Euler Graph,
Fleury’s Algorithm,
Hamiltonian Graph

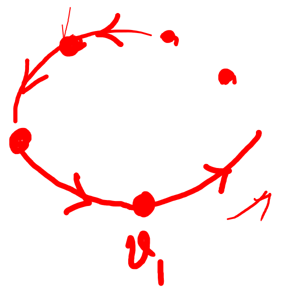
Euler Graph

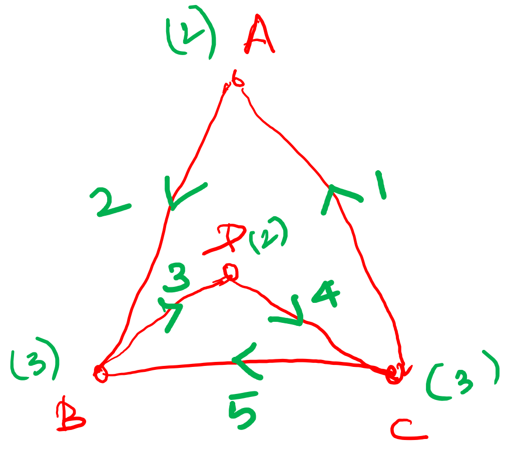
If some closed walk in a graph contains all the edges of the graph, then the walk is called an *Euler line* and the graph an *Euler graph*.

THEOREM

A given connected graph G is an Euler graph if and only if all vertices of G are of even degree.

- Proof: Suppose that G is an Euler graph.
- It therefore contains an Euler line (which is a closed walk).
- In tracing this walk we observe that every time the walk meets a vertex v it goes through two “new” edges incident on v—with one we “entered” v and with the other “exited.” This is true not only of all intermediate vertices of the walk but also of the terminal vertex, because we “exited” and “entered” the same vertex at the beginning and end of the walk, respectively.
- Thus if G is an Euler graph, the degree of every vertex is even.





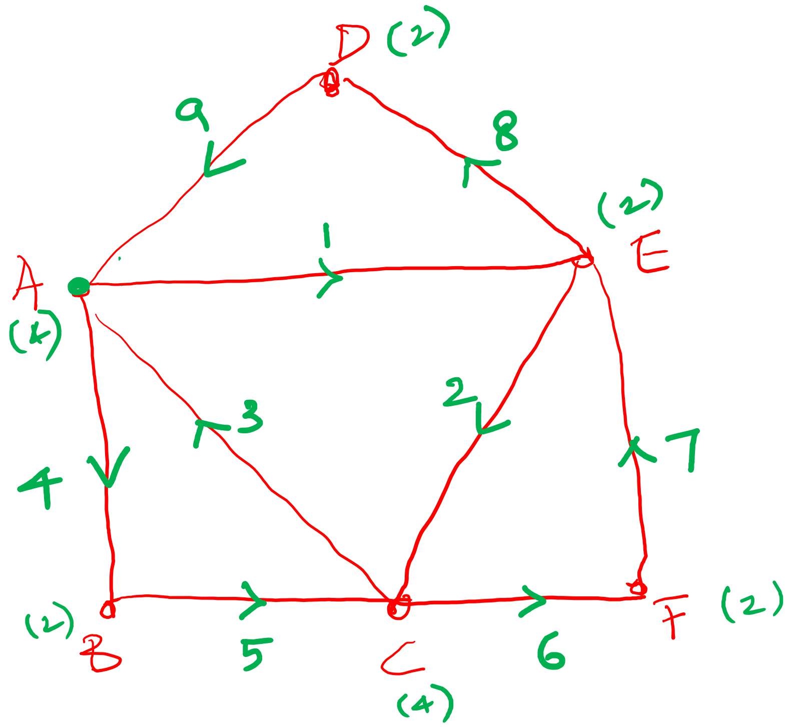
C 1 A 2 B 3 D 4 C 5 B

Open walk

Euler path

Open Euler line

Unicursal line



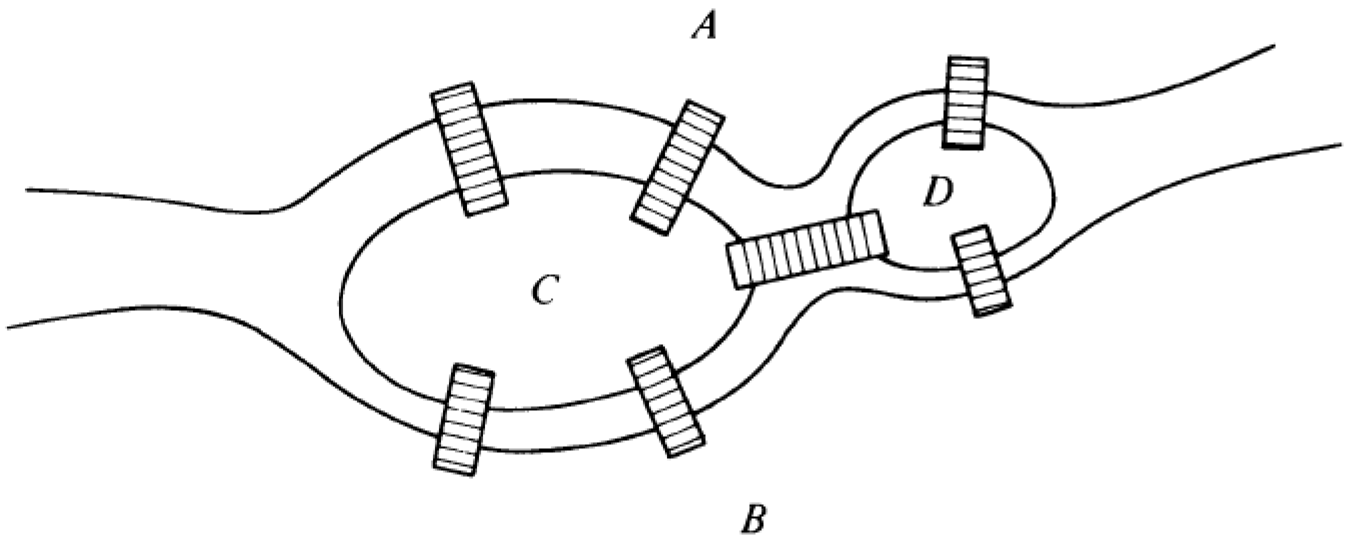
✓ A 1 E 2 C 3 A 4 B 5 C 6 F 7 E 8 D 9 A

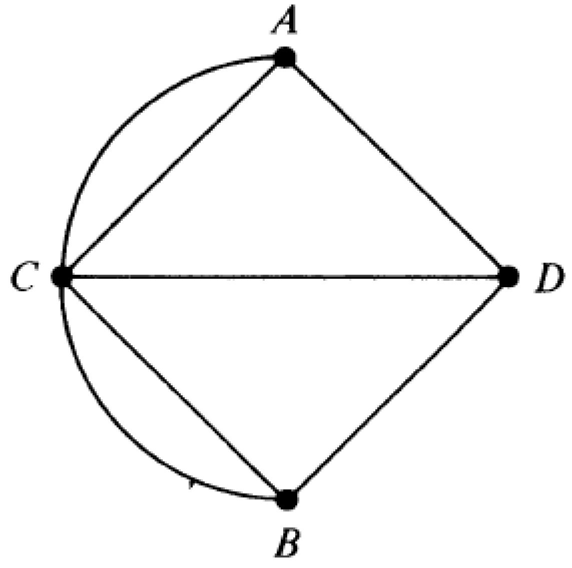
Closed walk ✓

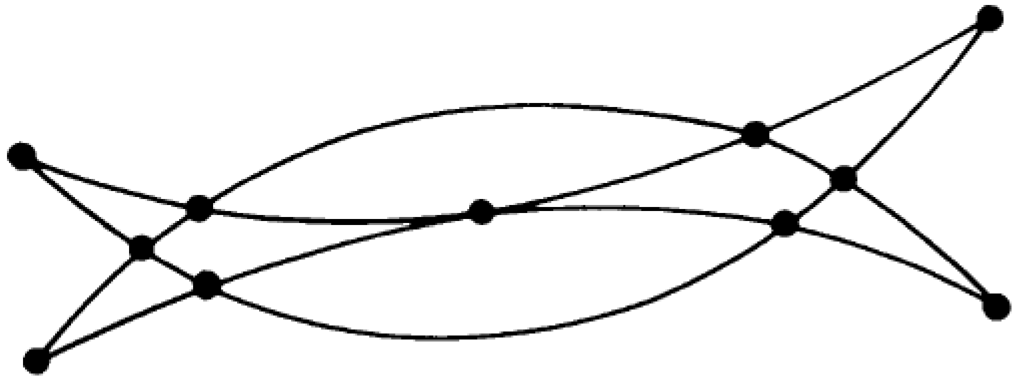
Euler circuit / Euler line

Euler Graph

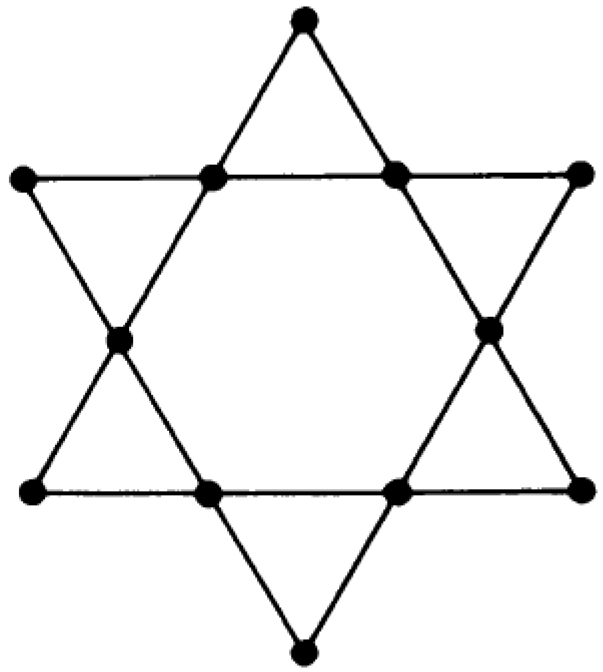
Königsberg Bridge Problem.







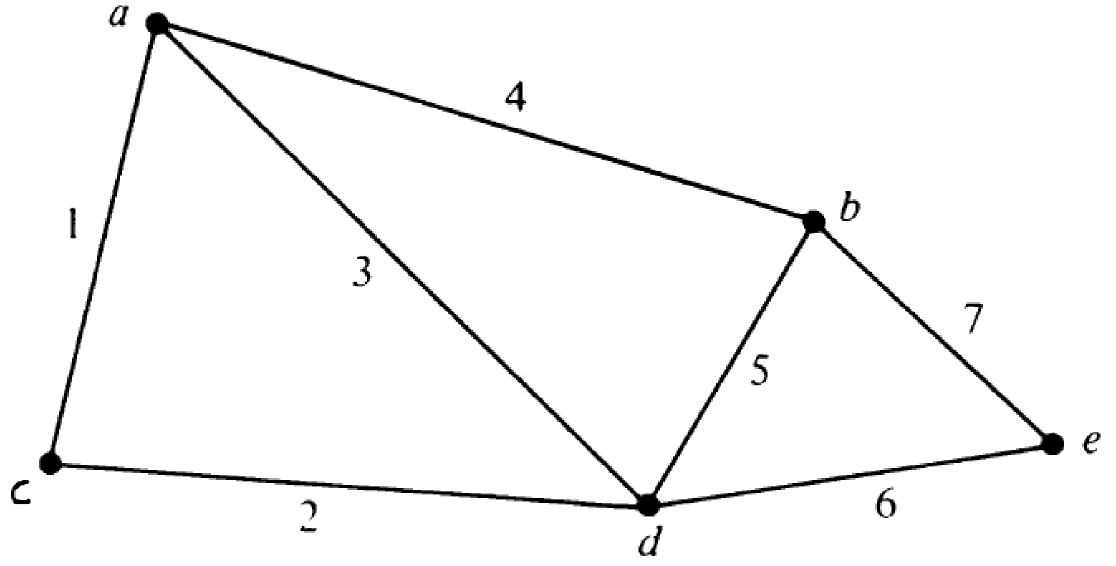
(a)



(b)

Two Euler graphs.

an open walk that includes (or traces or covers) all edges of a graph without retracing any edge a *unicursal line* or an *open Euler line*. A (connected) graph that has a unicursal line will be called a *unicursal graph*.



a 1 *c* 2 *d* 3 *a* 4 *b* 5 *d* 6 *e* 7 *b*

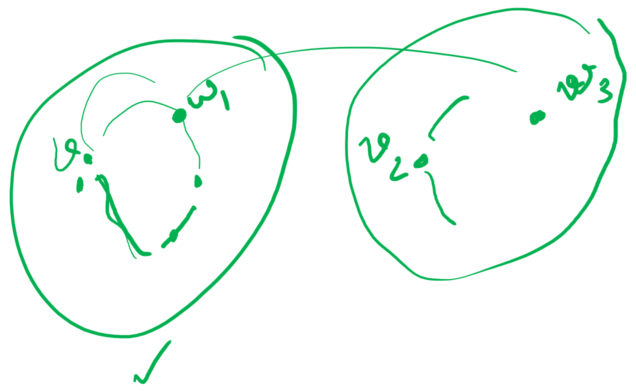
It is clear that by adding an edge between the initial and final vertices of a unicursal line we shall get an Euler line. Thus a connected graph is unicursal if and only if it has exactly two vertices of odd degree.

