



# 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)

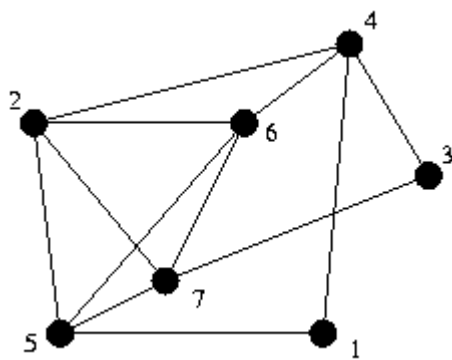
    if node i has no neighbors then
        circuit(circuitpos) = node i
        circuitpos = circuitpos + 1
    else
        while (node i has neighbors)
            pick a random neighbor node j of node i
            delete_edges (node j, node i)
            find_circuit (node j)
        circuit(circuitpos)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

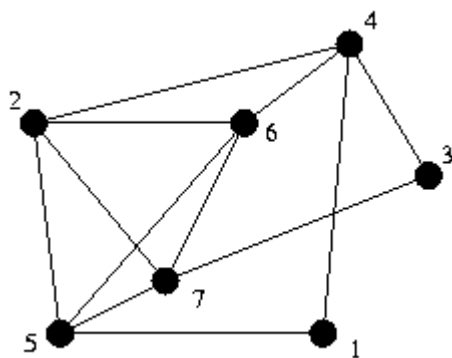
Both of these algorithms run in  $O(m + n)$  time, where  $m$  is the number of edges and  $n$  is the number of nodes, if you store the graph in adjacency list form. With larger graphs, there's a danger of overflowing the run-time stack, so you might have to use your own stack.

### Execution Example

Consider the following graph:

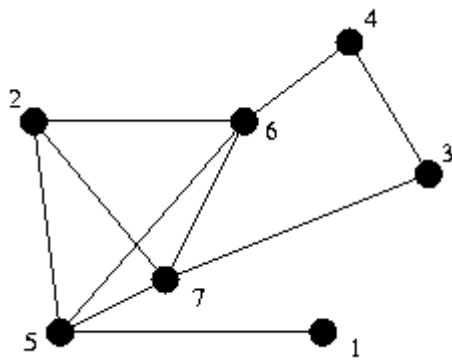
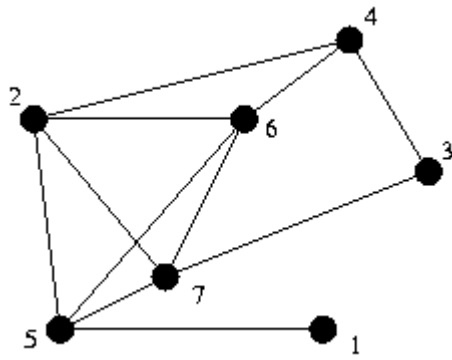


Assume that selecting a random neighbor yields the lowest numbered neighbor, the execution goes as follows:

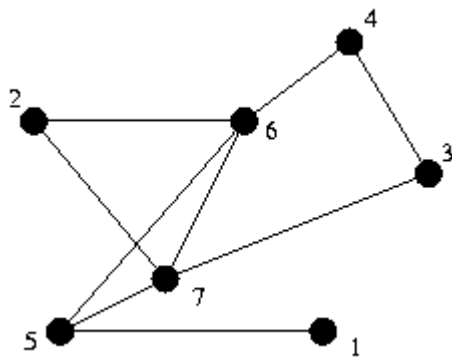


Stack:  
Location: 1  
Circuit:

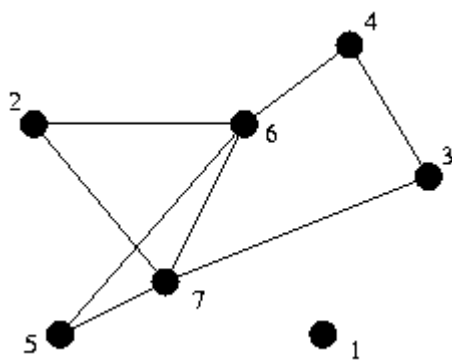
Stack: 1  
Location: 4  
Circuit:



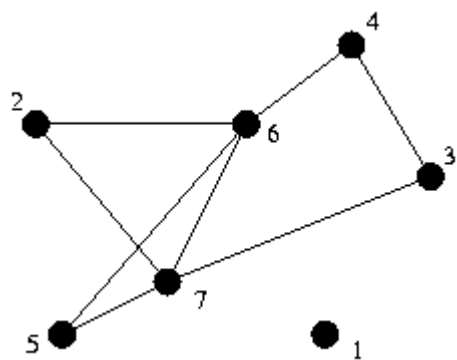
Stack: 1 4  
Location: 2  
Circuit:



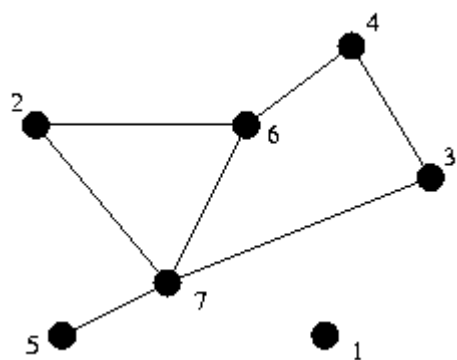
Stack: 1 4 2  
Location: 5  
Circuit:



