



GRAPH THEORY

1

Dr Honey Sharma

GGI, Ludhiana

Reference Book: Kenneth H. Rosen, Discrete Mathematics and its Applications, 7th ed, McGraw Hill

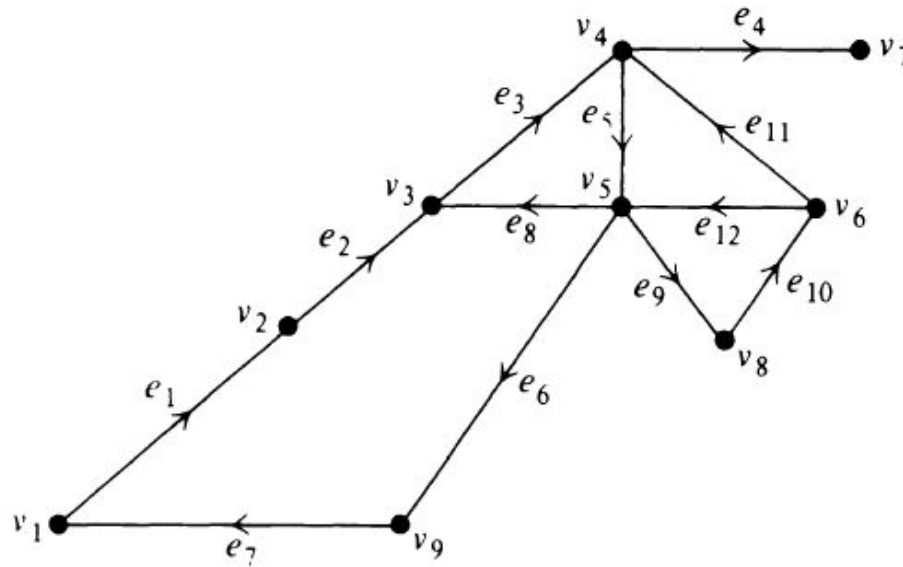
GRAPH CONNECTIVITY

- Many problems can be modeled with paths formed by traveling along the edges of graphs. For instance,
 - The problem of determining whether a message can be sent between two computers using intermediate links
 - Problems of efficiently planning routes for mail delivery, garbage pickup etc.

- A **path** is a sequence of edges that begins at a vertex of a graph and travels from vertex to vertex along edges of the graph.
- Let G be an undirected graph. A path of length n from vertex u to v in G is a sequence of n edges e_1, \dots, e_n of G for which there exists a sequence $v_0 = u, v_1, \dots, v_{n-1}, v_n = v$ of vertices such that e_i has, for $i = 1, \dots, n$, the endpoints v_{i-1} and v_i .
- The path is a circuit if it begins and ends at the same vertex, that is, if $u = v$, and has length greater than zero.

- The path or circuit is said to pass through the vertices v_1, v_2, \dots, v_{n-1} or traverse the edges e_1, e_2, \dots, e_n .
- A path or circuit is simple if it does not contain the same edge more than once.
- A path or circuit is elementary if it does not meet the same vertex more than once.
- In some books at place of path, word walk is used. The term **walk** is defined to be an alternating sequence of vertices and edges of a graph, $v_0, e_1, v_1, e_2, \dots, e_n, v_n$ where v_{i-1} and v_i are the endpoints of e_i for $i = 1, 2, \dots, n$.

- G be a directed graph. A *path* of length n from u to v in is a sequence of edges e_1, \dots, e_n of G such that e_i associated with (v_{i-1}, v_i) , where $v_0 = u$ and $v_n = v$.
- A path of length greater than zero that begins and ends at the same vertex is called a *circuit* or *cycle*.
- A path or circuit is called *simple* if it does not contain the same edge more than once.

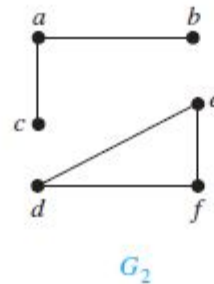
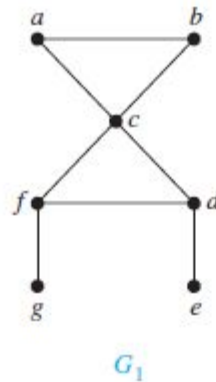


○ In above directed graph

- $(e_1, e_2, e_3, e_5, e_9, e_{10}, e_{12}, e_6, e_7)$ is a simple circuit, but not an elementary.
- $(e_1, e_2, e_3, e_5, e_9, e_6, e_7)$ is an elementary circuit.
- Some time a path or circuit is represented by sequence of vertices. For example, (e_1, e_2, e_3, e_4) can also be represented as $(v_1, v_2, v_3, v_4, v_7)$

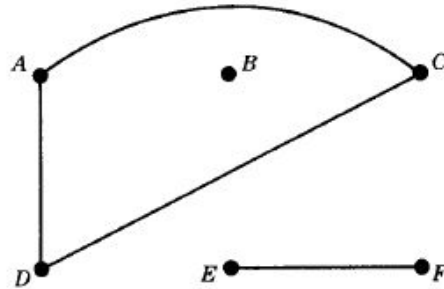
CONNECTEDNESS IN UNDIRECTED GRAPHS

- An undirected graph is called *connected* if there is a path between every pair of distinct vertices of the graph.
- An undirected graph that is not *connected* is called *disconnected*.