Theorem

In a connected graph G with exactly $2k$ odd vertices, there exist k edge-disjoint subgraphs such that they together contain all edges of G and that each is a unicursal graph.

$v_1, v_2, \dots \dots v_k$

$w_1, w_2, w_3 \dots \dots w_k$

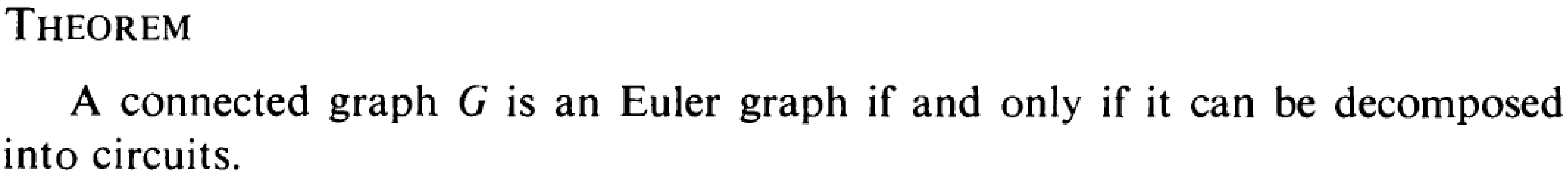


THEOREM

In a connected graph G with exactly $2k$ odd vertices, there exist k edge-disjoint subgraphs such that they together contain all edges of G and that each is a unicursal graph.

- Proof: Let the odd vertices of the given graph G be named v1, v2, . . . , vk; w1, w2, . . . , wk in any arbitrary order.
- Add k edges to G between the vertex pairs (v1, w1), (v2, w2), . . . , (vk, wk) to form a new graph G′.
- Since every vertex of G′ is of even degree, G′ consists of an Euler line p.
- Now if we remove from p the k edges we just added (no two of these edges are incident on the same vertex), p will be split into k walks, each of which is a unicursal line: The first removal will leave a single unicursal line; the second removal will split that into two unicursal lines; and each successive removal will split a unicursal line into two unicursal lines, until there are k of them.
- Thus the theorem.

Euler Graphs continued…

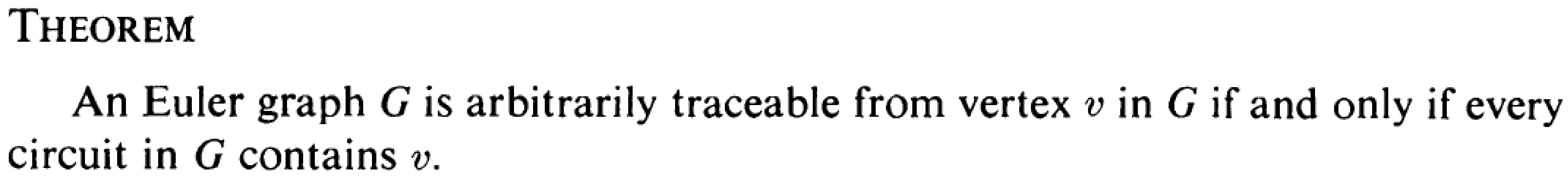


THEOREM

A connected graph G is an Euler graph if and only if it can be decomposed into circuits.

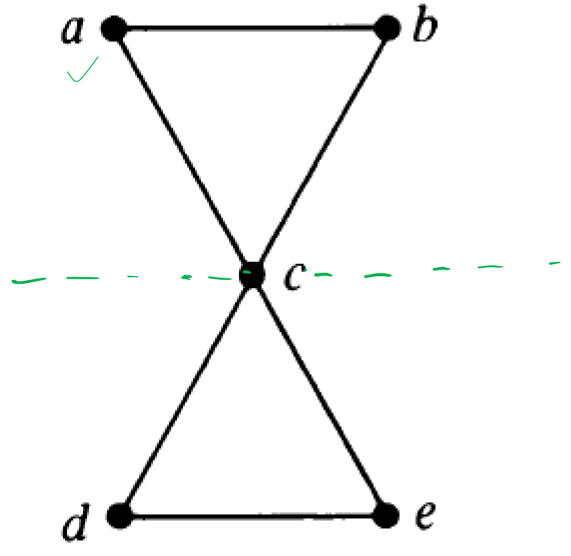
- Proof:
- Suppose graph G can be decomposed into circuits; that is, G is a union of edge-disjoint circuits. Since the degree of every vertex in a circuit is two, the degree of every vertex in G is even. Hence G is an Euler graph.
- Conversely, let G be an Euler graph. Consider a vertex v1. There are at least two edges incident at v1. Let one of these edges be between v1 and v2. Since vertex v2 is also of even degree, it must have at least another edge, say between v2 and v3. Proceeding in this fashion, we eventually arrive at a vertex that has previously been traversed, thus forming a circuit Γ. Let us remove Γ from G. All vertices in the remaining graph (not necessarily connected) must also be of even degree. From the remaining graph remove another circuit in exactly the same way as we removed Γ from G. Continue this process until no edges are left.
- Hence the theorem.

Euler Graphs continued…

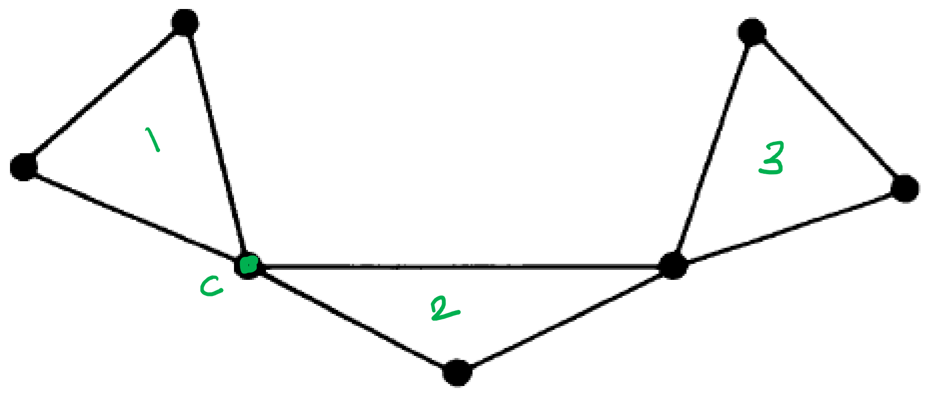


THEOREM

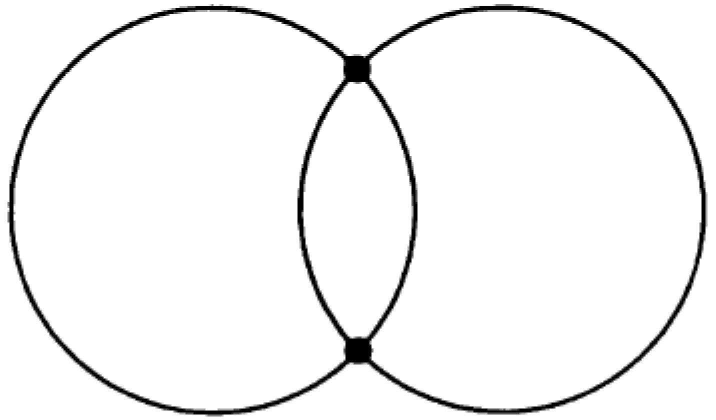
An Euler graph G is arbitrarily traceable from vertex v in G if and only if every circuit in G contains v.



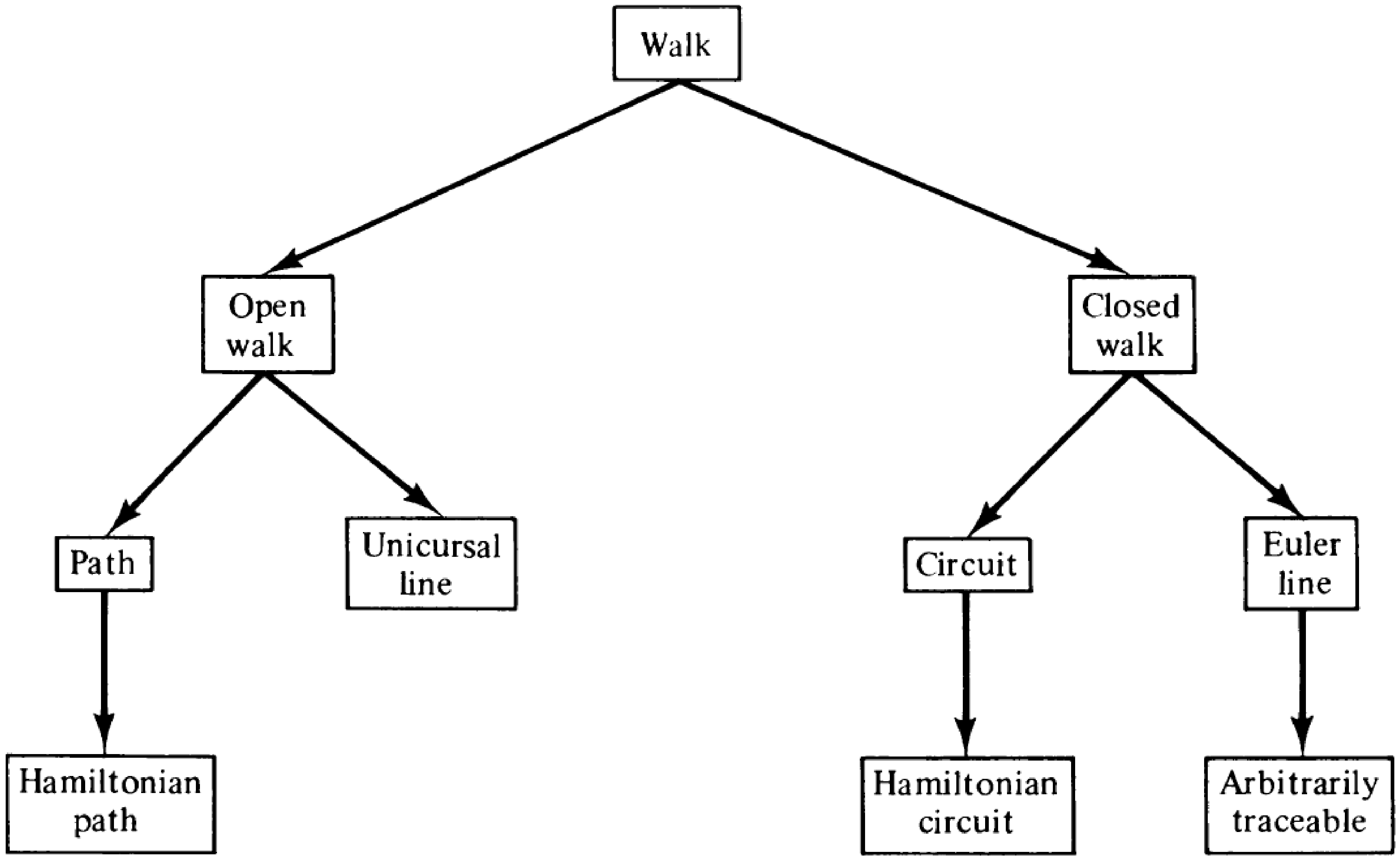
Arbitrarily traceable graph from *c*.



Euler graph; not arbitrarily traceable.



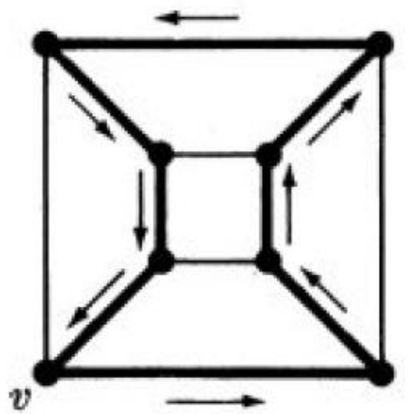
Arbitrarily traceable graph from all vertices.