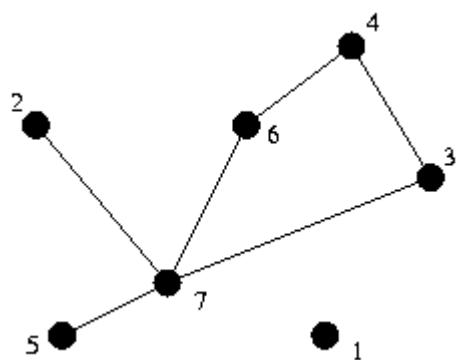
Stack: 1 4 2 5  
Location: 1  
Circuit:



Stack: 1 4 2  
Location: 5  
Circuit: 1

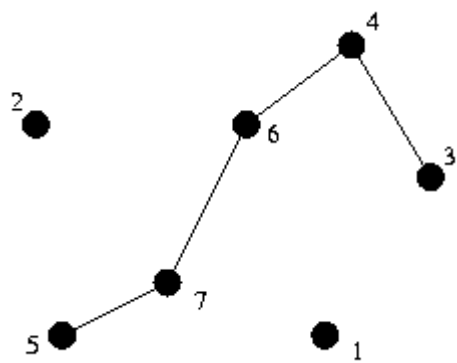
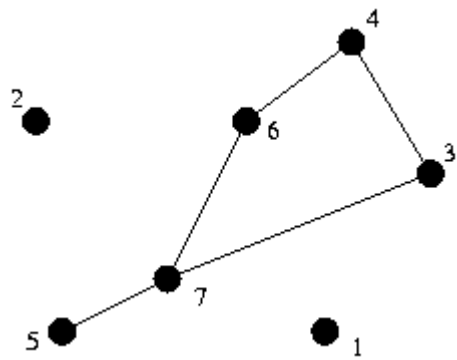


Stack: 1 4 2 5  
Location: 6  
Circuit: 1

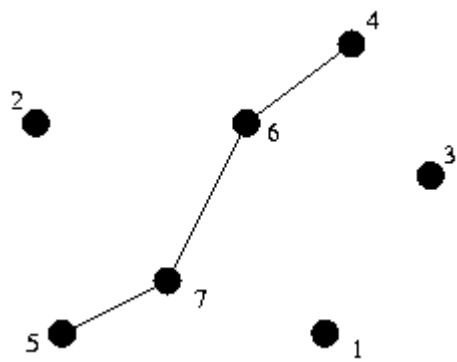


Stack: 1 4 2 5 6  
Location: 2  
Circuit: 1

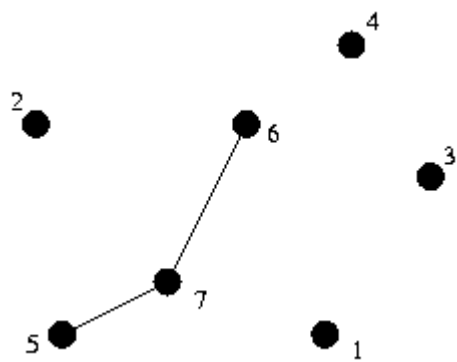
Stack: 1 4 2 5 6 2  
Location: 7  
Circuit: 1



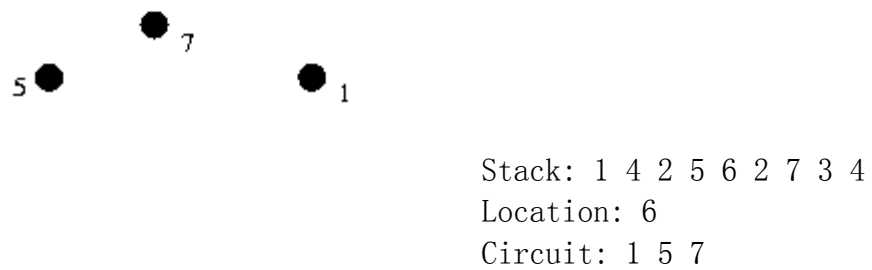
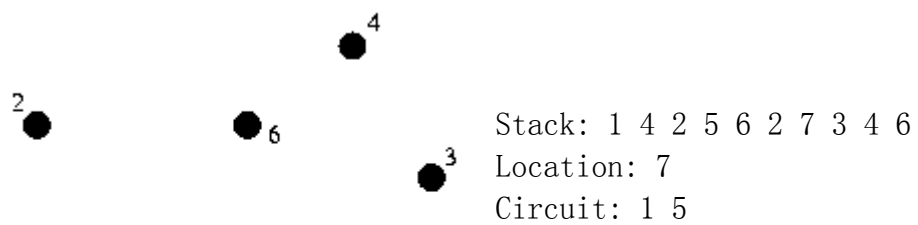
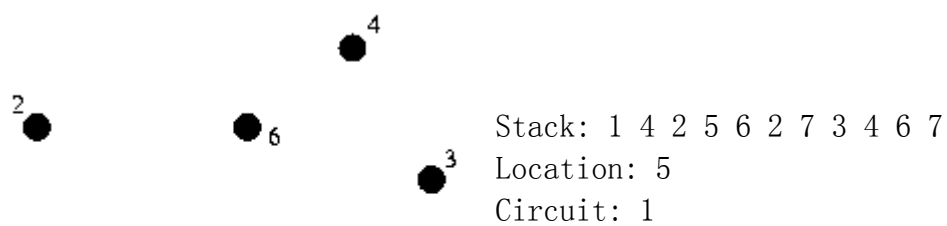
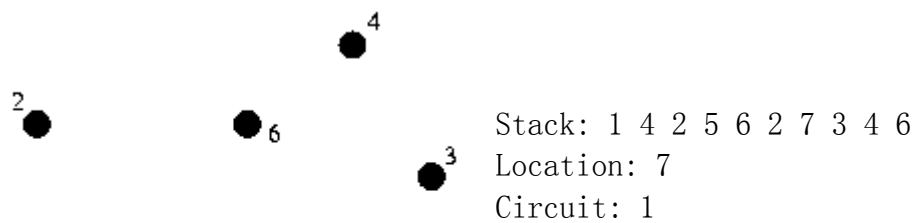
Stack: 1 4 2 5 6 2 7  
Location: 3  
Circuit: 1

