Exercise 4. Note that the argument (2e >= 3f, e being edges, f being faces in a simple planar graph) only works for v >= 3. What goes wrong if v <= 2?

We consider the following cases:

[Case 1: v = 0 or v = 1] Then the graph has e = 0 or f = 1, the single face has 0 sides, violating the argument's claim (2(0) >= 3(1)).

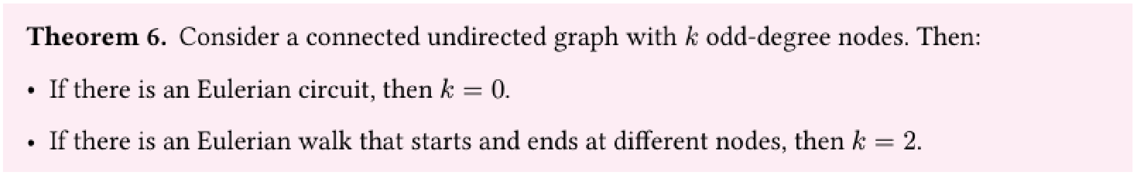
[Case 2: v = 2] Then the simple connected graph has e = 1 and f = 1. The single face is bounded by the edge counted twice, giving it two sides, not 3, so (2 >= 3) is false.

Thus, the counting argument and the resulting bounds (e <= 3v - 6, f <= 2v - 4, e <= 2v - 4) do not hold for these cases.

Exercise 7. Note that the connectedness requirement is important for the Eulerian circuit/walk. Why?

The connectedness requirement is important because an Eulerian circuit or walk must traverse every edge in the graph. If the graph is not connected, edges in a separate component cannot be reached from the starting point, making a single continuous walk covering all edges impossible.

Furthermore, the degree conditions (k = 0 or k = 2) are necessary but not sufficient if isolated vertices or separate components exist.

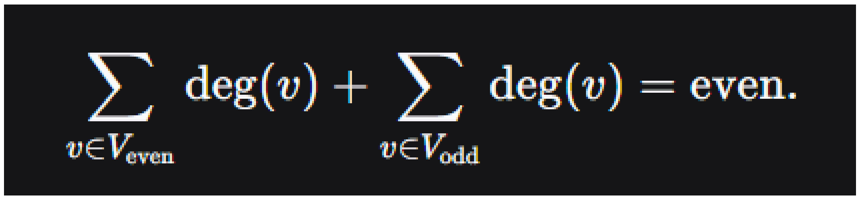


Theorem 6. Consider a connected undirected graph with k odd-degree nodes. Then:

- If there is an Eulerian circuit, then $k = 0$.
- If there is an Eulerian walk that starts and ends at different nodes, then $k = 2$.

Exercise 8. Note that the above enumeration is exhaustive, because k must always be even. Prove this fact.

By the Handshaking Lemma, the sum of all vertices' degrees is equal to 2e, which is even. We split the sum into even-degree vertices and odd-degree vertices:



$$\sum_{v \in V_{\text{even}}} \deg(v) + \sum_{v \in V_{\text{odd}}} \deg(v) = \text{even}.$$

The first term is even since it is the sum over even-degree vertices. Therefore, the second term must also be even for their sum to be even.

This second term is the sum of k odd numbers. A sum of odd numbers is even if and only if k is even. Hence, k, the number of odd-degree vertices, must be even.

Exercise 11: Suppose you have an O(f(n, e)) algorithm that computes the Eulerian circuit for any connected graph with k = 0. Find an O(f(n, e + 1)) algorithm that computes an Eulerian walk for any connected graph with k = 2. (Hint: Add a single edge.)