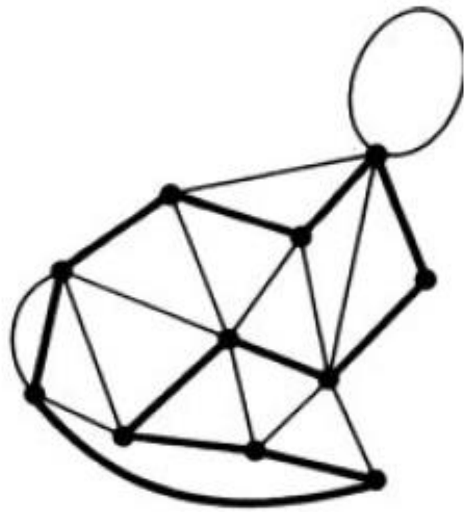


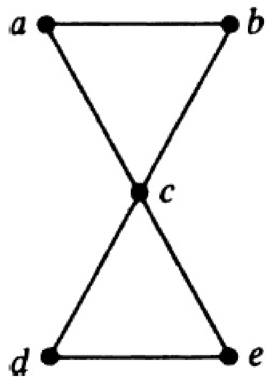
Different types of walks.

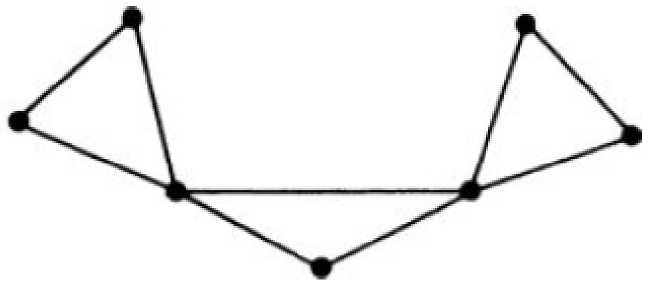
Hamiltonian Paths and Circuits

- A Hamiltonian circuit in a connected graph is defined as a closed walk that traverses every vertex of G exactly once, except of course the starting vertex, at which the walk also terminates.
- A circuit in a connected graph G is said to be Hamiltonian if it includes every vertex of G. Hence a Hamiltonian circuit in a graph of n vertices consists of exactly n edges.
- If we remove any one edge from a Hamiltonian circuit, we are left with a path, called a Hamiltonian path.
- A Hamiltonian path in a graph G traverses every vertex of G.
- Since a Hamiltonian path is a subgraph of a Hamiltonian circuit (which in turn is a subgraph of another graph), every graph that has a Hamiltonian circuit also has a Hamiltonian path
- The length of a Hamiltonian path in a connected graph of n vertices is n — 1.









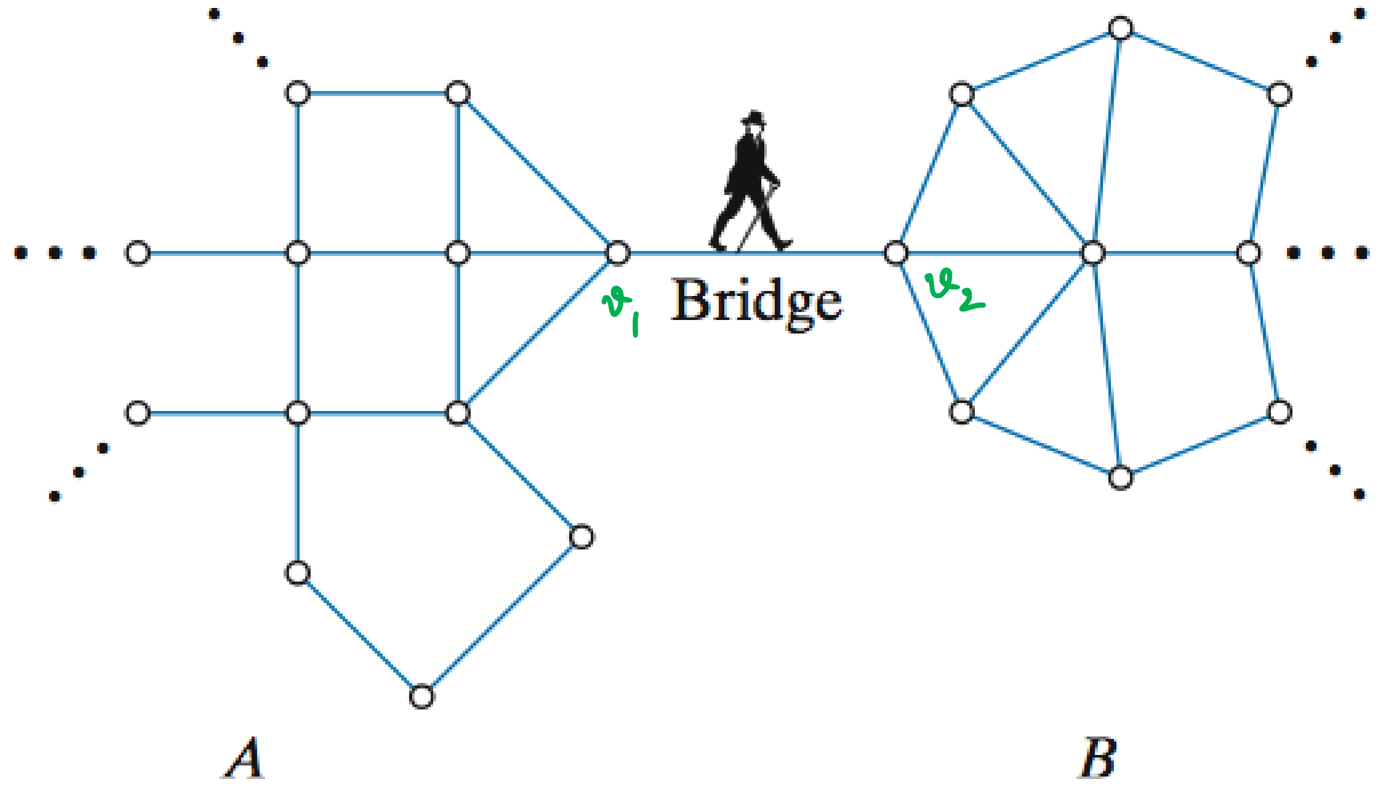
Second and third graphs don’t have Hamiltonian circuit

Fleury’s Algorithm

- We will now turn our attention to an algorithm that finds an *Euler circuit* or an *Euler path* in a connected graph. Technically speaking, these are two separate algorithms, but in essence they are identical, so they can be described as one.
- The idea behind Fleury’s algorithm can be paraphrased by that old piece of folk wisdom: *Don’t burn your bridges behind you*.

Fleury’s Algorithm

In graph theory the word bridge has a very specific meaning–it is the only edge connecting two separate sections (subgraph *A* and *B*) of a graph, as illustrated in Fig.



Fleury’s Algorithm

Fleury’s algorithm is based on a simple principle: To find an Euler circuit or an Euler path, *bridges are the last edges you want to cross*.

Our concerns lie only on how we are going to get around the *yet-to-be-traveled* part of the graph.

Thus, when we talk about bridges that we want to leave as a last resort, we are really referring to *bridges of the to-be-traveled part of the graph*.

FLEURY’S ALGORITHM FOR FINDING AN EULER CIRCUIT (PATH)

- **Preliminaries.** Make sure that the graph is connected and either (1) has no odd vertices (circuit) or (2) has just two odd vertices (path).
- **Start.** Choose a starting vertex. [In case (1) this can be any vertex; in case (2) it must be one of the two odd vertices.]

FLEURY'S ALGORITHM FOR FINDING AN EULER CIRCUIT (PATH)

- **Intermediate steps.** At each step, if you have a choice, don't choose a bridge of the yet-to-be-traveled part of the graph. However, if you have only one choice, take it.
- **End.** When you can't travel any more, the circuit (path) is complete. [In case (1) you will be back at the starting vertex; in case (2) you will end at the other odd vertex.]

Fleury’s Algorithm Bookkeeping

In implementing Fleury’s algorithm it is critical to separate the past (the part of the graph that has already been traveled) from the future (the part of the graph that still needs to be traveled).

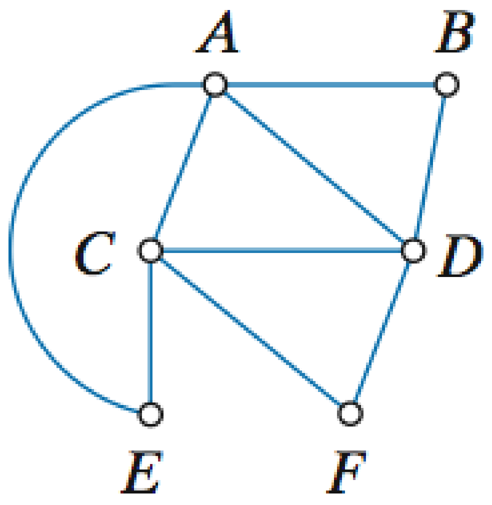
While there are many different ways to accomplish this (you are certainly encouraged to come up with one of your own), a fairly reliable way goes like this: Start with two copies of the graph. Copy 1 is to keep track of the “future”; copy 2 is to keep track of the “past.”

Fleury’s Algorithm Bookkeeping

Every time you travel along an edge, erase the edge from copy 1, but mark it (say in red) and label it with the appropriate number on copy 2. As you move forward, copy 1 gets smaller and copy 2 gets redder. At the end, copy 1 has disappeared; copy 2 shows the actual Euler circuit or path.

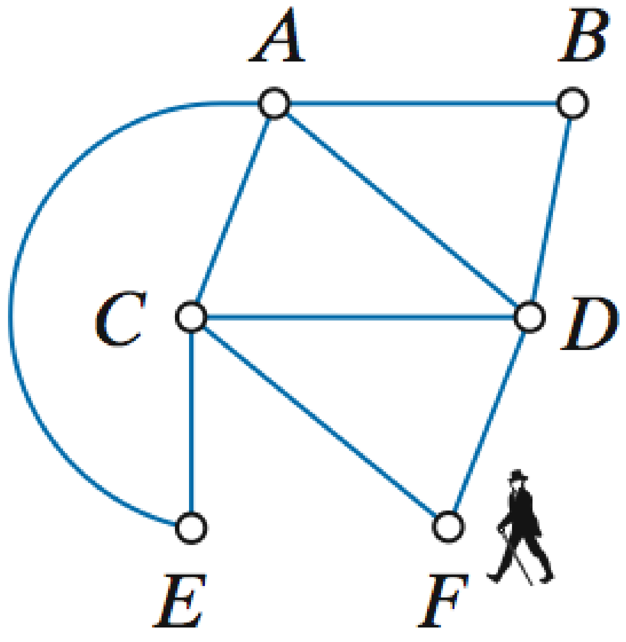
Example: Implementing Fleury's Algorithm

The graph in Fig is a very simple graph – it would be easier to find an Euler circuit just by trial-and-error than by using Fleury's algorithm. Nonetheless, we will do it using Fleury's algorithm. The real purpose of the example is to see the algorithm at work. Each step of the algorithm is explained in Figs.

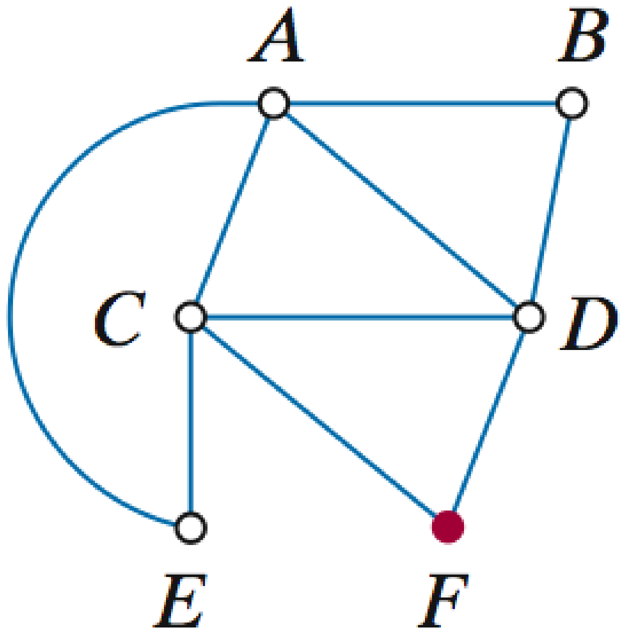


Example Implementing Fleury's Algorithm…

Start: We can pick any starting point we want. Let's say we start at *F*.



