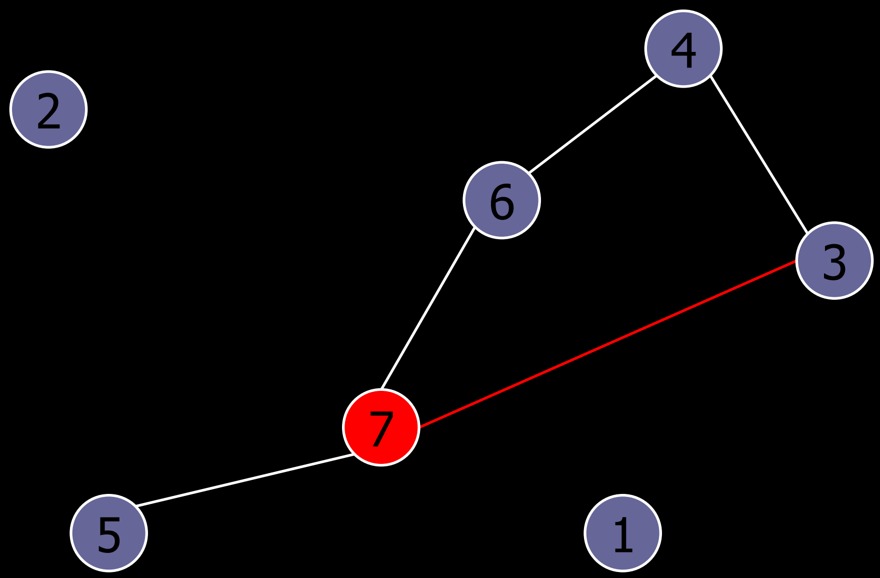
- Stack: 1 4 2 5
- Location: 6
- Circuit: 1

Execution Example



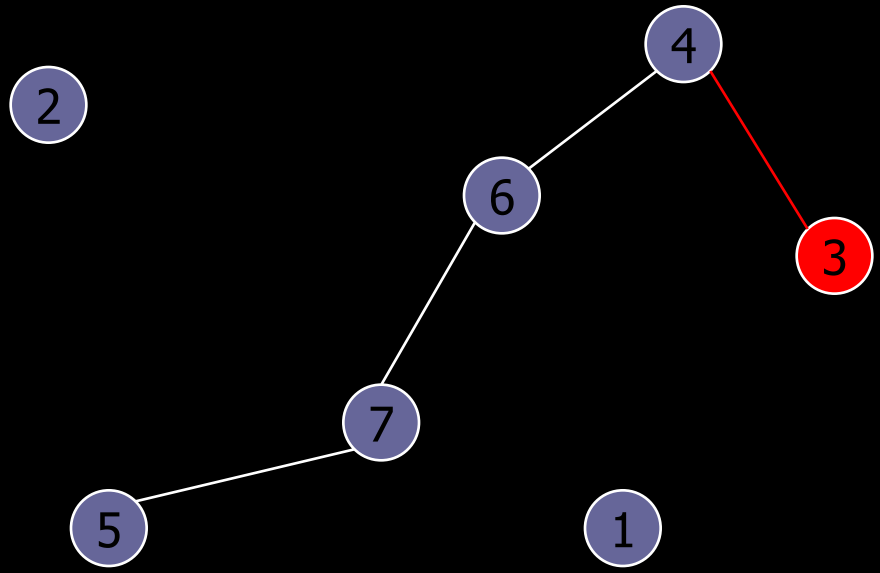
- Stack: 1 4 2 5 6
- Location: 2
- Circuit: 1