- An Eulerian walk is essentially an Eulerian circuit with one edge "missing".
- If we add an edge between the two nodes with odd degrees, they become even because their degrees are incremented by 1.
    - All other vertices remain even

We convert the Eulerian circuit algorithm into an Eulerian walk algorithm as follows:
- Let G be a connected graph with exactly two odd-degree vertices u and v.
- Add a new edge (u, v) to G, forming G'. Now G' has all vertices of even degree and e + 1 edges.
- Run the given Eulerian circuit algorithm on G' to obtain an Eulerian circuit C in O(f(n, e + 1)) time.
- Locate the added edge (u,v) in C and remove it.
- The resulting sequence is an Eulerian walk from one endpoint of the removed edge to the other, using every edge of G exactly once.

Exercise 12: Suppose you have an O(f(n, e)) algorithm that computes the Eulerian walk for any connected graph with k = 2. Find an O(f(n + 1, e + 1)) algorithm that computes an Eulerian circuit for any connected graph with k = 0. (Hint: Add a node and edge.)

We convert the Eulerian walk algorithm into an Eulerian circuit algorithm as follows:
- Let G be a connected graph with all vertices of even degree, k = 0.
- Pick any vertex x $\in$ V(G). Add a new vertex x' and a new edge (x, x') to form G'.
    - G' has n + 1 vertices, e + 1 edges, and exactly two odd-degree vertices (x and x').
- Run the given Eulerian walk algorithm on G' from x to x', obtaining walk W in O(f(n + 1, e + 1)) time.
- Since x' has degree 1, the walk must end with the edge (x, x'). Remove this final edge and vertex x' from W.
- The resulting sequence is an Eulerian circuit in G, starting and ending at x.

Exercise 13: Suppose all nodes have even degree. Prove that performing just_walk(x,x) on any node x is guaranteed to succeed, i.e., it will end up at node x.

Consider any point during the walk when we are at a vertex v that is not x. We must have arrived at v via some edge, consuming one incident edge. Since deg(v) is even, the number of remaining unused incident edges is now odd. An odd number cannot be 0, so there is at least one unused edge to leave by. We must then leave v, consuming another edge. Therefore, we can never get stuck at v != x.

The only place where getting stuck is possible is the start vertex x. Since the graph is finite, the walk must eventually terminate, with the only possible termination point being at x. Hence, just_walk(x,x) always ends at x.

Exercise 14. Suppose a graph has no odd-degree nodes. Prove that removing any edge yields a graph with at most 2 odd-degree nodes.

Let G be a graph with all vertices of even degree. Remove any edge e. Consider the two endpoints of e, call them u and v. Consider these two cases:

[Case 1: u = v] Then removing the edge (u,v) would still lead to u/v having an even degree. All vertices remain to have an even degree. Therefore, 0 odd-degree vertices.

[Case 2: u != v] Then removing the edge (u, v) would lead to both u and v having odd degrees as (even - 1 = odd). The degrees of the remaining vertices remain unchanged, so we have exactly 2 odd-degree vertices.

In both cases, the resulting graph has at most 2 odd-degree vertices.

Exercise 15. Suppose a graph has 2 odd-degree nodes *x* and *y*. Prove that removing any edge incident to *x* yields a graph with at most 2 odd-degree nodes.

Let G be a graph containing the two odd-degree nodes x and y, with all other vertices having even degree. We consider the following cases:

[Case 1: Remove edge (x,y)] Since both x and y have odd degrees, removing an edge between them would result in both having an even degree. Hence, there would be 0 odd-degree vertices.

[Case 2: Remove edge (x, z) with z != y] Consider another node z with even degree in graph G, incident to x. If we remove an edge between x and z, then the degree of x would become even and the degree of z would become odd. With y and z, the resulting graph G’ has at most 2 odd-degree nodes.

In both cases, the resulting graph has at most 2 odd-degree vertices.

Exercise 16. Prove that at any point in Fleury's algorithm, x is incident to at most one bridge.

- **(Hint: Let (x, y) and (x, z) be two bridges. Remove them, yielding three connected components, and then consider the sum of the degrees of the resulting components.)**
- **(Hint 2: Note that there is at most one other odd-degree node other than x.)**

Suppose, for contradiction, that x is incident to two bridges (x, y) and (x, z) with y != z. Remove both edges, yielding three components: A (containing x), B (containing y), and C (containing z).