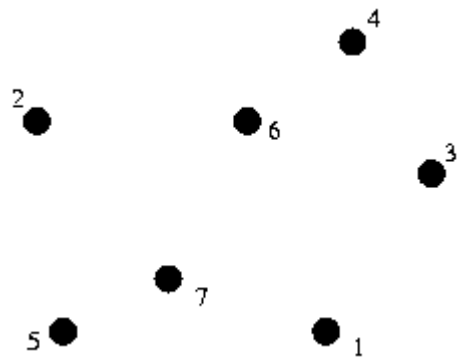


Stack: 1 4 2 5 6 2 7 3  
Location: 4  
Circuit: 1

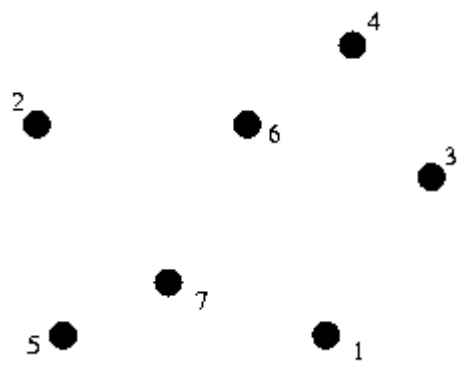


Stack: 1 4 2 5 6 2 7 3 4  
Location: 6  
Circuit: 1

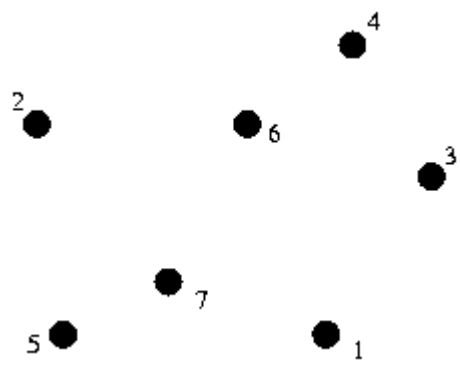
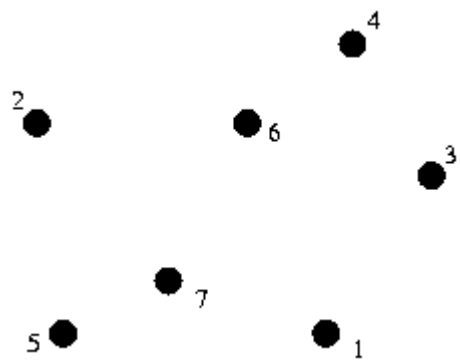




Stack: 1 4 2 5 6 2 7 3  
Location: 4  
Circuit: 1 5 7 6

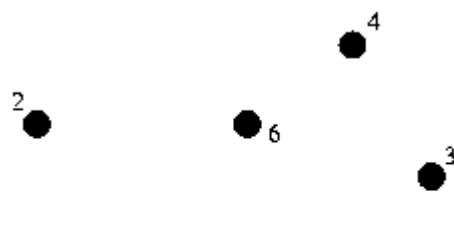


Stack: 1 4 2 5 6 2 7  
Location: 3  
Circuit: 1 5 7 6 4

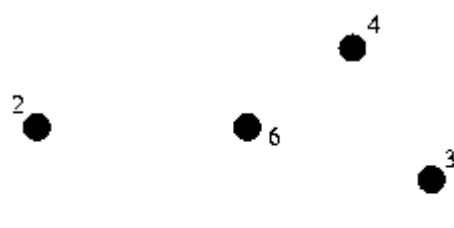


Stack: 1 4 2 5 6 2  
Location: 7  
Circuit: 1 5 7 6 4 3

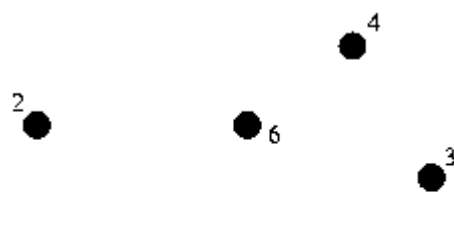




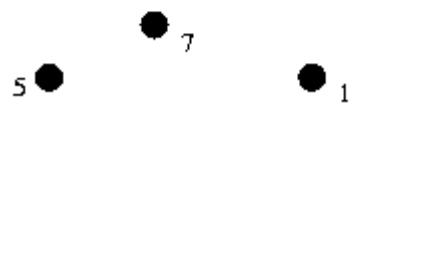
Stack: 1 4 2 5 6  
 Location: 2  
 Circuit: 1 5 7 6 4 3 7