Execution Example



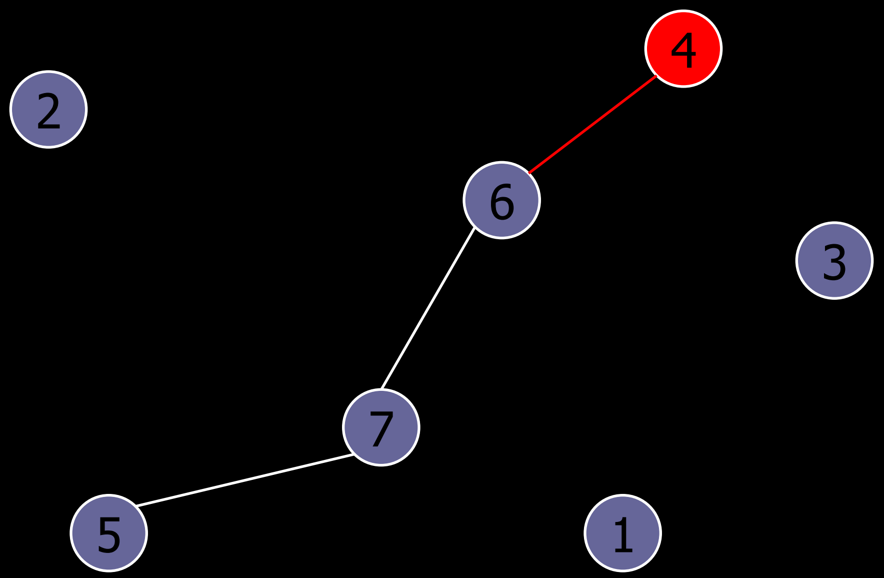
- Stack: 1 4 2 5 6 2
- Location: 7
- Circuit: 1

Execution Example



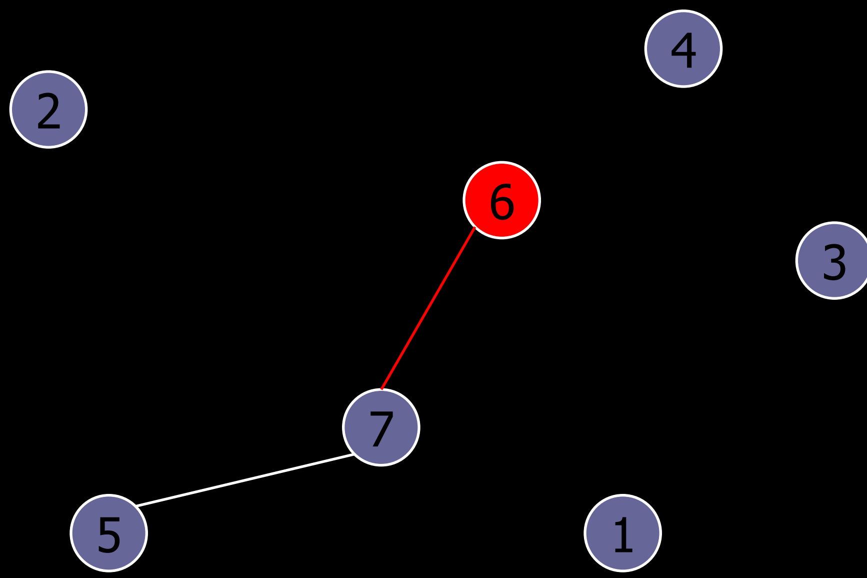
- Stack: 1 4 2 5 6 2 7
- Location: 3
- Circuit: 1

Execution Example



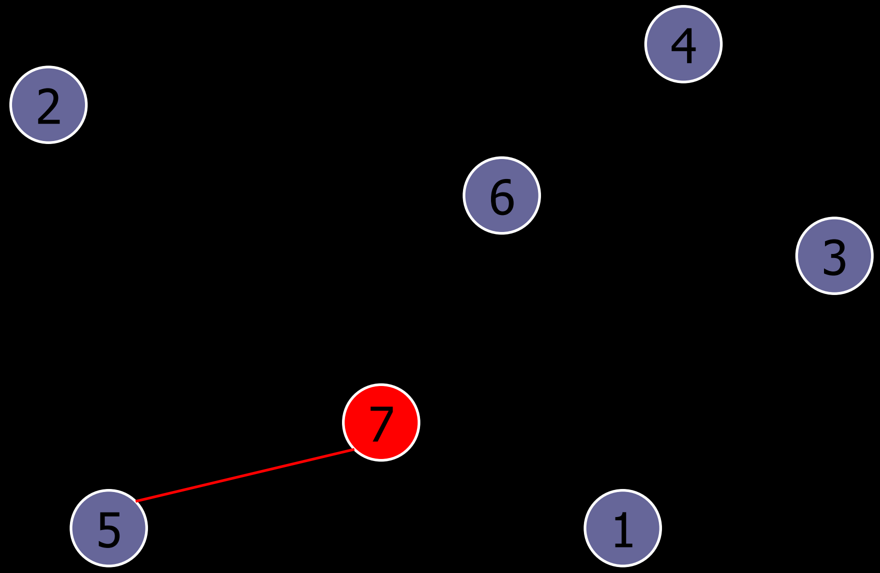
- Stack: 1 4 2 5 6 2 7 3
- Location: 4
- Circuit: 1