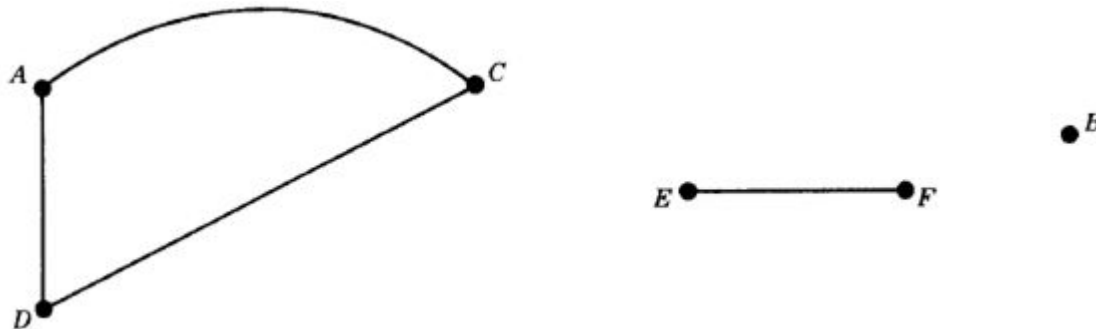


- Graph G_1 is connected as there is a path between every two vertices of G_1
- Graph G_2 is disconnected as there is no path between vertex c and d

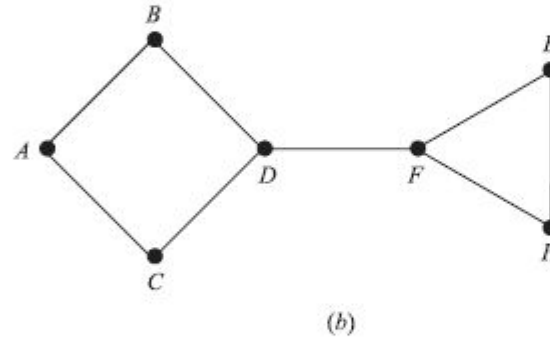
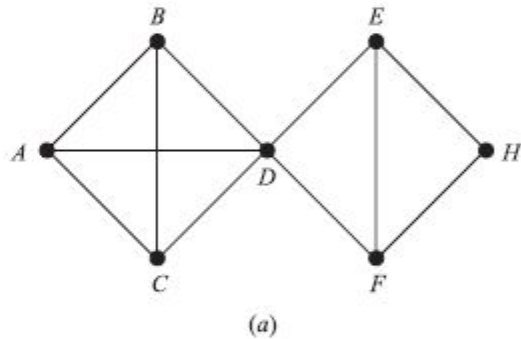
- Suppose G is a graph. A connected subgraph H of G is called a *connected component* of G if H is not contained in any larger connected subgraph of G .
- A graph G that is not connected has two or more connected components that are disjoint and have G as their union.



- Above graph has 03 connected components



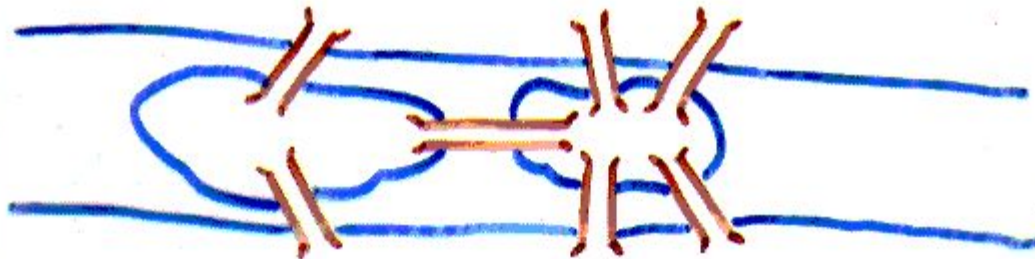
- Sometimes the removal from a graph of a vertex and all incident edges produces a subgraph with more connected components. Such vertices are called **cut vertices**.
- The removal of a cut vertex from a connected graph produces a subgraph that is not connected.
- An edge whose removal produces a graph with more connected components than in the original graph is called a **cut edge** or **bridge**.
- Let G is a complete graph, it has no cut point, because removing any of its vertex and all incident edges still leaves a complete graph



- In graph (a), vertex D is a cut point
- In graph (b), edge {D, F} is an bridge.