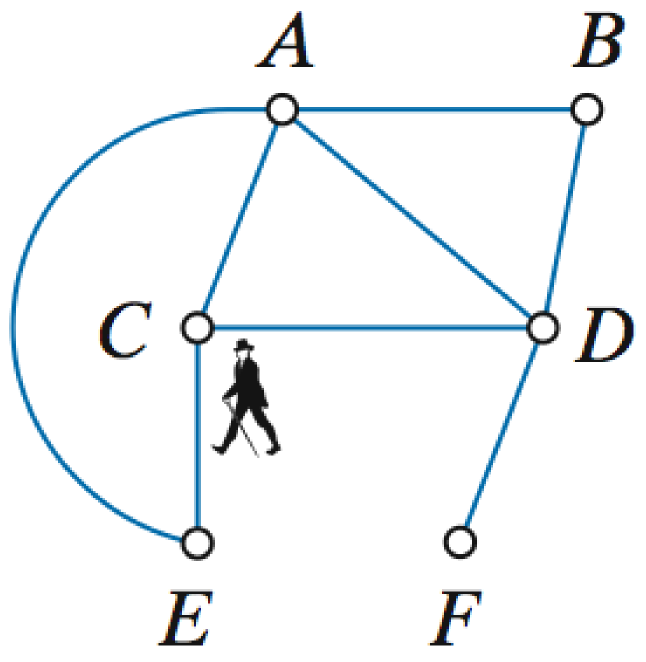
Copy 1



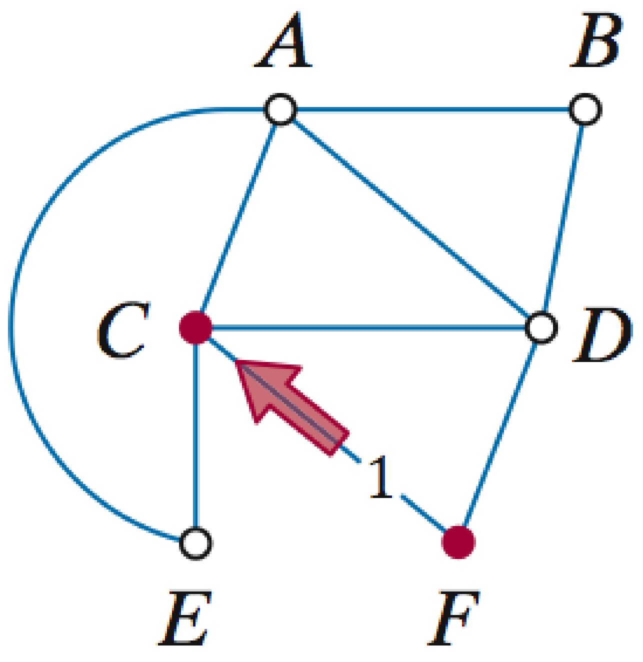
Copy 2

Example Implementing Fleury's Algorithm…

Step 1: Travel from *F* to *C*. (Could have also gone from *F* to *D*.)



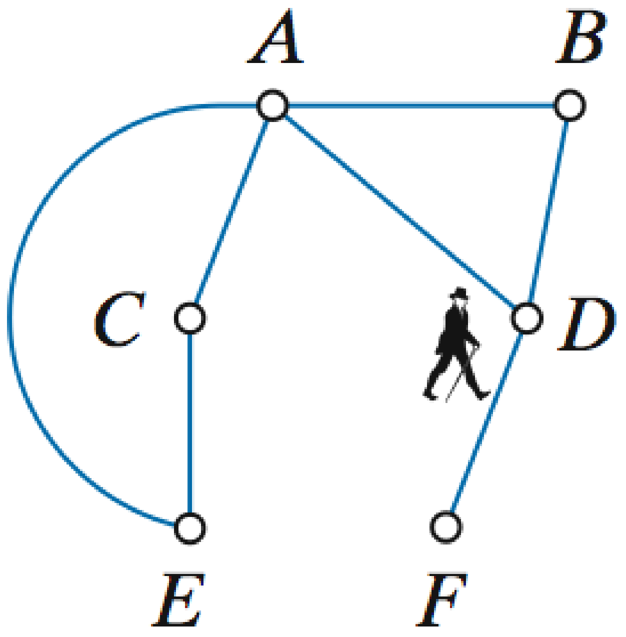
Copy 1



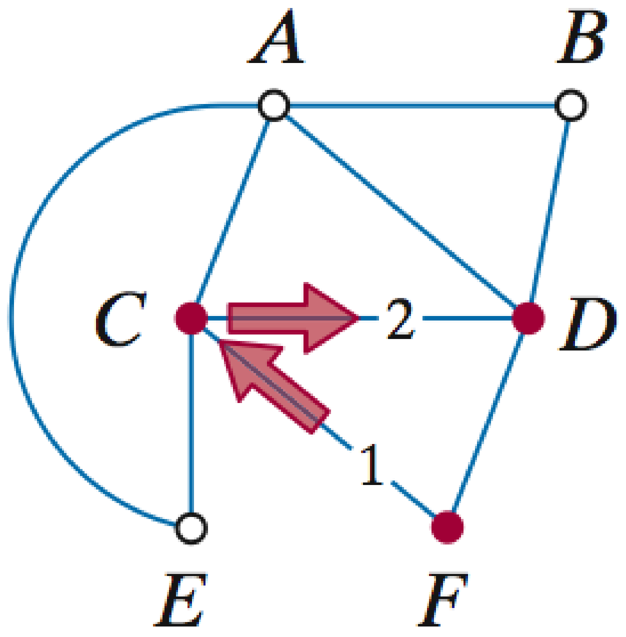
Copy 2

Example Implementing Fleury's Algorithm…

Step 2: Travel from *C* to *D*. (Could have also gone to *A* or to *E*.)



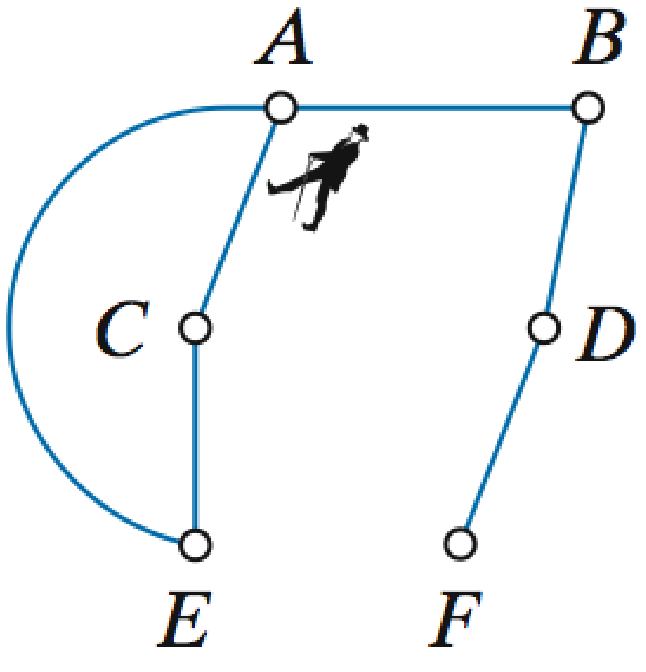
Copy 1



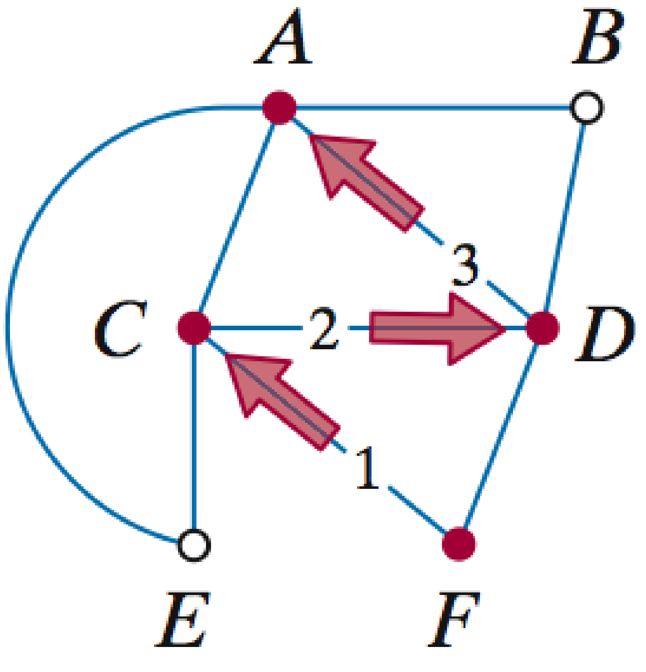
Copy 2

Example Implementing Fleury's Algorithm…

Step 3: Travel from *D* to *A*. (Could have also gone to *B* but not to *F* – *DF* is a bridge!)



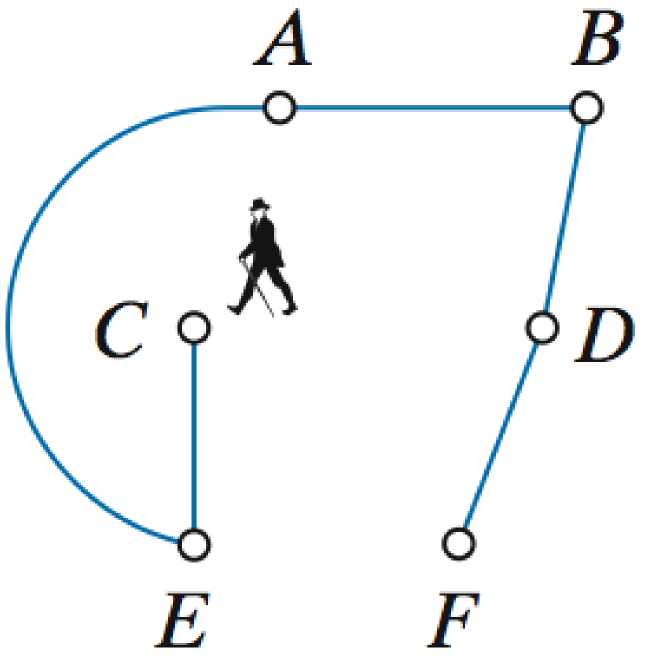
Copy 1



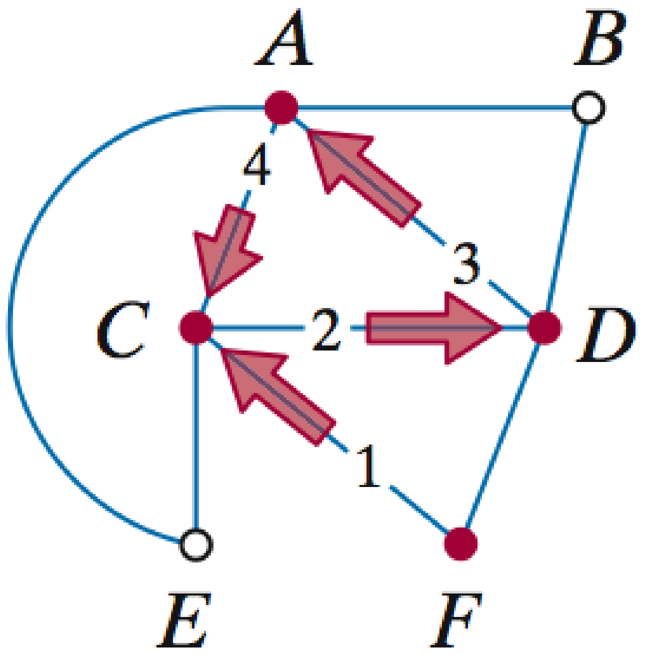
Copy 2

Example Implementing Fleury's Algorithm…

Step 4: Travel from *A* to *C*. (Could have also gone to *E* but not to *B* – *AB* is a bridge!)