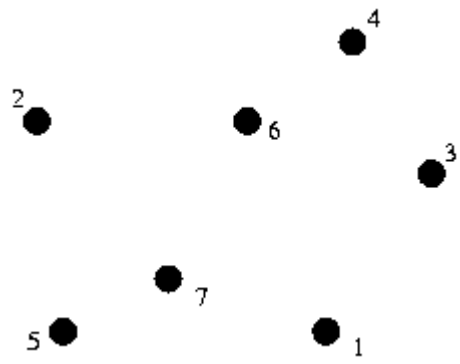
Stack: 1 4 2 5  
 Location: 6  
 Circuit: 1 5 7 6 4 3 7 2



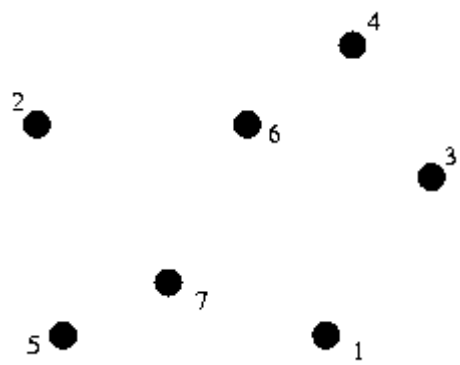
Stack: 1 4 2  
 Location: 5  
 Circuit: 1 5 7 6 4 3 7 2 6



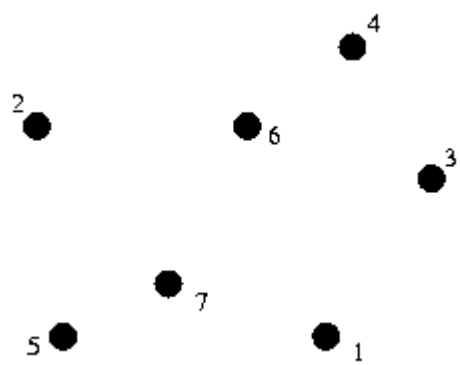
Stack: 1 4  
 Location: 2  
 Circuit: 1 5 7 6 4 3 7 2 6 5



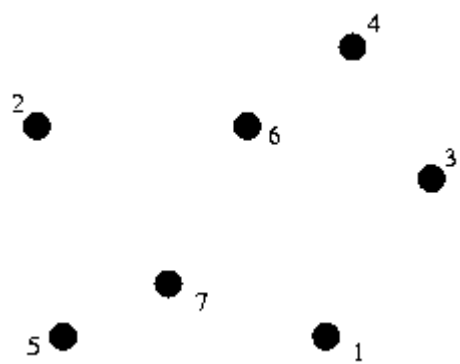
Stack: 1  
 Location: 4  
 Circuit: 1 5 7 6 4 3 7 2 6 5 2



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  $N$  ( $2 \leq N \leq 15$ ), 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.

## Eulerian Tour

### 欧拉路径

译 by 孖哥 and tim green

#### Sample Problem: Riding The Fences

农民 John 每年有很多栅栏要修理。他总是骑着马穿过每一个栅栏并修复它破损的地方。

John 是一个与其他农民一样懒的人。他讨厌骑马，因此从来不两次经过一个一个栅栏。你必须编一个程序，读入栅栏网络的描述，并计算出一条修栅栏的路径，使每个栅栏都恰好被经过一次。John 能从任何一个顶点(即两个栅栏的交点)开始骑马，在任意一个顶点结束。

每一个栅栏连接两个顶点，顶点用 1 到 500 标号(虽然有的农场并没有 500 个顶点)。一个顶点上可连接任意多( $\geq 1$ )个栅栏。所有栅栏都是连通的(也就是你可以从任意一个栅栏到达另外的所有栅栏)。

你的程序必须输出骑马的路径(用路上依次经过的顶点号码表示)。我们如果把输出的路径看成是一个 500 进制的数，那么当存在多组解的情况下，输出 500 进制表示法中最小的一个(也就是输出第一个数较小的，如果还有多组解，输出第二个数较小的，等等)。

输入数据保证至少有一个解。

#### The Abstraction

已知:一幅无向图

寻找一条只过每边一次的路径. 这条路径叫欧拉路径(Eulerian tour). 如果此路径和起点和终点是同一点, 则此种路径叫欧拉回路.

#### The Algorithm

判断一幅图有没有欧拉路径或欧拉回路是很简单, 有两个不同的规则可用.

- 当且仅当一幅图是相连的(只要你去掉所有度数为 0 的点)且每个点的度都是偶数, 这幅图有欧拉回路.
- 当且仅当一幅图是相连的且除两点外所有的点的度都是偶数.
- 在第二种情况中, 那两个度为奇数的节点一个为起点, 剩下的一个是终点.