Execution Example



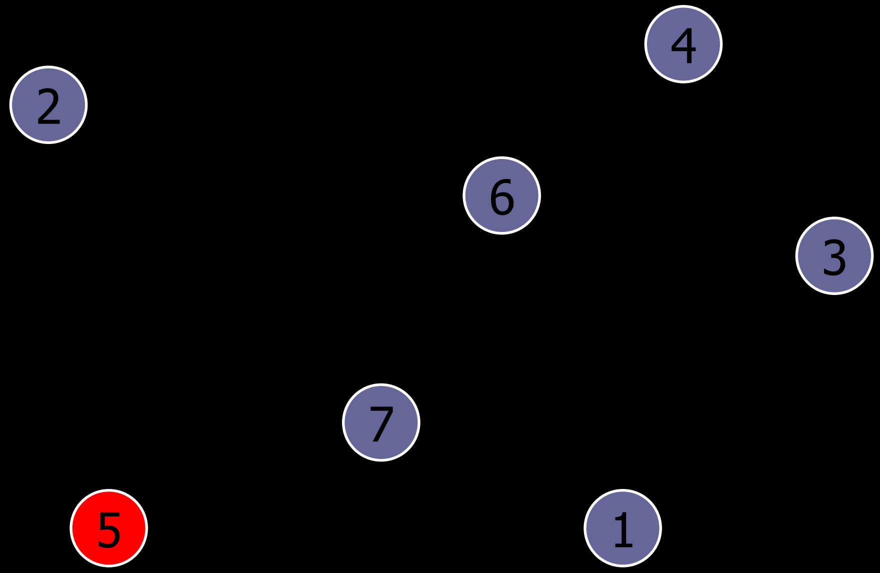
- Stack: 1 4 2 5 6 2 7 3 4
- Location: 6
- Circuit: 1

Execution Example



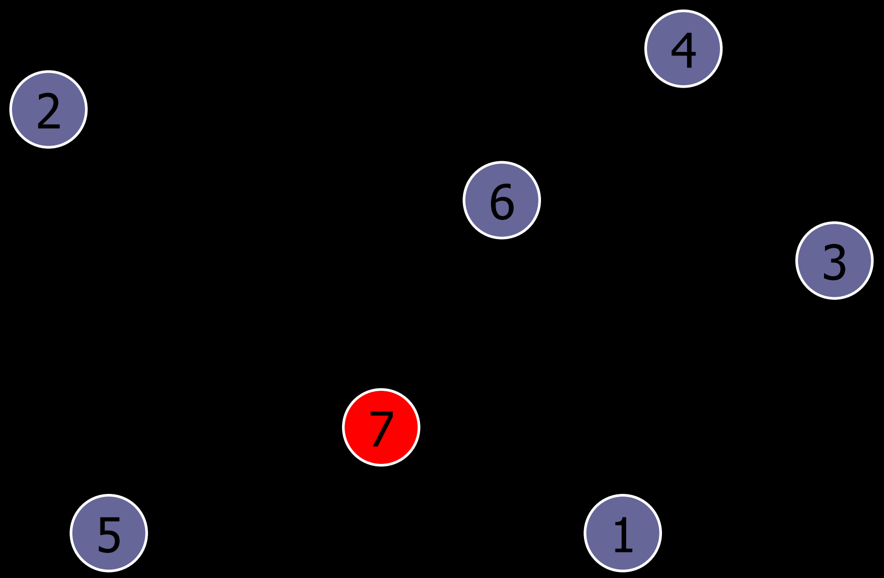
- Stack: 1 4 2 5 6 2 7 3 4 6
- Location: 7
- Circuit: 1