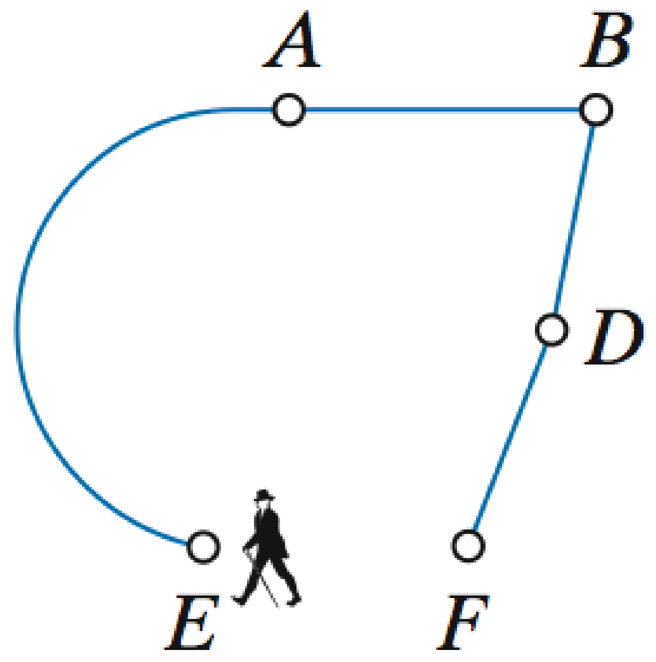
Copy 1



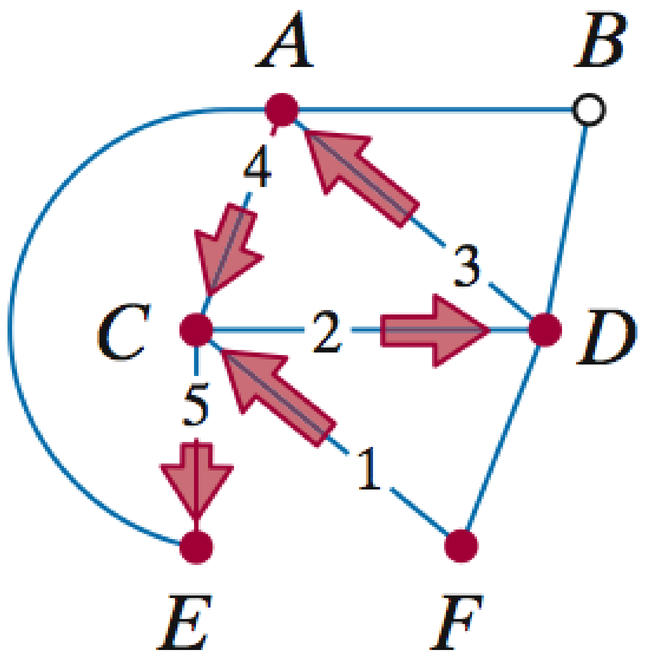
Copy 2

Example Implementing Fleury's Algorithm…

Step 5: Travel from *C* to *E*. (There is no choice!)



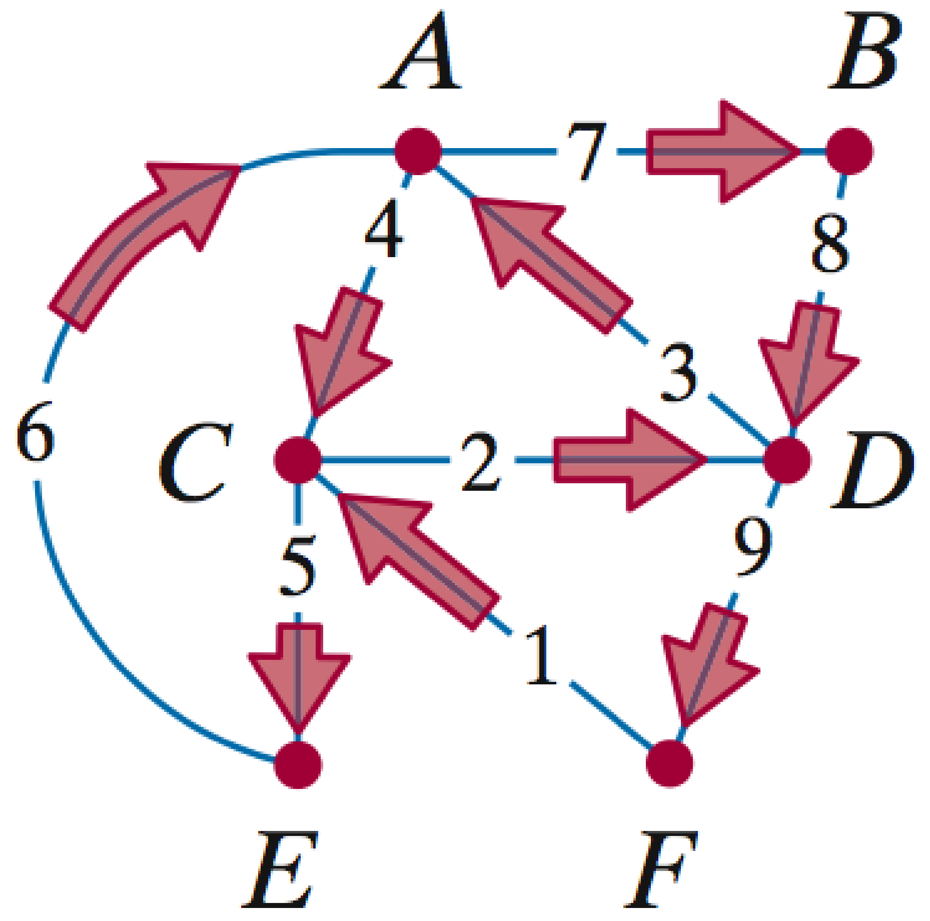
Copy 1



Copy 2

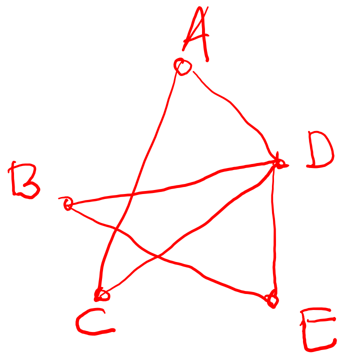
Example Implementing Fleury's Algorithm…

Steps 6, 7, 8, and 9: Only one way to go at each step.



F 1 C 2 D 3 A 4 C 5 E 6 A 7 B 8 D 9 F

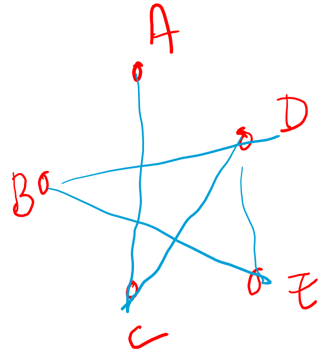
Example: Fleury's Algorithm for Euler Circuit



Starting ~~from~~ A

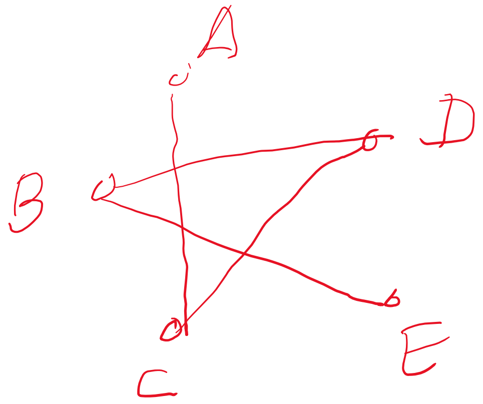
Choose A - D

circuit : AD



- D is current
- can't choose DC
- Can choose B or E
- Choosing DE

circuit: ADE

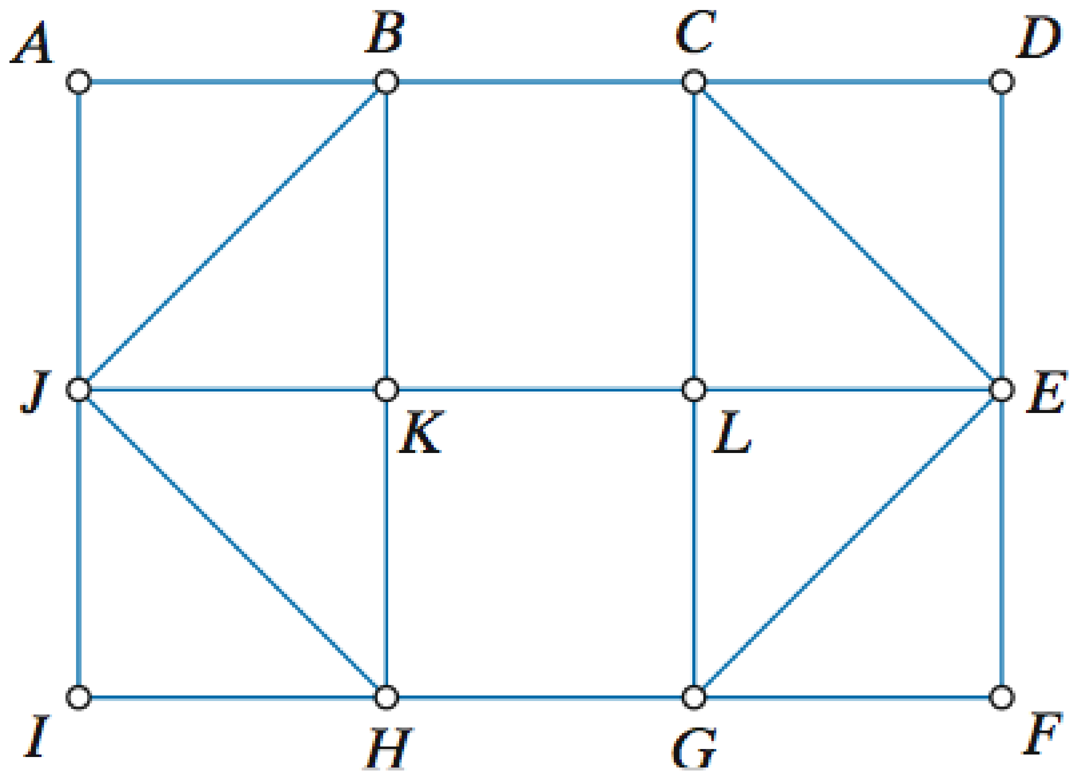


- E is current
- from here only one option

circuit : ADEBDCA

Example: Fleury’s Algorithm for Euler Paths

We will apply Fleury’s algorithm to the graph in Fig.



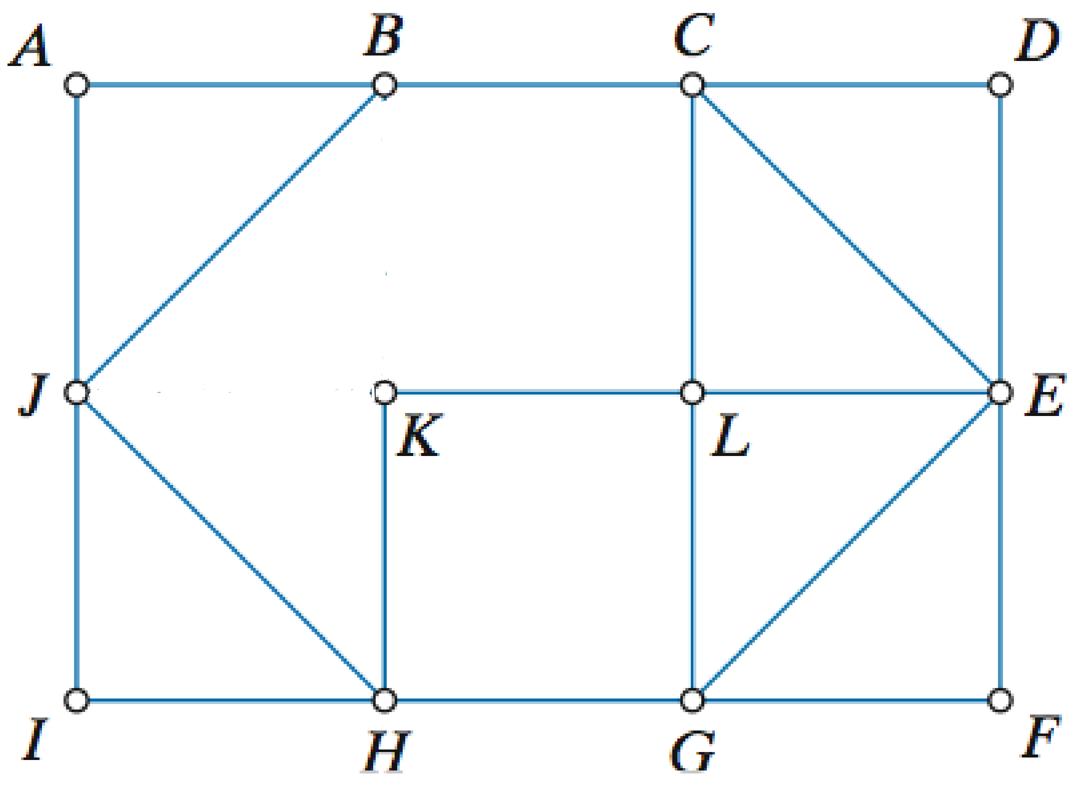
Example: Fleury’s Algorithm for Euler Paths…

- **Start.** This graph has two odd vertices, *E* and *J*. We can pick either one as the starting vertex. Let’s start at *J*.

