Solution:

1. Devise an algorithm for constructing Euler circuits in directed graphs.
   
   Suppose the given directed graph does have a Euler circuit.
   
   We can slightly modify the DFS algorithm in the lecture to find the Euler Circuit. In the original DFS, we mark each node whether we have visited, but in our new DFS algorithm DFS.EDGE, we mark the edges and examine whether we have visited the edge every time we find a new edge to traverse. Then, we run DFS.EDGE starting from any node until we need to backtrack. At this point, we have found a circuit.

   **Algorithm 1 Euler Circuit Search**
   
   1: Pick any vertex in the graph to run DFS.EDGE until we need to backtrack. When we need to backtrack, we have found a circuit. Store the circuit as the CURRENT.CIRCUIT.
   2: Pick the first vertex along the CURRENT.CIRCUIT that has an outgoing edge not visited. Run DFS.EDGE starting with that vertex until we need to backtrack. At this point, we have found another circuit.
   3: Splice the newly found circuit with CURRENT.CIRCUIT.
   4: Repeat 2,3 until all edges are visited.
   5: Return CURRENT.CIRCUIT.

2. Describe the trees produced by breadth-first search and depth-first search of the complete graph $K_n$, where $n$ is a positive integer. Justify your answers.

   Since every vertex is connected to every other vertex in $K_n$, the BFS will construct a tree $K_{1,n-1}$ with every vertex adjacent to the starting vertex. The DFS will produce a simple path of length $n-1$ for the same reason.

3. Show that an edge with smallest weight in a connected weighted graph must be part of any minimum spanning tree.

   We prove this by contradiction. Suppose $e$ is the minimum-weight edge in the connected weighted graph and assume that there exists a minimum spanning tree MST who does not contain $e$. Since MST is a tree, adding any extra edge to it will produce a simple cycle. Then, if we add $e$ to MST, MST $\cup \{e\}$ is a graph having a cycle. Then, we can remove any other edge in the cycle to make it a tree, which is also a spanning tree. Since we added $e$ and removed another edge whose weight is greater than $e$, the newly created spanning tree has a smaller total weight, which contradicts our first assumption that MST has the minimum total weight. Therefore, it is not possible that the minimum spanning tree does not contain $e$.

4. Prove that if a connected undirected graph has no vertex of degree 1, then it contains a cycle.

   We again prove this statement by contradiction. Suppose a connected graph $G$ does not have a 1-degree vertex and assume that the graph does not have a cycle. Since $G$ does not have a cycle, it must be a
Every tree has at least one leaf node whose degree is 1, which contradicts our first condition that the graph does not have any vertex with degree 1. Therefore, any connected graph that has no vertex of degree 1 must contain a cycle.

5. (a) If there is an Euler circuit, \(P\), in \(G\). Every time a vertex is listed, that accounts for two edges adjacent to that vertex, the one before it in the list and the one after it in the list. This circuit uses every edge exactly once. So every edge is accounted for and there are no repeats. Thus every degree must be even.

Suppose every degree is even. We will show that there is an Euler circuit by induction on the number of edges in the graph. The base case is for a graph \(G\) with two vertices with two edges between them. This graph is obviously Eulerian. Suppose we have a graph \(G\) on \(k > 2\) edges. We start at an arbitrary vertex \(v\) and follow edges, arbitrarily selecting one after another until we return to \(v\). Call this trail \(C\). We know that we will return to \(v\) eventually because every time we encounter a vertex other than \(v\) we are listing one edge adjacent to it. There are an even number of edges adjacent to every vertex, so there will always be a suitable unused edge to list next. So this process will always lead us back to \(v\). Let \(E\) be the edges of \(C\). The graph \(G - E\) has components \(G_1, G_2, \ldots, G_k\). These each satisfy the induction hypothesis: connected, less than \(k\) edges, and every degree is even. We know that every degree is even in \(G - E\), because when we removed \(C\), we removed an even number of edges from those vertices listed in the circuit. By induction, each circuit has an Eulerian circuit, call them \(C_1, C_2, \ldots, C_k\). Since \(G\) is connected, there is a vertex \(a_i\) in each component \(C_i\) on both \(C\) and \(C_i\). Assume the vertices \(a_1, a_2, \ldots, a_k\) are visited in that order. We describe an Euler circuit in \(G\) by starting at \(v\) follow \(C\) until reaching \(a_1\), follow the entire \(G_1\) ending back at \(a_1\), follow \(W\) until reaching \(a_2\), follow the entire \(G_2\), ending back at \(a_2\) and so on. End by following \(W\) until reaching \(a_k\), follow the entire \(G_k\), ending back at \(a_k\), then finish off \(C\), ending at \(v\).