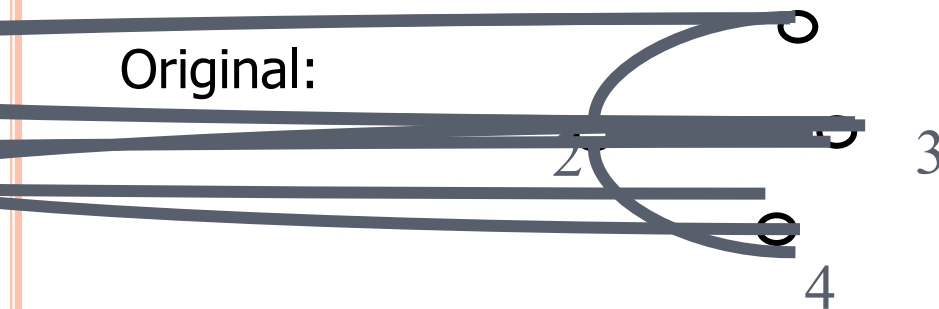
THE SEVEN BRIDGES OF KÖNIGSBERG, GERMANY

- The residents of Königsberg, Germany, wondered if it was possible to take a walking tour of the town that crossed each of the seven bridges over the Presel river exactly once. Is it possible to start at some node and take a walk that uses each edge exactly once, and ends at the starting node?

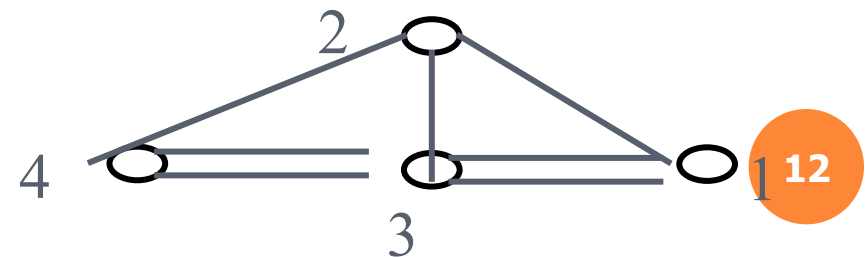


THE SEVEN BRIDGES OF KÖNIGSBERG, GERMANY

You can redraw the original picture as long as for every edge between nodes i and j in the original you put an edge between nodes i and j in the redrawn version (and you put no other edges in the redrawn version).

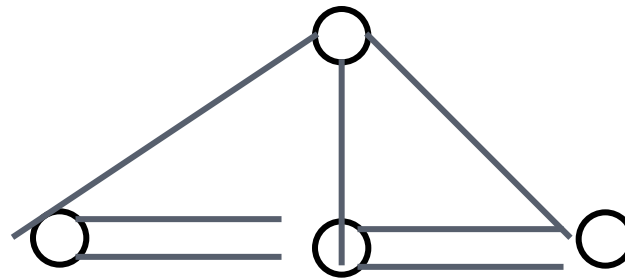


Redrawn:



THE SEVEN BRIDGES OF KÖNIGSBERG, GERMANY

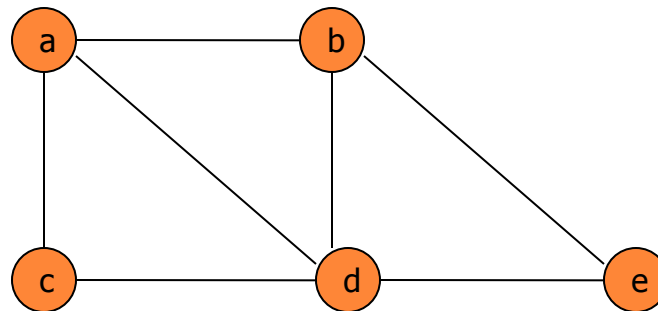
Euler:



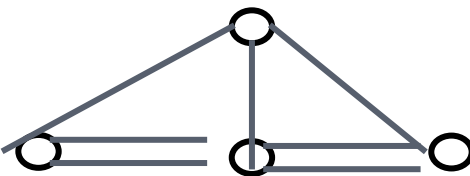
- Has no tour that uses each edge exactly once.
- (Even if we allow the walk to start and finish in different places.)
- Can you see why?

EULER - DEFINITIONS

- An **Eulerian path** (**Eulerian trail**, **Euler walk**) in a graph is a path that uses each edge precisely once. If such a path exists, the graph is called **traversable**.
- An **Eulerian cycle** (**Eulerian circuit**, **Euler tour**) in a graph is a cycle that uses each edge precisely once. If such a cycle exists, the graph is called **Eulerian** (also **unicursal**).
- Representation example: G1 has Euler path a, c, d, e, b, d, a, b



The problem in our language:

Show that  is not Eulerian.

In fact, it contains no Euler trail.