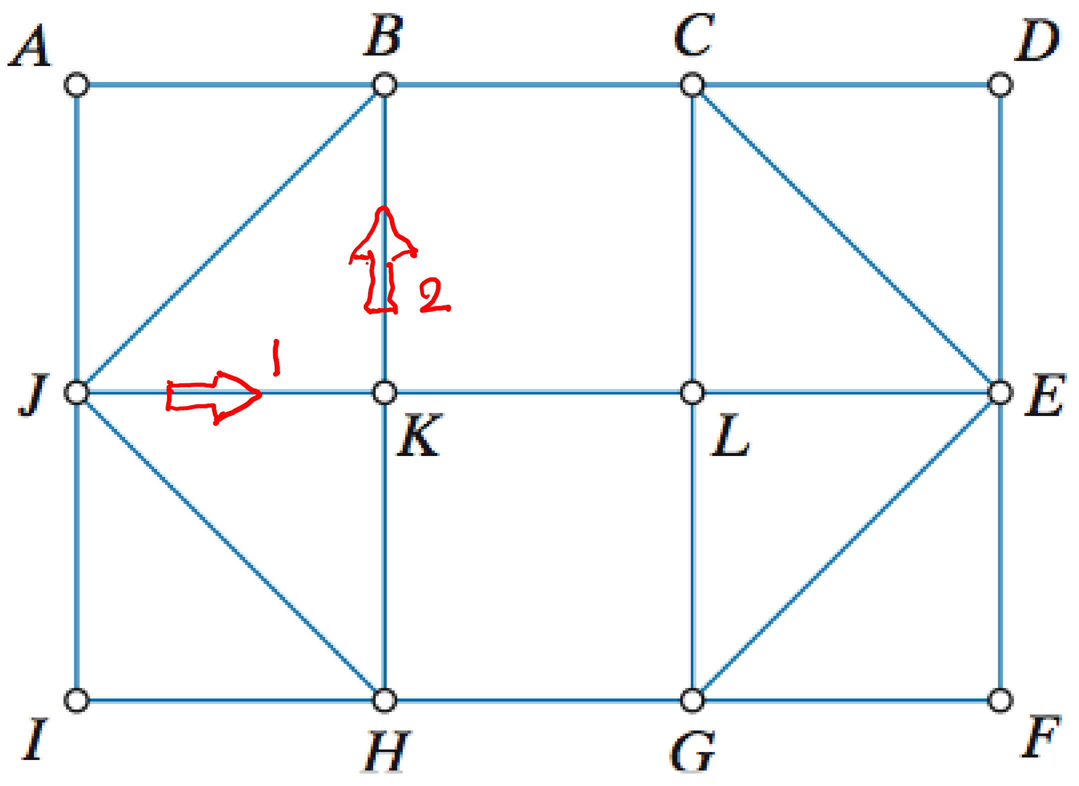
Example: Fleury’s Algorithm for Euler Paths…

- **Step 1.** From *J* we have five choices, all of which are OK. We’ll randomly pick *K*. (Erase *JK* on copy 1, and mark and label *JK* with a 1 on copy 2.)

- **Step 2.** From *K* we have three choices (*B*, *L*, or *H*). Any of these choices is OK. Say we choose *B*. (Now erase *KB* from copy 1 and mark and label *KB* with a 2 on copy 2.)



Copy 1

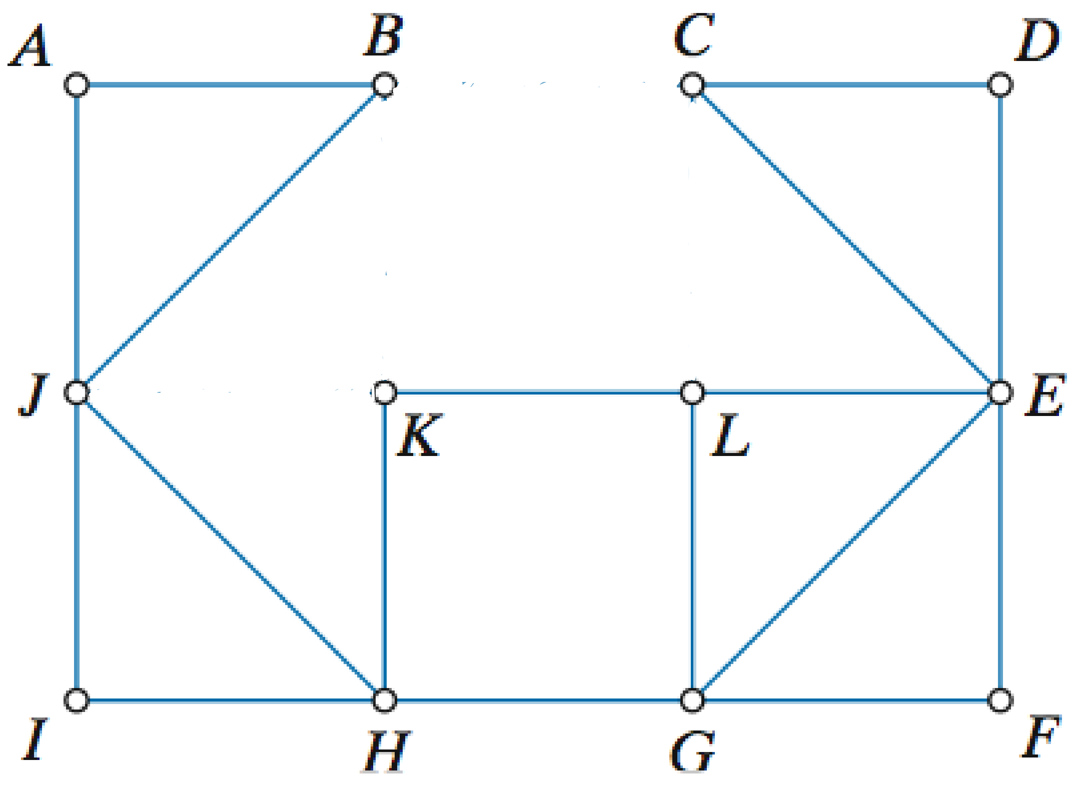


Copy 2

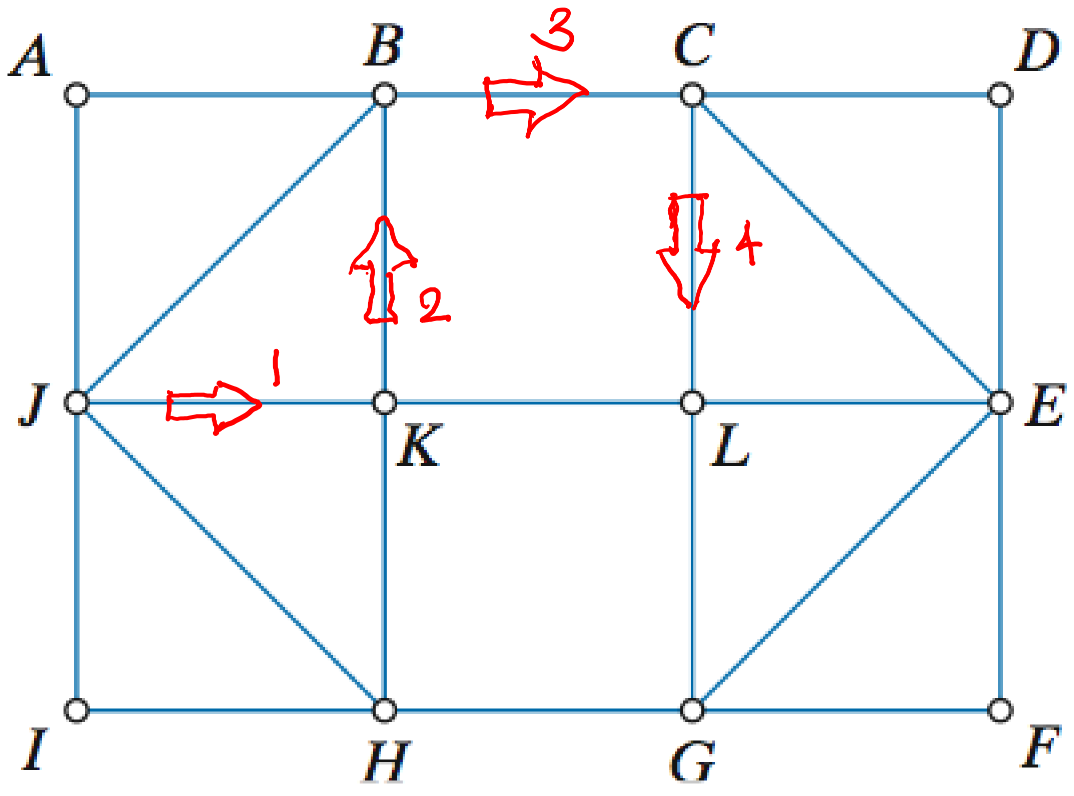
Example: Fleury's Algorithm for Euler Paths…

- **Step 3.** From *B* we have three choices (*A*, *C*, or *J*). Any of these choices is OK. Say we choose *C*. (Now erase *BC* from copy 1 and mark and label *BC* with a 3 on copy 2.)

- **Step 4.** From *C* we have three choices (*D*, *E*, or *L*). Any of these choices is OK. Say we choose *L*. (EML–that's shorthand for erase, mark, and label.)



Copy 1



Copy 2

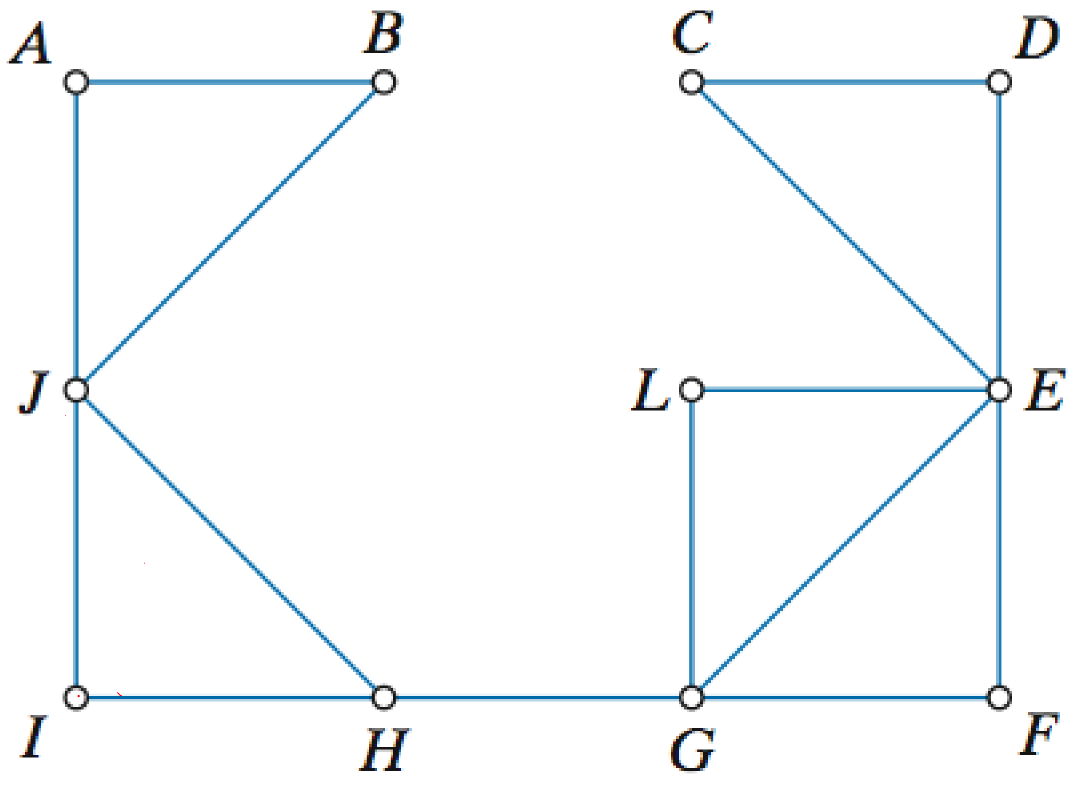
Example: Fleury’s Algorithm for Euler Paths…

- **Step 5.** From L we have three choices (*E*, *G*, or *K*). Any of these choices is OK. Say we choose *K*. (EML.)