(b) The idea of algorithm is same as (a). It involves following steps:
   i. Start with an empty stack and an empty Eulerian cycle. If all vertices have even degree, choose any of them. Otherwise no Eulerian cycle. This takes \(O(V)\) time.
   ii. If current vertex has no neighbors, add it to circuit, remove the last vertex from the stack and set it as the current one. Otherwise (in case it has neighbors), add the vertex to the stack, take any of its neighbors, remove the edge between selected neighbor and that vertex, and set that neighbor as the current vertex.
   iii. Repeat step ii until the current vertex has no more neighbors and the stack is empty.

   Since we visit all edges in graph once, which should be \(O(|E|)\).

In the following, we use Python to implement the algorithm, which takes \(O(|V| + |E|)\)

```python
""
DFS to find an Eulerian cycle in an undirected graph
""
import copy
def findEulerianCycle(graph):
    graphcopy = copy.deepcopy(graph)
    if not graph:
        print("ERROR: graph is empty")
        return
    totaledges = 0  # total number of edges in graph
    edgecntr = {v: len(adjvets) for vet, adjvets in enumerate(graph)}
    for vet, adjvets in enumerate(graph):
        if not adjvets:
            print(’ERROR: vertex %d is disconnected’ % vet)
```
```python
return
if len(adjvets) & 1:
    print('ERROR: degree of vertex %d is odd' %vet)
return
totaledges += len(adjvets)
edgecnt[vet] = len(adjvets)
totaledges //= 2
stack = [0]  # maintain a stack to keep vertices
curvet = 0  # current vertex
cycle = []  # final Eulerian cycle
edgevisited = set()
while stack:
    if edgecnt[curvet] != 0:
        stack.append(curvet)
        nextvet = graphcopy[curvet][-1]
        # pop out adjacent vertices whose edge has been visited
        while graphcopy[curvet] and (min(curvet, nextvet), max(curvet, nextvet))
        ) in edgevisited:
            graphcopy[curvet].pop()
            nextvet = graphcopy[curvet][-1]
            graphcopy[curvet].pop()
            # mark all visited edge as visited
            edgevisited.add((min(curvet, nextvet), max(curvet, nextvet)))
        # pop out adjacent vertices whose edge has been visited
        # because in total one edge will be at most be added in edgevisited for
        # once, so the time complexity for these two while is O(E)
        while graphcopy[nextvet] and (min(graphcopy[nextvet][-1], nextvet), max(  
            graphcopy[nextvet][-1], nextvet)) in edgevisited:
            graphcopy[nextvet].pop()

        edgecnt[curvet] -= 1
        edgecnt[nextvet] -= 1
        curvet = nextvet
    else:
        cycle.append(curvet)
        curvet = stack.pop()
for i in range(len(graph)):
    if i not in set(cycle):
        print('ERROR: vertex %d is disconnected' %i)
        return
if totaledges != len(cycle)-1:
    print('ERROR: No Eulerian cycle exists')
    return
return ','.join(map(str, cycle))
```

Code 1: DFS to find an Eulerian cycle in an undirected graph