Execution Example



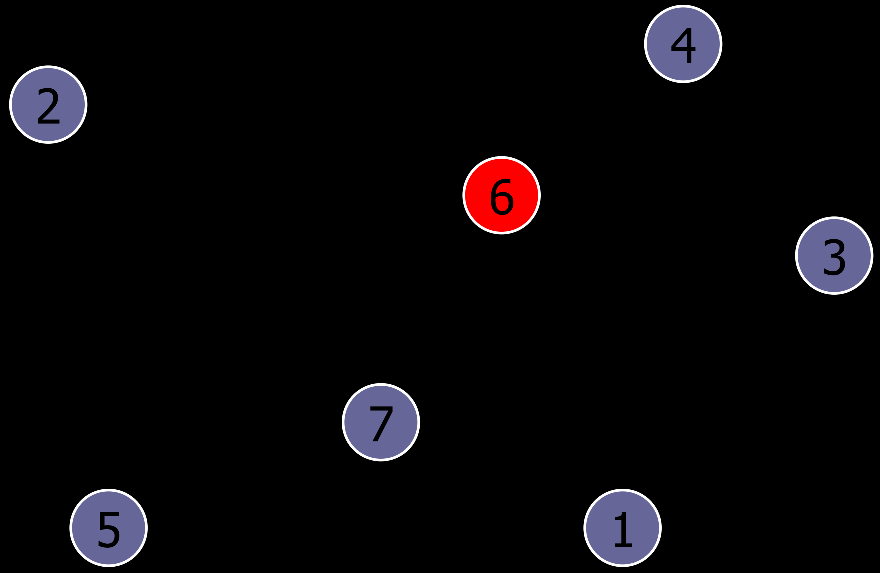
- Stack: 1 4 2 5 6 2 7 3 4 6 7
- Location: 5
- Circuit: 1

Execution Example



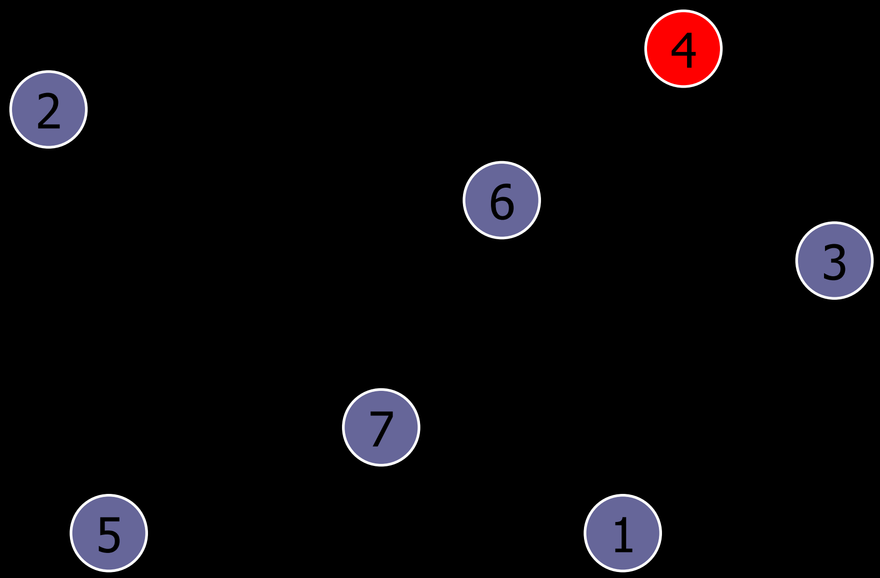
- Stack: 1 4 2 5 6 2 7 3 4 6
- Location: 7
- Circuit: 1 5