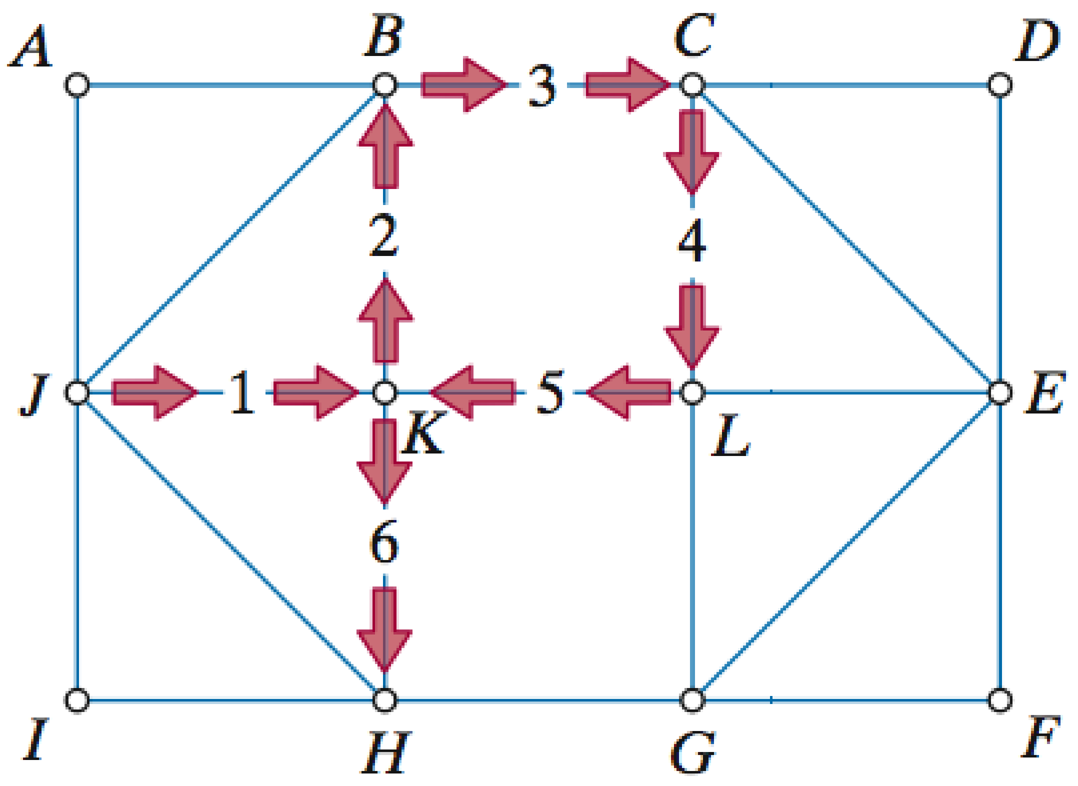
- **Step 6.** From *K* we have only one choice– to H. We choose *H*. (EML.)

Example: Fleury’s Algorithm for Euler Paths…

- **Step 7.** From H we have three choices (*G*, *I*, or *J*). We should not choose *G*, as *HG* is a bridge of the yet-to-be-traveled part of the graph



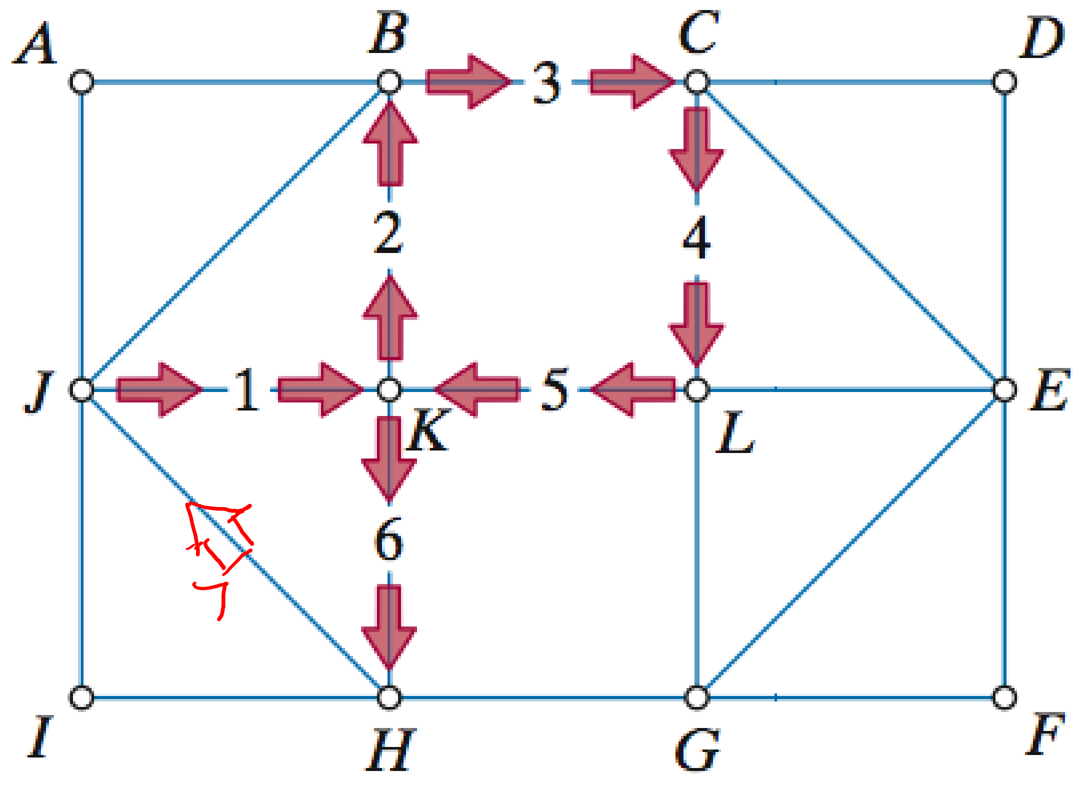
Copy 1 at Step 7



Copy 2 at Step 7

Example: Fleury's Algorithm for Euler Paths…

- **Step 7.** Either of the other two choices is OK. Say we choose J. (EML.)



Copy 2

Example: Fleury's Algorithm for Euler Paths…

- **Step 8.** From *J* we have three choices (*A*, *B*, or *I*), but we should not choose *I*, as *JI* has just become a bridge. Either of the other two choices is OK. Say we choose *B*. (EML)

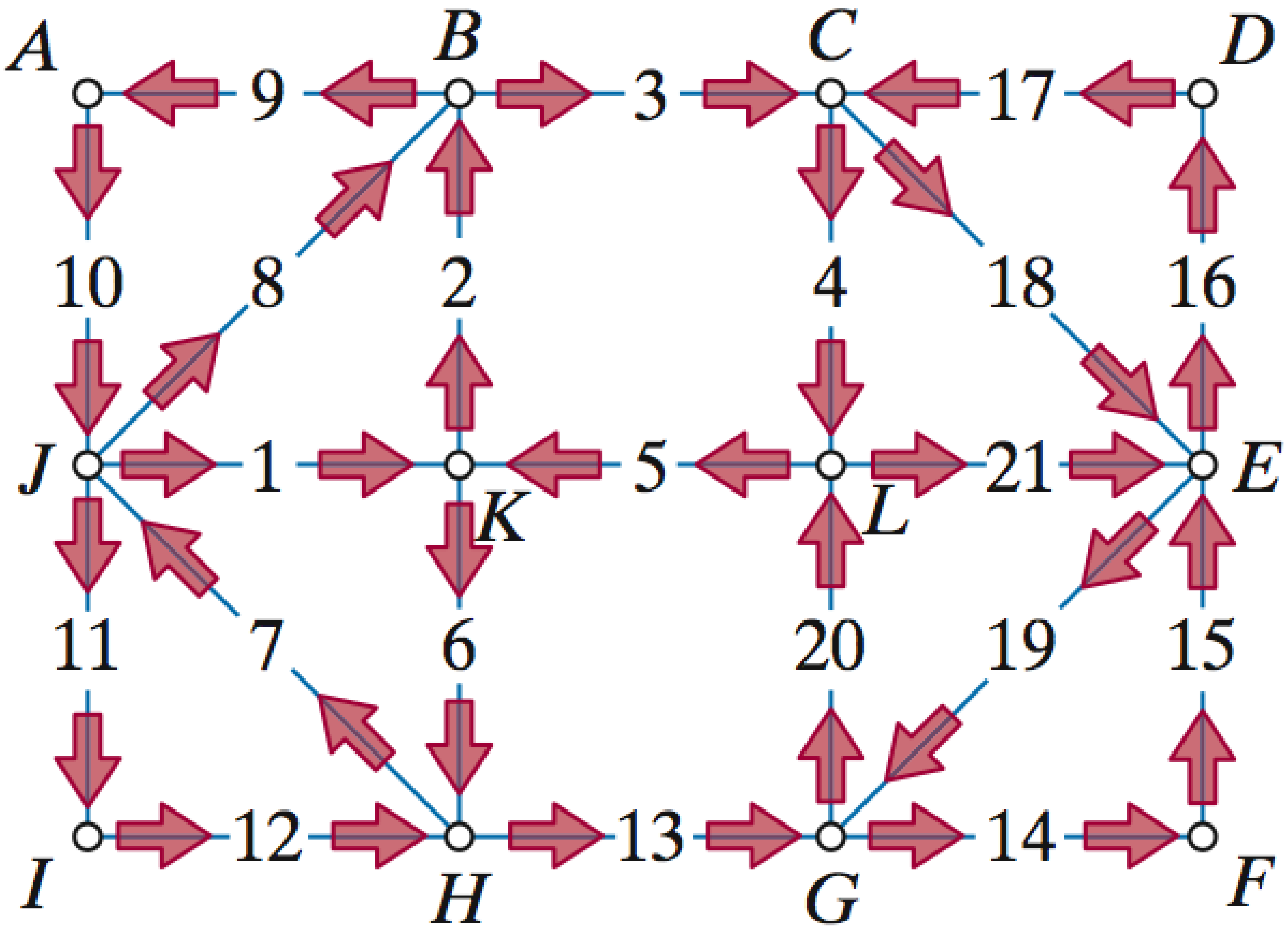
- **Step 9 through 13.** Each time we have only one choice. From *B* we have to go to *A*, then to *J*, *I*, *H*, and *G*.

Example: Fleury’s Algorithm for Euler Paths…

- **Step 14 through 21.** Not to belabor the point, let’s just cut to the chase. The rest of the path is given by *G*, *F*, *E*, *D*, *C*, *E*, *G*, *L*, *E*. There are many possible endings, and you should find a different one by yourself.

The completed Euler path (one of hundreds of possible ones) is shown in Fig.

Example: Fleury’s Algorithm for Euler Paths…



Hamiltonian Paths and Circuits Continued…

- What general class of graphs is guaranteed to have a Hamiltonian circuit?
- Complete graphs of three or more vertices constitute one such class.
- It is easy to construct a Hamiltonian circuit in a complete graph of n vertices.
- Let the vertices be numbered v1, v2 ,. . . , vn. Since an edge exists between any two vertices, we can start from v1 and traverse to v2, and v3, and so on to vn, and finally from vn to v1. This is a Hamiltonian circuit.

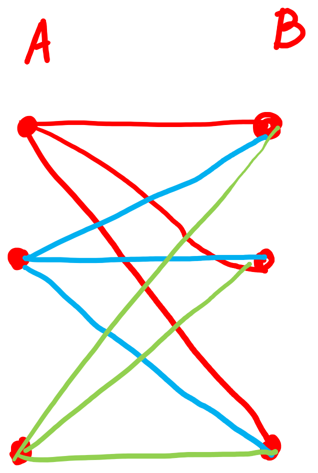
Number of Hamiltonian Circuits in a Graph

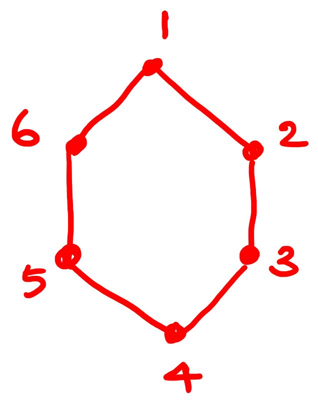
- A given graph may contain more than one Hamiltonian circuit.
- **THEOREMS**
    - A sufficient (but by no means necessary) condition for a simple graph G to have a Hamiltonian circuit is that the degree of every vertex in G be at least n/2, where n is the number of vertices in G.
    - In a complete graph with n vertices there are $(n - 1)/2$ edge-disjoint Hamiltonian circuits, if n is an odd number ≥ 3.
    - The total number of different (not edge disjoint) Hamiltonian circuits in a complete graph of n vertices is $(n - 1)!/2$.
    - A bipartite graph is Hamiltonian only if $n = m \geq 2$, where n and m are the cardinality of vertex partitions

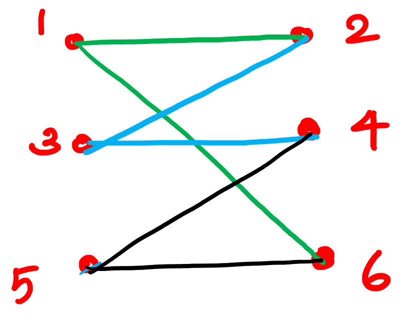
- A bipartite graph is Hamiltonian only if n = m ≥ 2, where n and m are the cardinality of vertex partitions
- **Proof:**
- Let $K_{m,n}$ have bipartition X, Y where $|x| = n$ and $|y| = m$.
- Let C be a Hamiltonian cycle of $K_{m,n}$
- Each cycle in $K_{m,n}$ has equal length and thus the cycle visits X and Y equally many times.
- Then C visits X and Y equally many times and is incident with all the vertices. Therefore, $|x| = |y|$

Bipartite graph

- Graph whose vertices can be divided into two parts A and B such that every edge connects a vertex in A to a vertex in B.