EULER - THEOREMS

1. A connected graph G is Eulerian if and only if G is connected and has no vertices of odd degree
2. A connected graph G has an Euler trail from node a to some other node b if and only if G is connected and $a \neq b$ are the only two nodes of odd degree

EULER – THEOREMS (\Rightarrow)

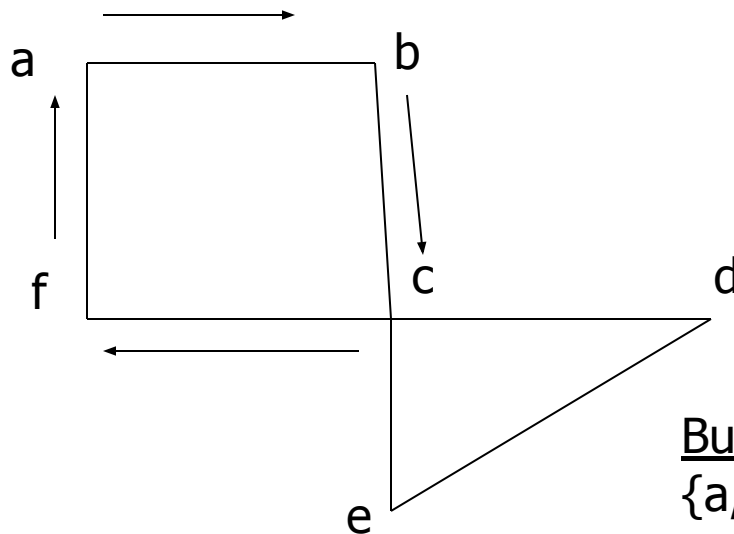
Assume G has an Euler trail T from node a to node b (a and b not necessarily distinct).

For every node besides a and b , T uses an edge to exit for each edge it uses to enter. Thus, the degree of the node is even.

1. If $a = b$, then a also has even degree. \square Euler circuit
2. If $a \neq b$, then a and b both have odd degree. \square Euler path

EULER - THEOREMS

1. A connected graph G is Eulerian if and only if G is connected and has no vertices of odd degree

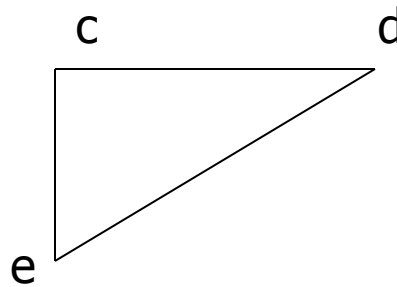


Building a simple path:
 $\{a,b\}, \{b,c\}, \{c,f\}, \{f,a\}$

Euler circuit constructed if all edges are used. True here?

EULER - THEOREMS

1. A connected graph G is Eulerian if and only if G is connected and has no vertices of odd degree

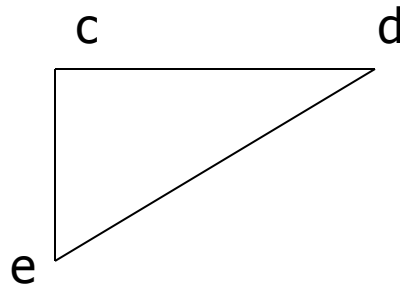


Delete the simple path:
 $\{a,b\}, \{b,c\}, \{c,f\}, \{f,a\}$

C is the common vertex for this sub-graph with its "parent".

EULER - THEOREMS

1. A connected graph G is Eulerian if and only if G is connected and has no vertices of odd degree



Constructed subgraph may not be connected.

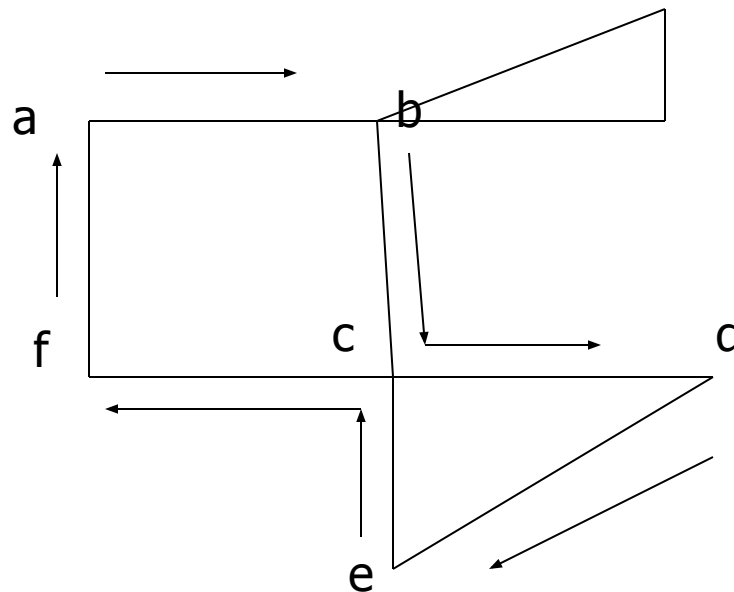
C is the common vertex for this sub-graph with its "parent".

C has even degree.