Execution Example



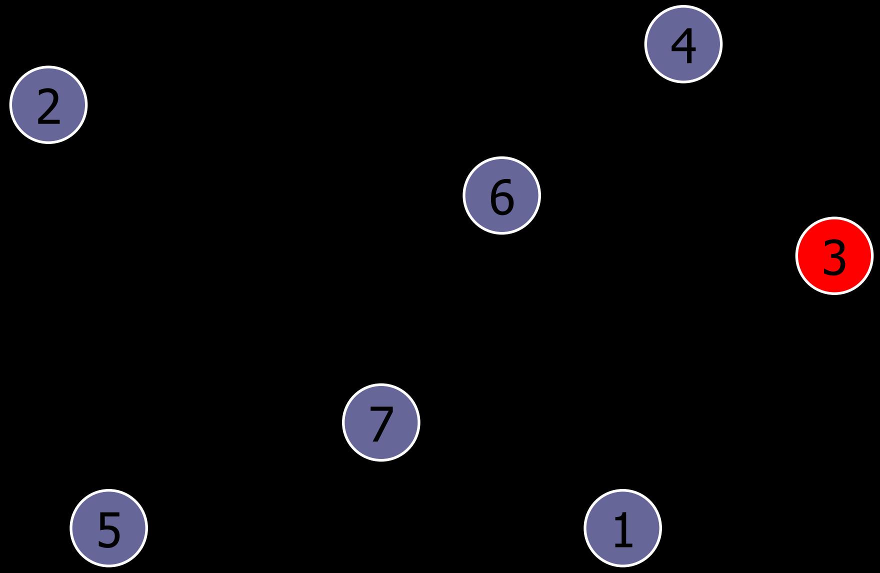
- Stack: 1 4 2 5 6 2 7 3 4
- Location: 6
- Circuit: 1 5 7

Execution Example



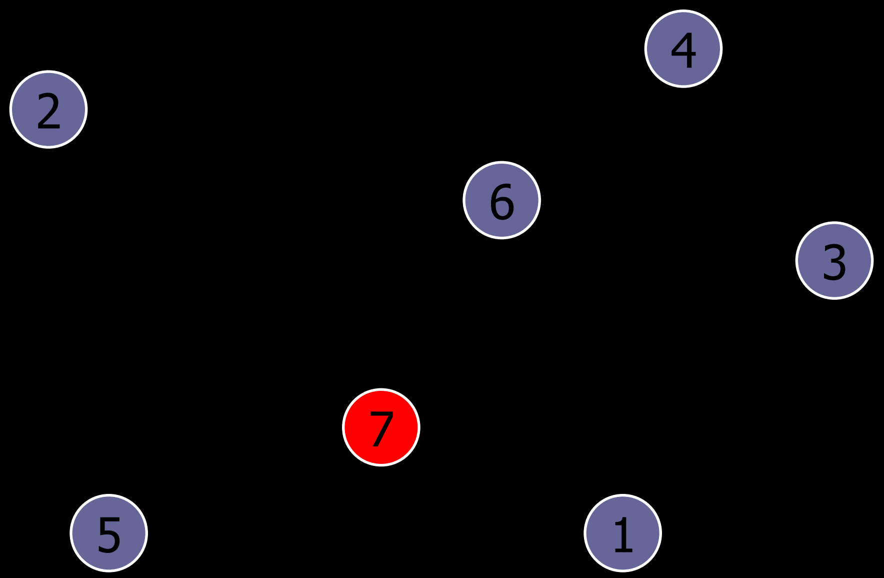
- Stack: 1 4 2 5 6 2 7 3
- Location: 4
- Circuit: 1 5 7 6