一个解决此类问题基本的想法是从某个节点开始, 然后查出一个从这个点出发回到这个点的环路径. 现在, 环已经建立, 这种方法保证每个点都被遍历. 如果有某个点的边没有被遍历就让这个点为起点, 这条边为起始边, 把它和当前的环衔接上. 这样直至所有的边都被遍历. 这样, 整个图就被连接到一起了.

更正式的说, 要找出欧拉路径, 就要循环地找出出发点. 按以下步骤:

- 任取一个起点, 开始下面的步骤
  - 如果该点没有相连的点, 就将该点加进路径中然后返回.
  - 如果该点有相连的点, 就列一张相连点的表然后遍历它们直到该点没有相连的点。(遍历一个点, 删除一个点)
  - 处理当前的点, 删除和这个点相连的边, 在它相邻的点上重复上面的步骤, 把当前这个点加入路径中.

下面是伪代码:

```
# 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)

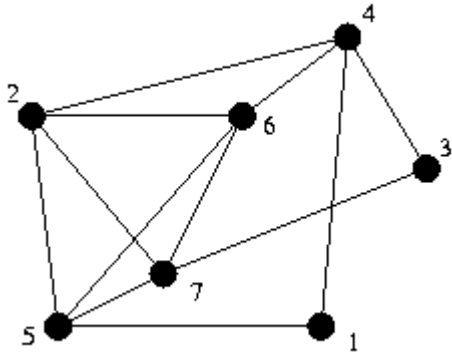
    if node i has no neighbors then
      circuit(circuitpos) = node i
      circuitpos = circuitpos + 1
    else
      while (node i has neighbors)
        pick a random neighbor node j of node i
        delete_edges (node j, node i)
        find_circuit (node j)
      circuit(circuitpos) = node i
    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

要找欧拉路径, 只要简单的找出一个度为奇数的节点, 然后调用 `find_circuit` 就可以了.

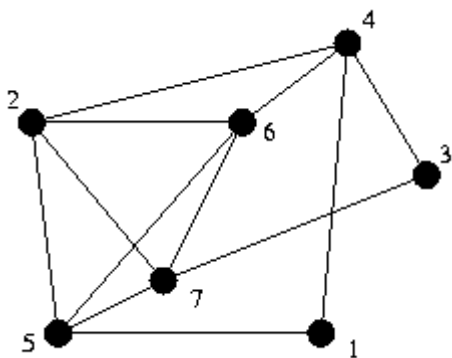
这两个算法的效率都是  $O(m + n)$ ,  $m$  是边数,  $n$  是节点数, 如果你用邻接表来储存图, 有可能会使程序堆栈溢出, 所以你要用自己的堆栈.

### Execution Example

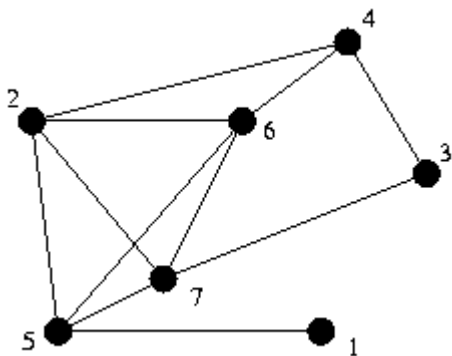