In any graph, each component has an even number of odd-degree vertices. Let o_A, o_B, o_C be the number of odd vertices in each component in the current graph, with each being even.

The total number of odd vertices in the whole graph is at most 2, by properties of Eulerian graphs, and one of them could be x. So, $o_B + o_C <= 1$.

But o_B and o_C are even, so the only possibility is $o_B = o_C = 0$. Thus, all vertices in B and C have even degree.

Consider y in B. Its degree in the whole graph is $\deg(y) = \deg_B(y) + 1$ from the bridge. Since deg(y) is even, $\deg_B(y)$ is even, $\deg_B(y)$ must be odd. Thus, y has odd degree within B. But then B has at least one odd vertex (y) which is a contradiction since $o_B = 0$.

Therefore, x cannot have two bridges.

Exercise 17. Using Exercise 16, prove that Fleury's algorithm correctly gives an Eulerian walk.

Hint: Prove that there's a non-bridge incident to x at any point in the algorithm, except when x is a "leaf".

We prove by induction on the number of edges. At any step, let x be the current vertex.

By Exercise 16, x has at most one bridge. If deg(x) >= 2, at least one incident edge is a non-bridge, otherwise x would have >= 2 bridges. Thus, the algorithm can always choose a non-bridge unless x is a leaf, in which case the only edge is taken.

Removing a non-bridge preserves connectivity, and removing any edge preserves the "at most 2 odd vertices" property. By induction, the remaining graph has an Eulerian walk from the new vertex to the target. Prepending the taken edge yields an Eulerian walk for the original graph.

The algorithm never gets stuck and uses all edges, ending at the correct target vertex.

[Base Case: m = 0] Then, there are no edges, the walk is trivial. If x != y, the graph must have had 2 odd vertices initially, but with no edges, all vertices have degree 0, so x = y.

[Inductive Step] Assume Fleury's algorithm works for all graphs with fewer than m edges. Consider a graph G with m edges, starting at x.

By the hint, at the current vertex x:

- If deg(x) = 1, take the only edge (which is a bridge).
- If deg(x) >= 2, there exists a non-bridge edge incident to x. Choose such an edge e = (x,z).

Remove e from G to get G', and remove x if it becomes isolated.

[Claim 1: G' is connected, or has the right connectivity for the remaining walk]

- If e was a non-bridge, removing it wouldn't have disconnected the graph.
- If e was a bridge and deg(x) = 1, removing it disconnects the graph, but the component containing z is the only one with edges left. So, the walk continues from z.

[Claim 2: G' has at most 2 odd vertices, and if it has 2 odd vertices, they are the current vertex z and the target y (or z = y for a circuit)]

- By the inductive hypothesis, Fleury's algorithm can complete an Eulerian walk from z to y in G'.
- Prepending e to that walk gives an Eulerian walk from x to y in G.

Thus, by induction, Fleury's algorithm always produces an Eulerian walk.

Exercise 18: Prove that Hierholzer's algorithm, as implemented above, correctly gives an Eulerian walk.

We prove correctness by induction on the number of edges k. The algorithm maintains that all vertices have even degree throughout (removing cycles preserves evenness).

[Base Case: k = 0] The empty circuit is vacuously correct.

[Inductive Step] For a graph with m > 0 edges, we start at x. Then, call just_walk(x,x) which returns a cycle C starting and ending at x (by Exercise 13). Remove C's edges, yielding G' with fewer edges and all vertices even.

The edges of C are stored in later. The algorithm then processes each edge of C in order. At each vertex v along C, it calls just_walk(v, v) on G', which by induction, finds an Eulerian circuit in the remaining component containing v. These sub-circuits are prepended to later before continuing along C.

Thus, the final sequence interleaves C with Eulerian circuits of the remaining components, forming an Eulerian circuit of the original graph. Each edge is used exactly once, and the walk starts and ends at x.