Execution Example



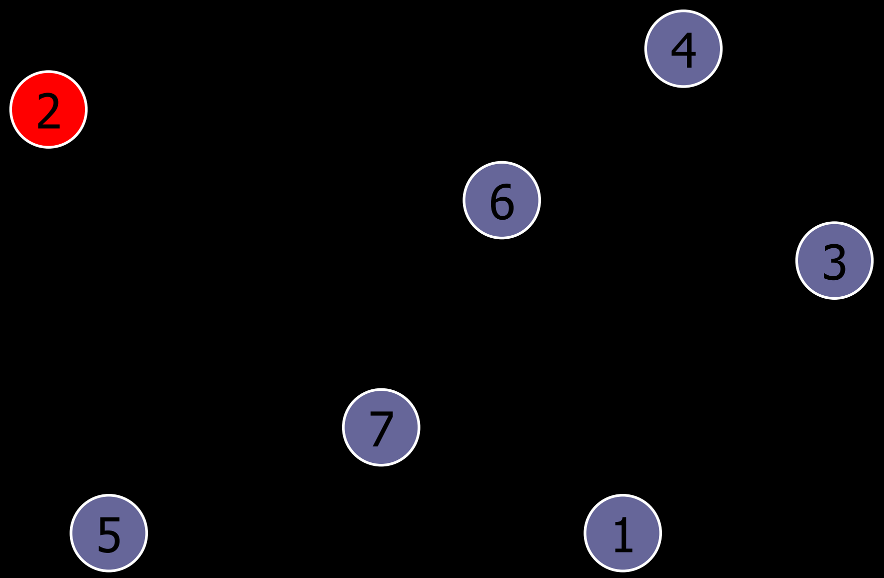
- Stack: 1 4 2 5 6 2 7
- Location: 3
- Circuit: 1 5 7 6 4

Execution Example



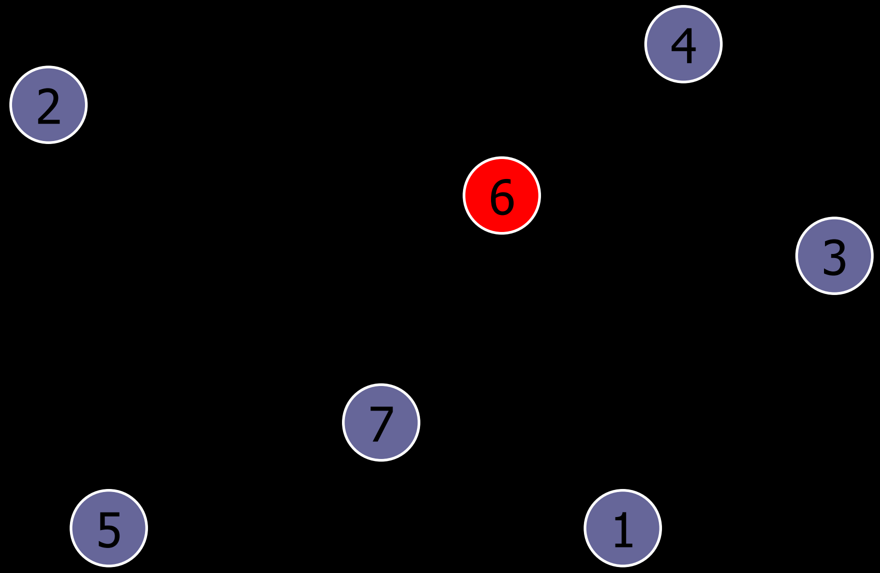
- Stack: 1 4 2 5 6 2
- Location: 7
- Circuit: 1 5 7 6 4 3

Execution Example



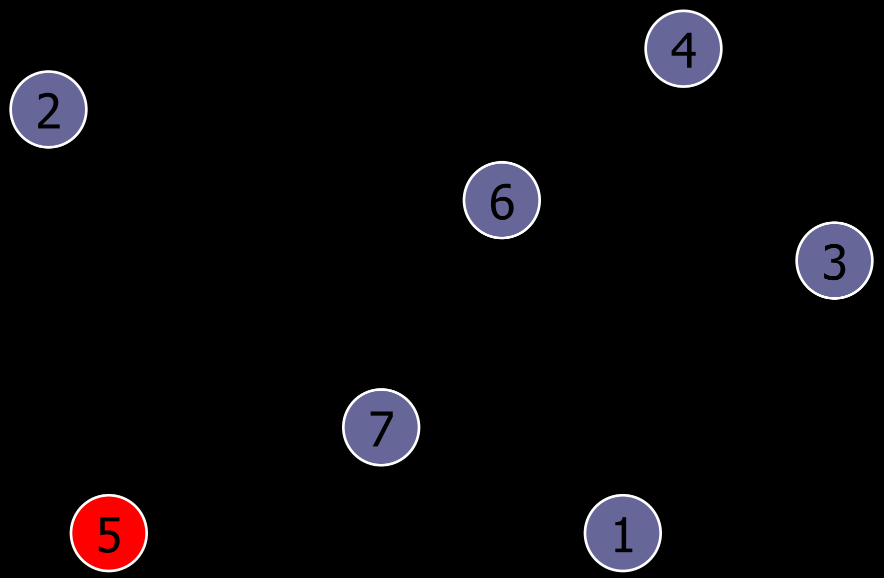
- Stack: 1 4 2 5 6
- Location: 2
- Circuit: 1 5 7 6 4 3 7