考虑下面这个图:



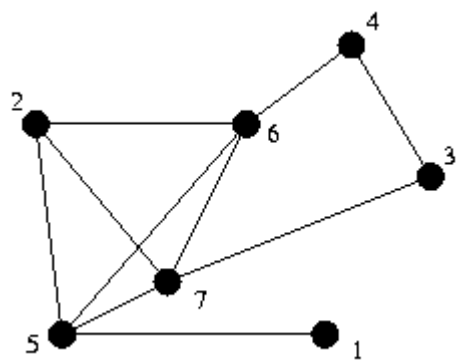
假定随机选择相邻的节点时从编号最小的开始, 下面是执行的结果:



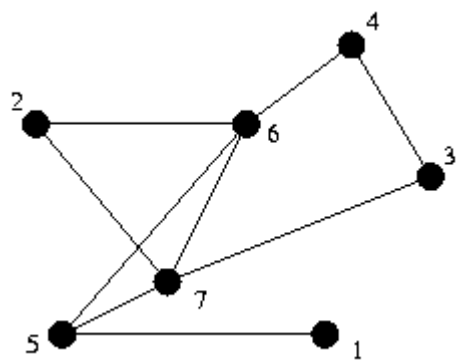
Stack:  
Location: 1  
Circuit:



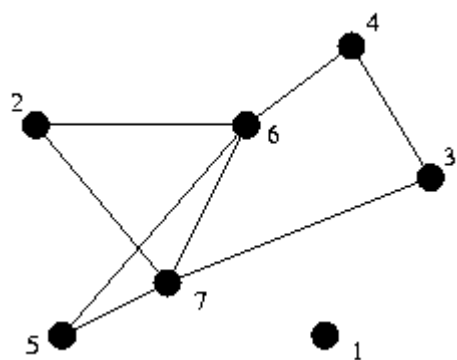
Stack: 1  
Location: 4  
Circuit:



Stack: 1 4  
 Location: 2  
 Circuit:

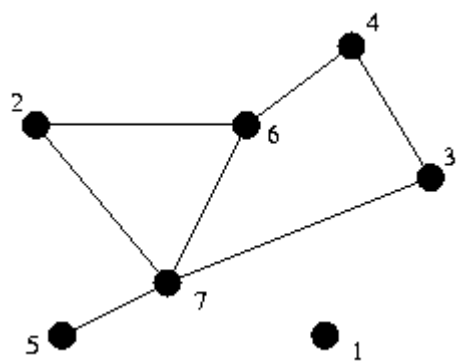
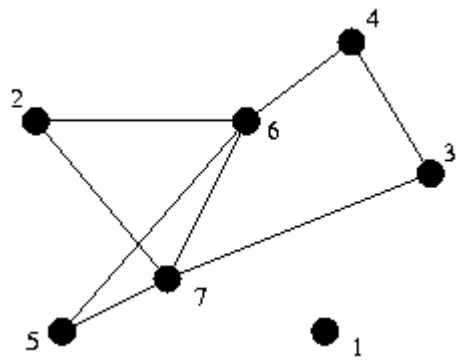


Stack: 1 4 2  
 Location: 5  
 Circuit:

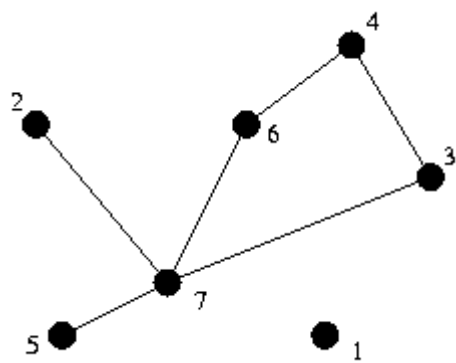


Stack: 1 4 2 5  
 Location: 1  
 Circuit:

Stack: 1 4 2  
 Location: 5  
 Circuit: 1



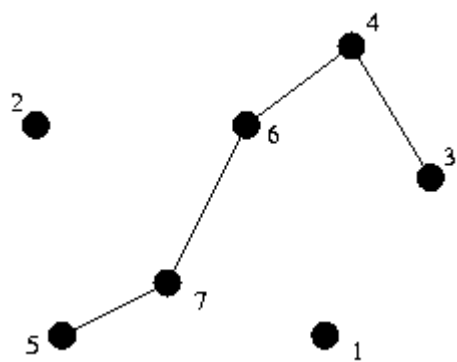
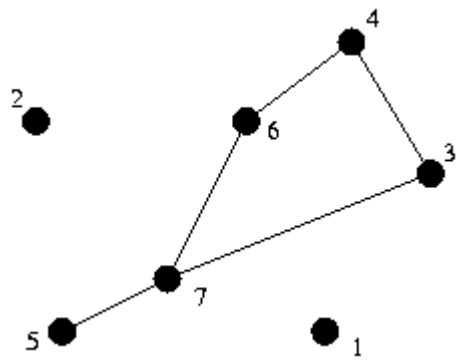
Stack: 1 4 2 5  
Location: 6  
Circuit: 1



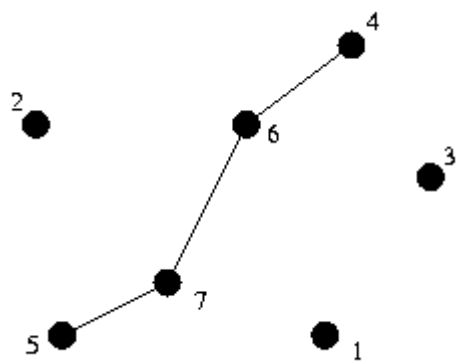
Stack: 1 4 2 5 6  
Location: 2  
Circuit: 1

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



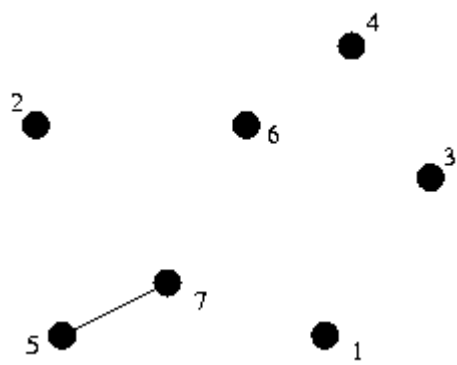
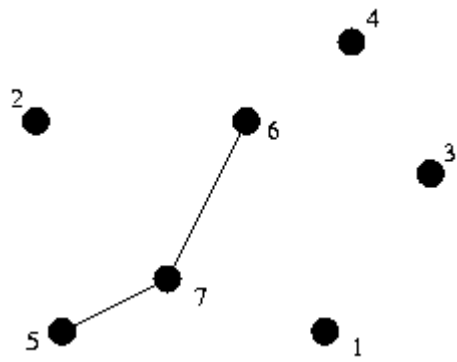


Stack: 1 4 2 5 6 2 7  
Location: 3  
Circuit: 1

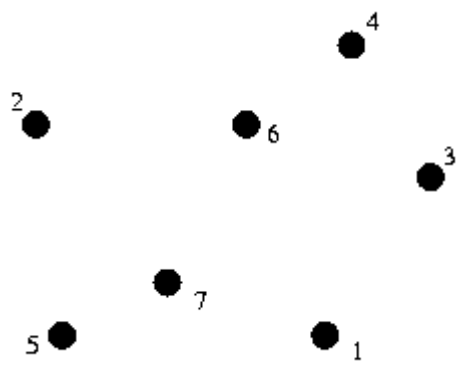


Stack: 1 4 2 5 6 2 7 3  
Location: 4  
Circuit: 1