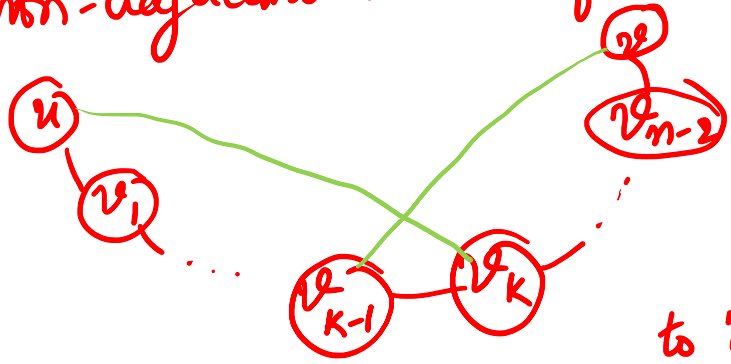


- The total number of different (not edge disjoint) Hamiltonian circuits in a complete graph of n vertices is $(n - 1)!/2$.
- Proof
- This follows from the fact that starting from any vertex we have $n - 1$ edges to choose from the first vertex, $n - 2$ from the second, $n - 3$ from the third, and so on.
- These being independent choices, we get $(n - 1)!$ possible number of choices.
- This number is divided by 2, because each Hamiltonian circuit has been counted twice

If G is a simple graph with $n (\geq 3)$ vertices and if $d(v) \geq n/2$ for each vertex v, then G is Hamiltonian.

Proof by Contradiction

Let G be such a counter example to the theorem so that no graph on n vertices with more edges than G is also a counter example. Let u, v be 2 non-adjacent vertices of G. Then, there is a Hamiltonian path joining u & v in G.



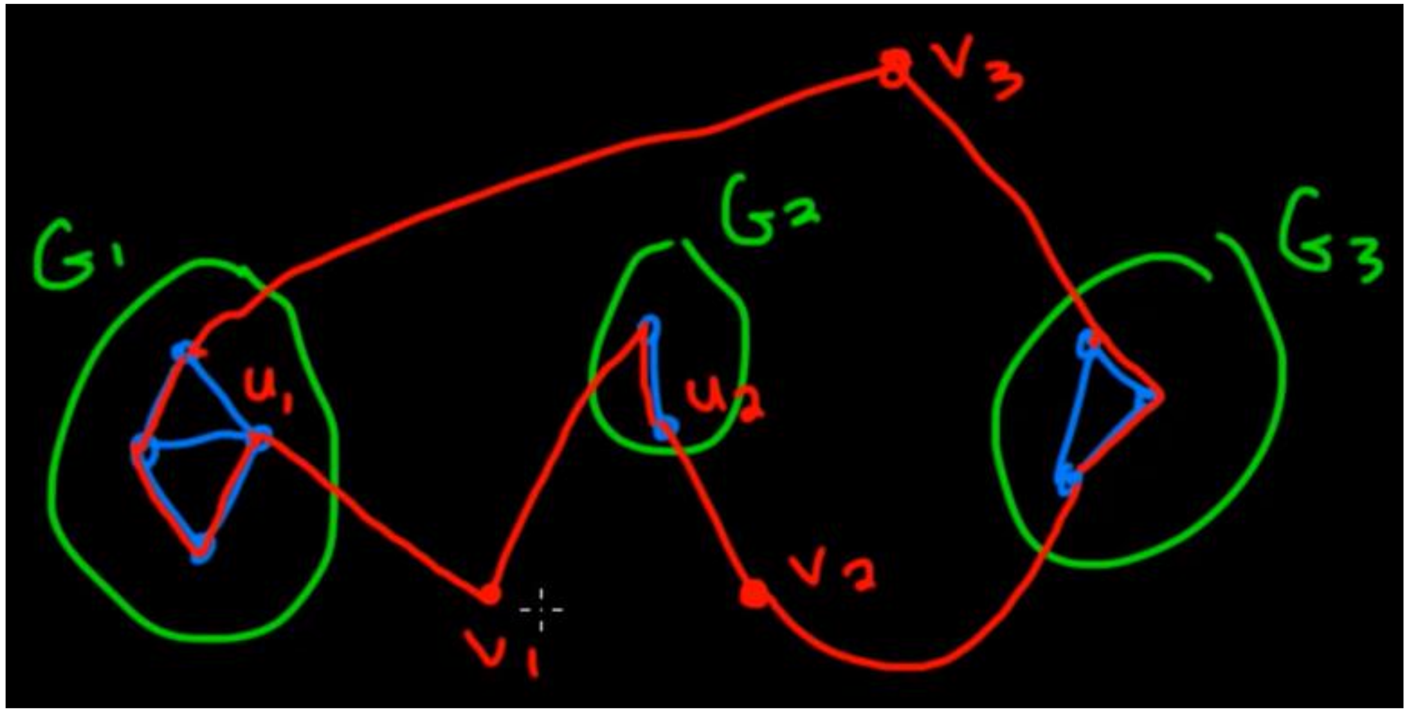
Let $d(u) = K \geq n/2$

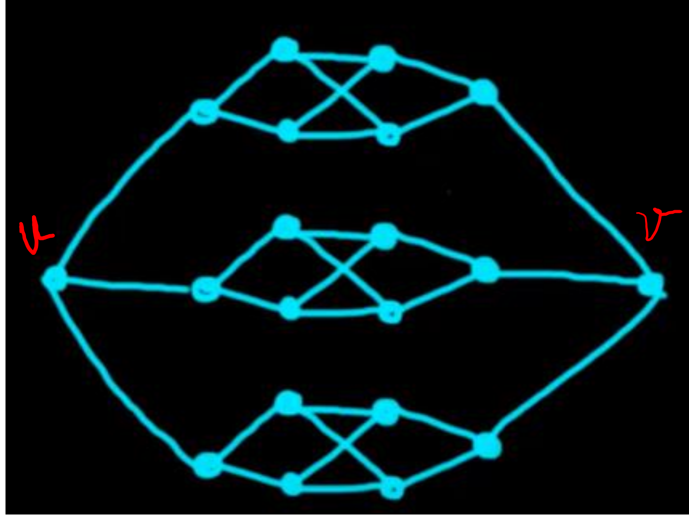
Now, we prove that if u is adjacent to v_K, then v_{K-1} can't be adjacent to v. If possible, let v be adjacent to v_{K-1}. Then, we have the Hamiltonian cycle,

$v v_{n-2} \ldots v_K u v_1 \ldots v_{K-1} v$. Since, we assumed that G is not Hamiltonian, this is not possible. So, v is not adjacent to v_{K-1}. Since u is adjacent to K vertices, v can't be adjacent to at least K of the $n-1$ vertices. Thus $d(v) \leq n-1-K \leq n-1-n/2 = n/2 - 1$; Contradiction. Hence, G must be Hamiltonian

- Let G be a simple graph G on n (≥ 3) vertices. Suppose u, v ∈ V(G) and (u, v) ∉ E(G) and d(u) + d(v) $\geq$ n then G is Hamiltonian if G + (u, v) is Hamiltonian.
- **Proof:** G is Hamiltonian then obviously G+(u, v) is Hamiltonian.
- Given G+(u,v) is ian, we need to prove that G is Hamiltonian.
- Suppose G is not Hamiltonian. Then, there is a Hamiltonian path or spanning path in G joining u and v. If u is adjacent to v_k, then v can't be adjacent to v_{k-1}. Let d(u) = k. So, v is not adjacent to at least k of the n-1 vertices.
- Thus d(v) $\leq$ n – 1 – k
- d(u) + d(v) $\leq$ n – 1 – k + k $\leq$ n – 1 which is a contradiction.
- So, G is Hamiltonian.

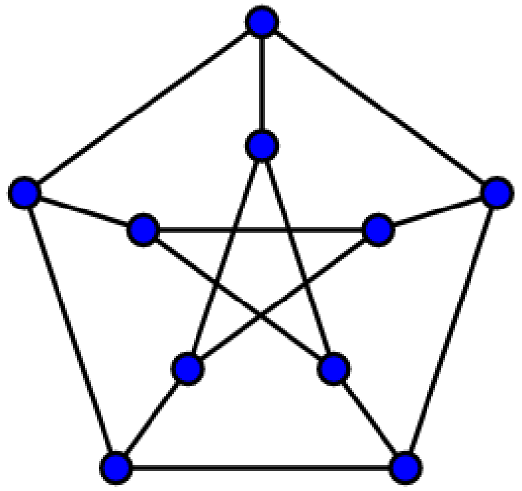
- If G is Hamiltonian, then for every non empty subset S ⊆ V(G), number of connected components c(G-S) $\leq |S|$
- **Proof:**
- G is Hamiltonian, let S ⊆ V(G), $S \neq \oint$ and let w∈ S and $C: w \longrightarrow w$ be a Hamiltonian cycle of G.
- Assume c(G-S) = k and let G_1, G_2, … G_k be the connected components of G-S.
- If k=1, $|S| \geq 1$
- So for k>1, consider an orientation of C.
- Let u_i be the last vertex of C that belongs to G_i. Let v_i be the next vertex of C after u_i. $v_i \notin V(Gi)$ by the definition of u_i and $v_i \notin V(Gj)$ since the components are mutually disjoint. So, $v_i \in S$.
- Also, $v_i \neq v_j$ for $i \neq j$, because C is a Hamiltonian cycle and $u_i v_i \in E(G)$ for all i.
- For each i = 1, 2, … , k there is a $v_i \in S$ and $|S| \geq k$
- K = c(G-S) $\leq |S|$


G_1
G_2
G_3
u_1
u_2
v_1
v_2
v_3



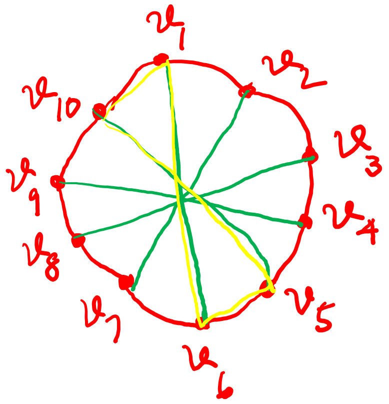
- S = {u, v}
- c(G-S) = 3 but |S| = 2.
- Hence, the graph is not Hamiltonian

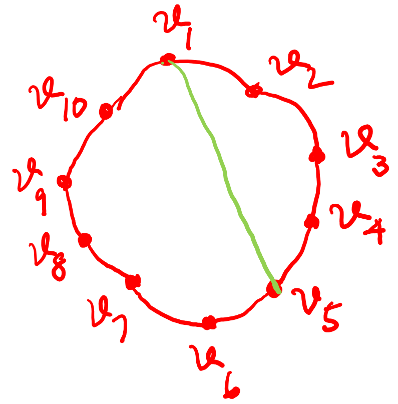
Petersen graph



- Not bipartite as cycles of odd length is present.
- Consists of Hamiltonian path
- It has no cycles on ≤ 4 vertices.
- It is not Hamiltonian.

- Proof by contradiction: Suppose C is a Hamiltonian cycle of the Petersen graph. $|V| = 10$, $|E| = 15$ and $d(v_i) = 3$ for all *i*.
- C is a 10-cycle together with 5 chords (edges that connect non-adjacent vertices in C).
- If each chord joins vertices opposite in C, then there exists a 4-cycle. It is contradiction to Petersen graph.
- There is a chord that connects vertices at distance <5 in C. These vertices must be at distance 4 in C. (Without loss of generality $v_1v_5 \in E(G)$).
- Now v_6 cannot be incident with any chord edge without making a cycle of length≤ 4. It is a contradiction to Petersen graph.
- Hence it is not Hamiltonian.





Traveling-salesman Problem

- A salesman is required to visit a number of cities during a trip. Given the distances between the cities, in what order should he travel so as to visit every city precisely once and return home, with the minimum mileage traveled?
- Representing the cities by vertices and the roads between them by edges, we get a graph. In this graph, with every edge e_i there is associated a real number (the distance in miles, say), $w(e_i)$.
- In our problem, if each of the cities has a road to every other city, we have a complete weighted graph. This graph has numerous Hamiltonian circuits, and we are to pick the one that has the smallest sum of distances (or weights).
- Theoretically, the problem of the traveling salesman can always be solved by enumerating all $(n - l)!/2$ Hamiltonian circuits, calculating the distance traveled in each, and then picking the shortest one. However, for a large value of n, the labor involved is too great even for a digital computer.
- You can be greedy in selecting edges.