Execution Example



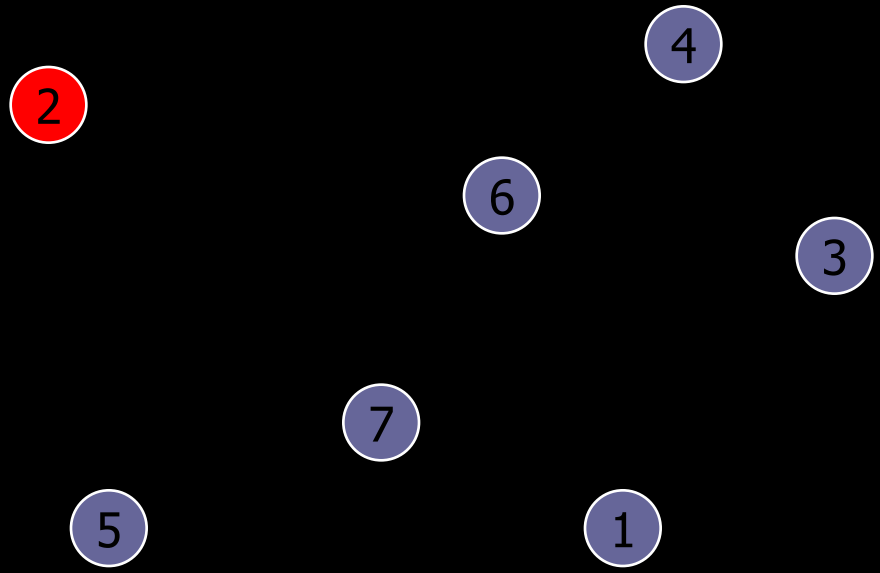
- Stack: 1 4 2 5
- Location: 6
- Circuit: 1 5 7 6 4 3 7 2

Execution Example



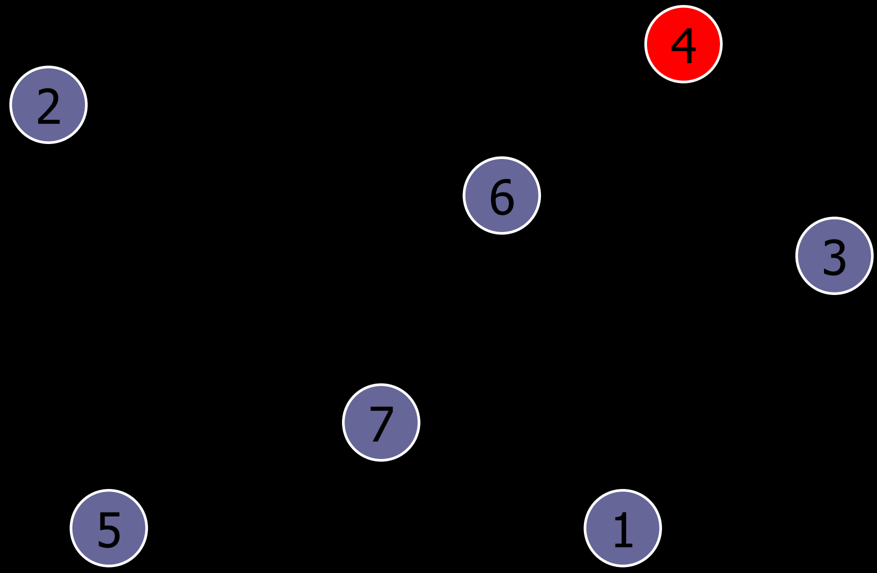
- Stack: 1 4 2
- Location: 5
- Circuit: 1 5 7 6 4 3 7 2 6