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

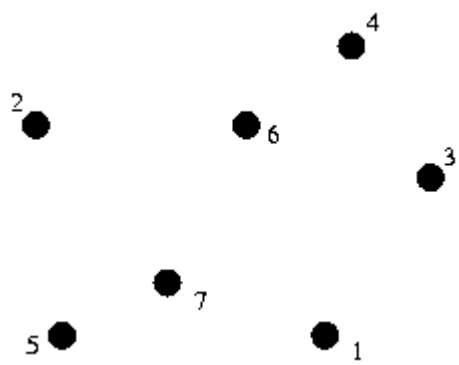
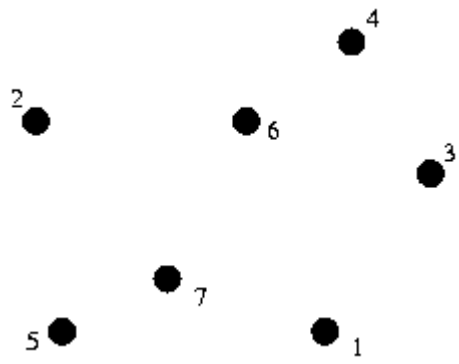


Stack: 1 4 2 5 6 2 7 3 4 6  
 Location: 7  
 Circuit: 1

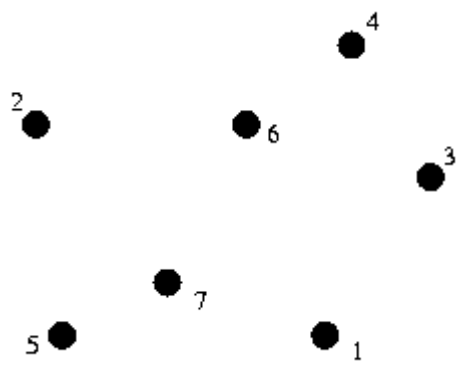


Stack: 1 4 2 5 6 2 7 3 4 6 7  
 Location: 5  
 Circuit: 1

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

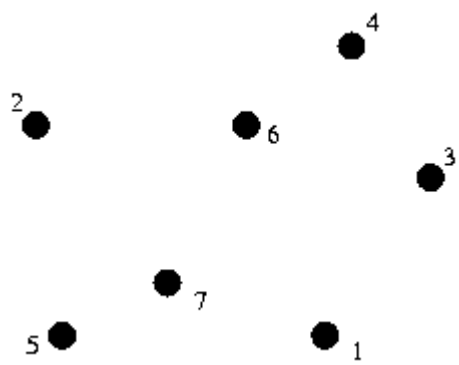
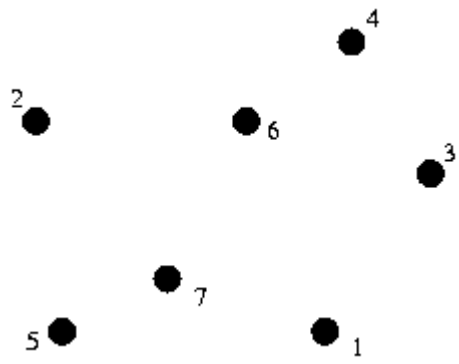


Stack: 1 4 2 5 6 2 7 3 4  
 Location: 6  
 Circuit: 1 5 7

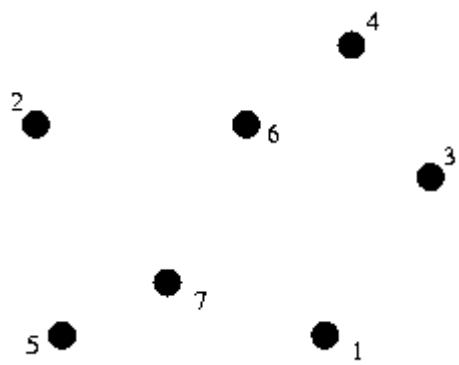


Stack: 1 4 2 5 6 2 7 3  
 Location: 4  
 Circuit: 1 5 7 6

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

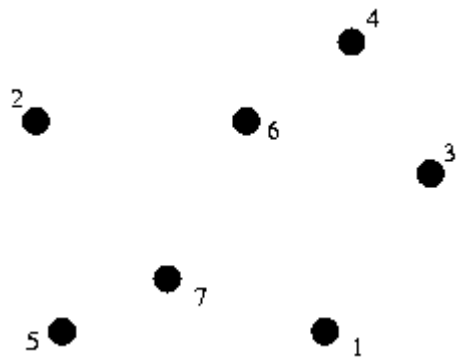


Stack: 1 4 2 5 6 2  
Location: 7  
Circuit: 1 5 7 6 4 3

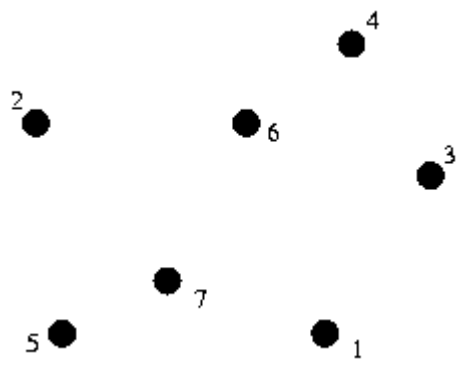


Stack: 1 4 2 5 6  
Location: 2  
Circuit: 1 5 7 6 4 3 7

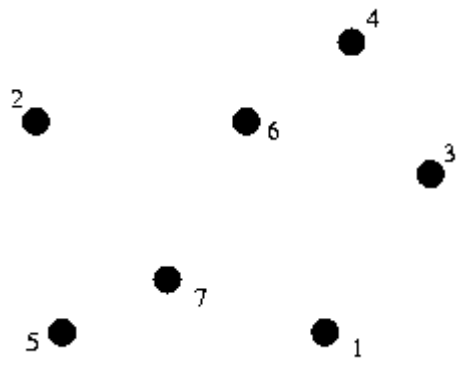
Stack: 1 4 2 5  
Location: 6  
Circuit: 1 5 7 6 4 3 7 2



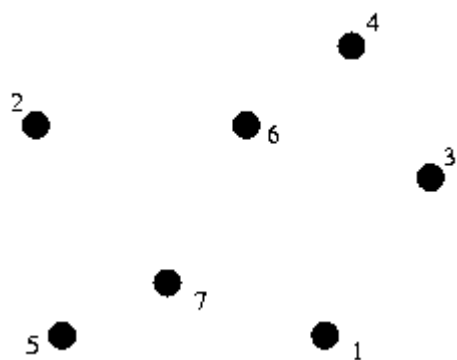
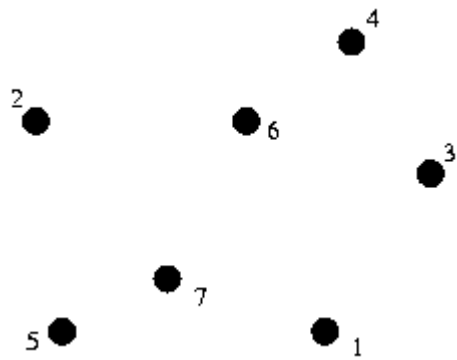
Stack: 1 4 2  
Location: 5  
Circuit: 1 5 7 6 4 3 7 2 6