Start at c and take a walk:
 $\{c,d\}, \{d,e\}, \{e,c\}$

EULER - THEOREMS

1. A connected graph G is Eulerian if and only if G is connected and has no vertices of odd degree



"Splice" the circuits in the 2 graphs:

$\{a,b\}, \{b,c\}, \{c,f\}, \{f,a\}$

"+"

$\{c,d\}, \{d,e\}, \{e,c\}$

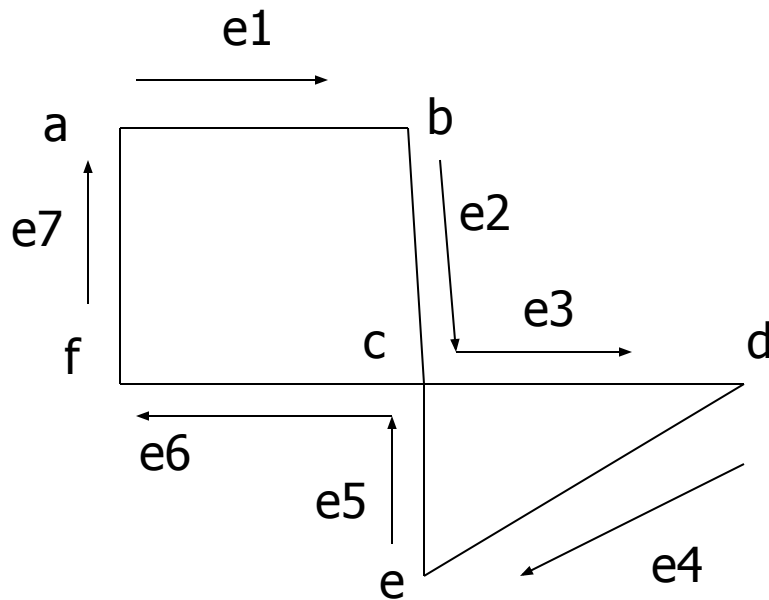
"="

$\{a,b\}, \{b,c\}, \{c,d\}, \{d,e\}, \{e,c\}, \{c,f\}, \{f,a\}$

EULER CIRCUIT

1. Circuit $C :=$ a circuit in G beginning at an arbitrary vertex v .
 1. Add edges successively to form a path that returns to this vertex.
2. $H := G - \text{above circuit } C$
3. While H has edges
 1. Sub-circuit $sc :=$ a circuit that begins at a vertex in H that is also in C (e.g., vertex “ c ”)
 2. $H := H - sc$ (- all isolated vertices)
 3. Circuit $:=$ circuit C “spliced” with sub-circuit sc
4. Circuit C has the Euler circuit.

REPRESENTATION- INCIDENCE MATRIX



	e_1	e_2	e_3	e_4	e_5	e_6	e_7
a	1	0	0	0	0	0	1
b	1	1	0	0	0	0	0
c	0	1	1	0	1	1	0
d	0	0	1	1	0	0	0
e	0	0	0	1	1	0	0
f	0	0	0	0	0	1	1