Execution Example



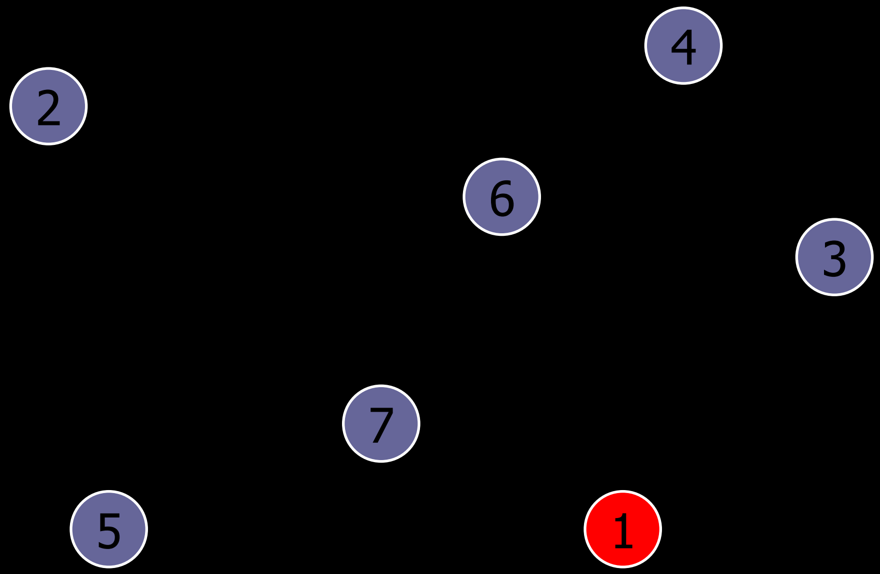
- Stack: 1 4
- Location: 2
- Circuit: 1 5 7 6 4 3 7 2 6 5

Execution Example



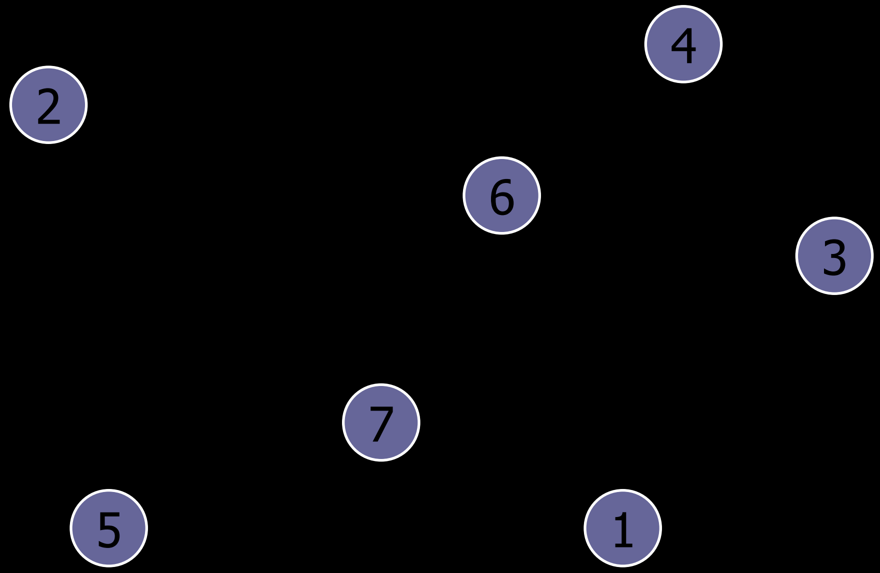
- Stack: 1
- Location: 4
- Circuit: 1 5 7 6 4 3 7 2 6 5 2

Execution Example



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

Execution Example



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

Analysis

- To find an Eulerian path, find one of the nodes which has odd degree (or any node if there are no nodes with odd degree) and call find_circuit with it.
- This algorithm runs 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.