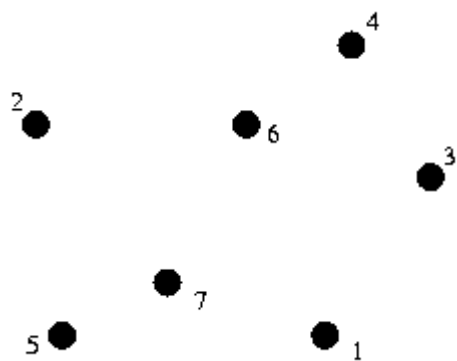
Stack: 1 4  
Location: 2  
Circuit: 1 5 7 6 4 3 7 2 6 5



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

两个点之间有多条边的图,也可以使用相同的算法。

有自环的图也可以使用这样的算法,前提是我们自环给这个节点增加的度为 2.

如果一个有向图是强连通的（不考虑入度出度都是 0 的点）并且每个点的入度等于出度。那么这个图有欧拉回路而这个算法仍然适用,但是你要记得输出时应该从最后一个开始。

在有向图中寻找一个欧拉路径是十分困难的。

## Example problems

### Airplane Hopping

#### 忙碌的飞机

给出地图上的城市，在一些城市之间有航线，请确定是否存在一条路线,使得飞机飞遍每一条航线又回到它出发的城市。

分析：这等价于在一个有向图中找欧拉回路。

### Cows on Parade

#### 奶牛游行

农民 John 有两种牛：黑色的 Angus 和白色的 Jerseys. While marching John 的妻子 Joanne 送 19 只母牛去集市时，她注意到 4 只母牛排要一起的全部的 16 种情况(e. g., bbbb, bbbw, bbwb, bbww, ..., wwww)。当然，一些情况和另一些是有重叠的.

给出  $N$  ( $2 \leq N \leq 15$ )，找出长度最短的一个母牛序列使得  $N$  只母牛排在一起的所有情况在这个序列中都出现过。

分析：这个图中的顶点是  $N-1$  只母牛排在一起的所有情况. [Being at a node corresponds to the last  $N-1$  cows matching the node in color.] 当  $N = 4$ , 如果最后 3 只母牛是 *wbw*, 那么你就停在节点 *wbw*。因为可能相应的增加 *w* 或 *b* 到序列的最后，所以每个点的出度为 2。同样，可能相应的增加 *w* 或 *b* 到序列的开始, 所以每个点的入度为也 2。

这个图是强连通的, 并且入度等于出度，所以存在欧拉回路.。

欧拉回路对应的序列是欧拉回路中第一个点对应的  $N-1$  序列加上后面边的颜色。