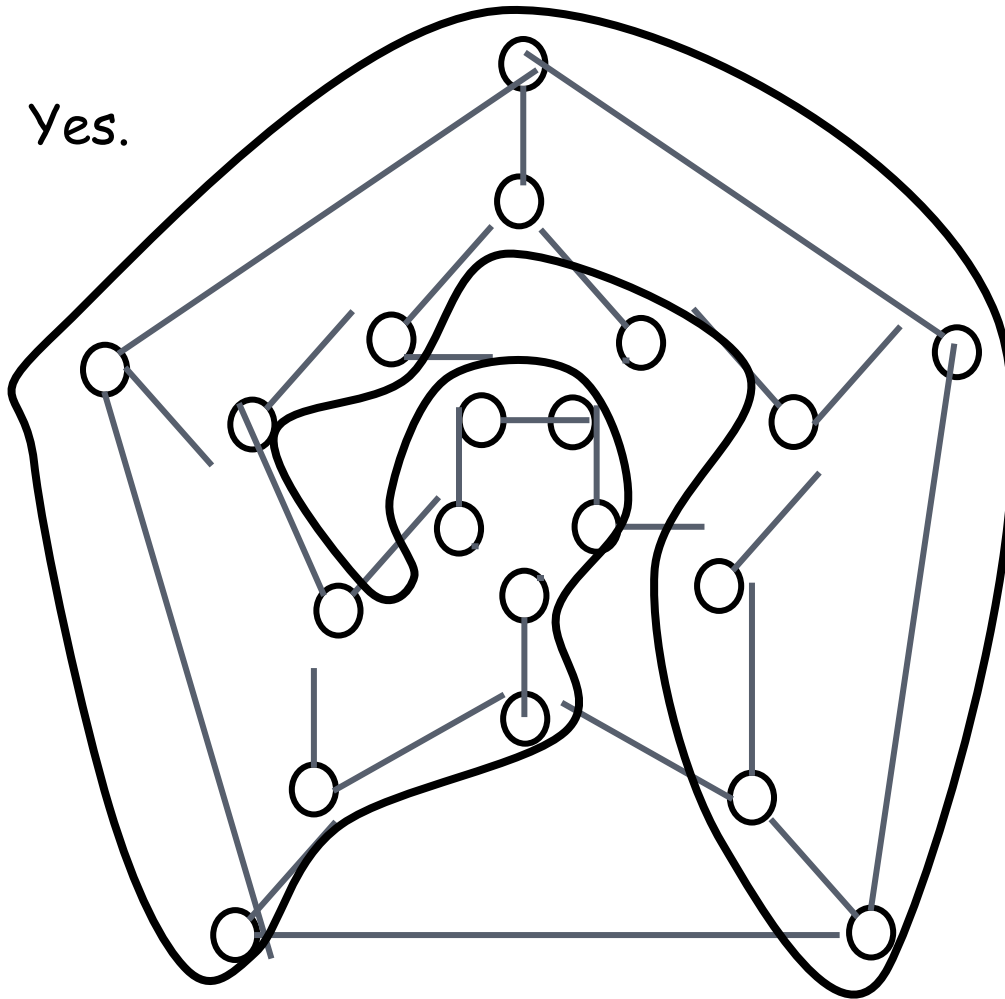
HAMILTONIAN GRAPH

- **Hamiltonian path** (also called *traceable path*) is a path that visits each vertex exactly once.
- A **Hamiltonian cycle** (also called *Hamiltonian circuit*, *vertex tour* or *graph cycle*) is a cycle that visits each vertex exactly once (except for the starting vertex, which is visited once at the start and once again at the end).
- A graph that contains a Hamiltonian path is called a **traceable graph**. A graph that contains a Hamiltonian cycle is called a **Hamiltonian graph**. Any Hamiltonian cycle can be converted to a Hamiltonian path by removing one of its edges, but a Hamiltonian path can be extended to Hamiltonian cycle only if its endpoints are adjacent.

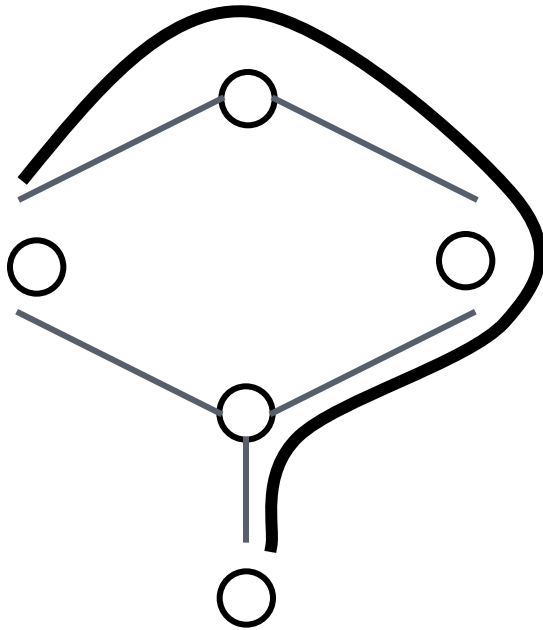
A graph of the vertices of a dodecahedron.

Is it Hamiltonian?

Yes.

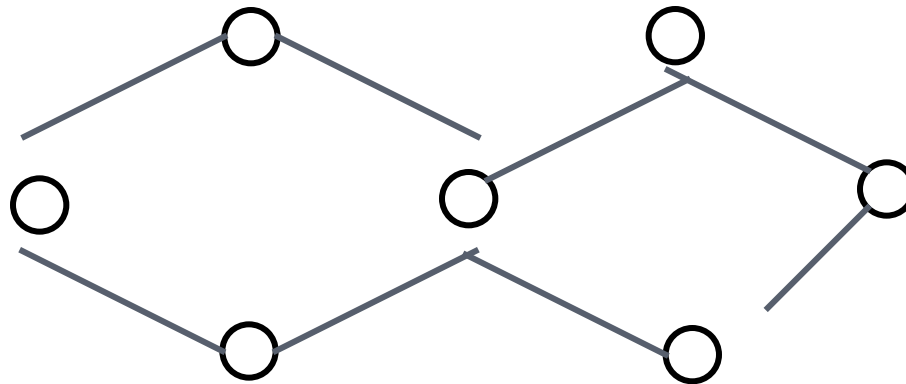


Hamiltonian Graph



THIS ONE HAS A HAMILTONIAN PATH, BUT NOT A HAMILTONIAN TOUR.

HAMILTONIAN GRAPH



This one has an Euler tour, but no Hamiltonian path.

HAMILTONIAN GRAPH

- Similar notions may be defined for directed graphs, where edges (arcs) of a path or a cycle are required to point in the same direction, i.e., connected tail-to-head.
- The *Hamiltonian cycle problem* or *Hamiltonian circuit problem* in graph theory is to find a Hamiltonian cycle in a given graph. The *Hamiltonian path problem* is to find a Hamiltonian path in a given graph.
- There is a simple relation between the two problems. The Hamiltonian path problem for graph G is equivalent to the Hamiltonian cycle problem in a graph H obtained from G by adding a new vertex and connecting it to all vertices of G .
- Both problems are NP-complete. However, certain classes of graphs always contain Hamiltonian paths. For example, it is known that every tournament has an odd number of Hamiltonian paths.

HAMILTONIAN GRAPH

- ❑ **DIRAC'S Theorem:** if G is a simple graph with n vertices with $n \geq 3$ such that the degree of every vertex in G is at least $n/2$ then G has a Hamilton circuit.
- ❑ **ORE'S Theorem:** if G is a simple graph with n vertices with $n \geq 3$ such that $\deg(u) + \deg(v) \geq n$ for every pair of nonadjacent vertices u and v in G , then G has a Hamilton circuit.

SHORTEST PATH

- Generalize distance to weighted setting
- Digraph $G = (V, E)$ with weight function $W: E \rightarrow R$ (assigning real values to edges)
- Weight of path $p = v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k$ is

$$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$$

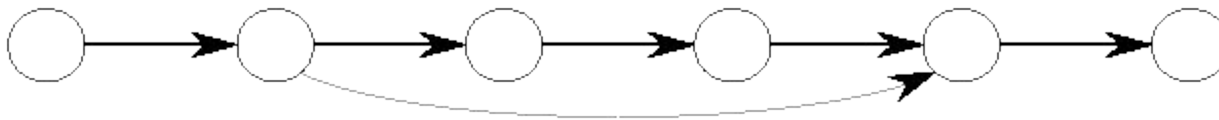
- Shortest path = a path of the minimum weight
- Applications
 - static/dynamic network routing
 - robot motion planning
 - map/route generation in traffic

SHORTEST-PATH PROBLEMS

- Shortest-Path problems
 - **Single-source (single-destination).** Find a shortest path from a given source (vertex s) to each of the vertices. The topic of this lecture.
 - **Single-pair.** Given two vertices, find a shortest path between them. Solution to single-source problem solves this problem efficiently, too.
 - **All-pairs.** Find shortest-paths for every pair of vertices. Dynamic programming algorithm.
 - Unweighted shortest-paths – BFS.

OPTIMAL SUBSTRUCTURE

- *Theorem*: subpaths of shortest paths are shortest paths
- Proof (“cut and paste”)
 - if some subpath were not the shortest path, one could substitute the shorter subpath and create a shorter total path



NEGATIVE WEIGHTS AND CYCLES?

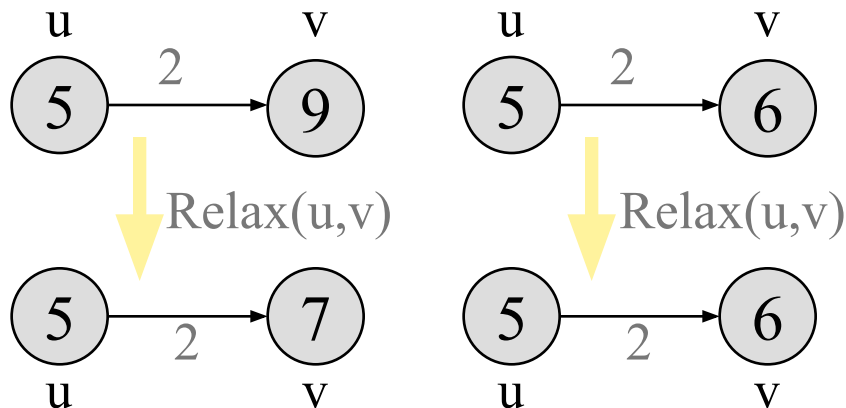
- Negative edges are OK, as long as there are no *negative weight cycles* (otherwise paths with arbitrary small “lengths” would be possible)
- Shortest-paths can have no cycles (otherwise we could improve them by removing cycles)
 - Any shortest-path in graph G can be no longer than $n - 1$ edges, where n is the number of vertices

SHORTEST PATH TREE

- The result of the algorithms – a *shortest path tree*. For each vertex v , it
 - records a shortest path from the start vertex s to v . $v.\text{parent}()$ gives a predecessor of v in this shortest path
 - gives a shortest path length from s to v , which is recorded in $v.\text{d}()$.
- The same pseudo-code assumptions are used.
- **Vertex** ADT with operations:
 - **adjacent():** *VertexSet*
 - **d():** *int* and **setd(k: int)**
 - **parent():** *Vertex* and **setparent(p: Vertex)**

RELAXATION

- For each vertex v in the graph, we maintain $v.d()$, the estimate of the shortest path from s , initialized to ∞ at the start
- Relaxing an edge (u, v) means testing whether we can improve the shortest path to v found so far by going through u



```
Relax ( $u, v, G$ )
if  $v.d() > u.d() + G.w(u, v)$  then
     $v.setd(u.d() + G.w(u, v))$ 
     $v.setparent(u)$ 
```

DIJKSTRA'S ALGORITHM

- Non-negative edge weights
- Greedy, similar to Prim's algorithm for MST
- Like breadth-first search (if all weights = 1, one can simply use BFS)
- Use Q , a priority queue ADT keyed by $v.d()$ (BFS used FIFO queue, here we use a PQ, which is re-organized whenever some d decreases)
- Basic idea
 - maintain a set S of solved vertices
 - at each step select "closest" vertex u , add it to S , and relax all edges from u

DIJKSTRA'S ALGORITHM

SOLUTION TO SINGLE-SOURCE (SINGLE-DESTINATION).

Input: Graph G , start vertex s

Dijkstra (G, s)

```
01 for each vertex  $u \in G.V()$ 
02      $u.setd(\infty)$ 
03      $u.setparent(NIL)$ 
04  $s.setd(0)$ 
05  $S \leftarrow \emptyset$  // Set  $S$  is used to explain the
    algorithm
06  $Q.init(G.V())$  //  $Q$  is a priority queue ADT
07 while not  $Q.isEmpty()$ 
08      $u \leftarrow Q.extractMin()$ 
09      $S \leftarrow S \cup \{u\}$ 
10     for each  $v \in u.adjacent()$  do
11         Relax ( $u, v, G$ )
12          $Q.modifyKey(v)$ 
```

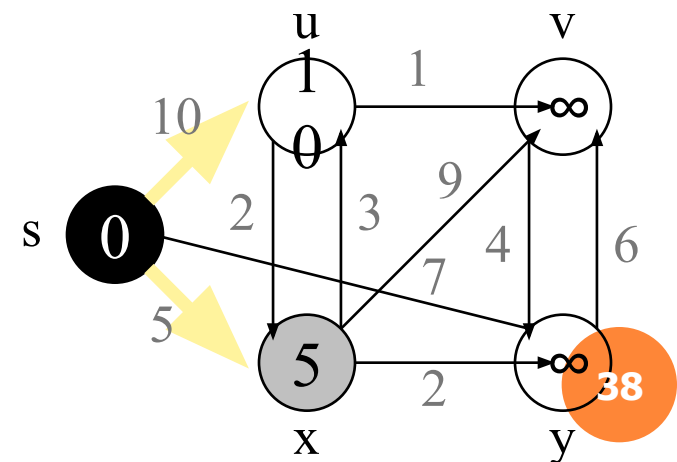
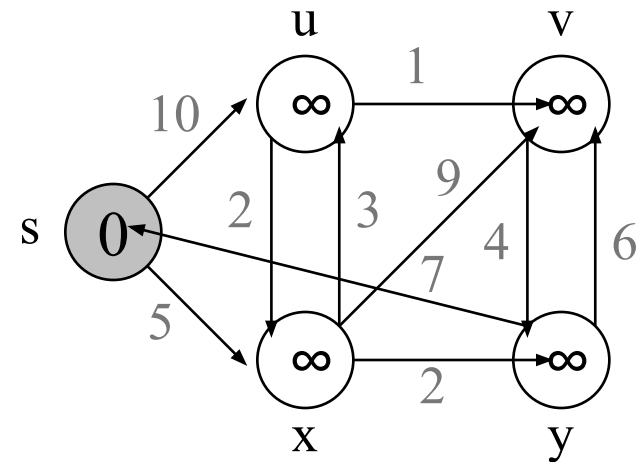
relaxing
edges

DIJKSTRA'S EXAMPLE

```

Dijkstra (G, s)
01 for each vertex  $u \in G.V()$ 
02      $u.setd(\infty)$ 
03      $u.setparent(NIL)$ 
04  $s.setd(0)$ 
05  $S \leftarrow \emptyset$ 
06  $Q.init(G.V())$ 
07 while not  $Q.isEmpty()$ 
08      $u \leftarrow Q.extractMin()$ 
09      $S \leftarrow S \cup \{u\}$ 
10     for each  $v \in u.adjacent()$  do
11         Relax(u, v, G)
12          $Q.modifyKey(v)$ 

```



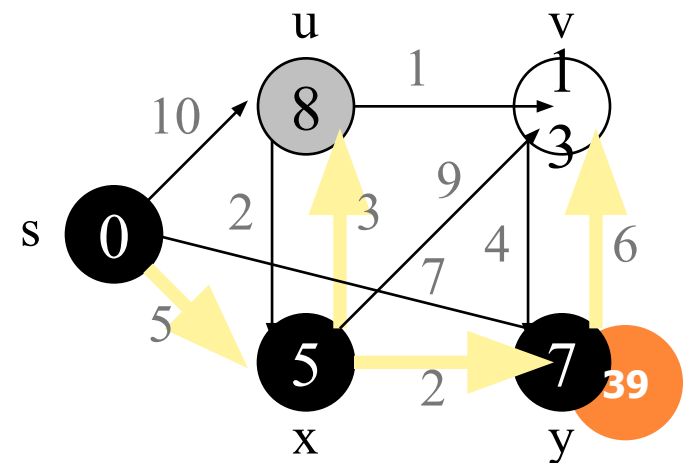
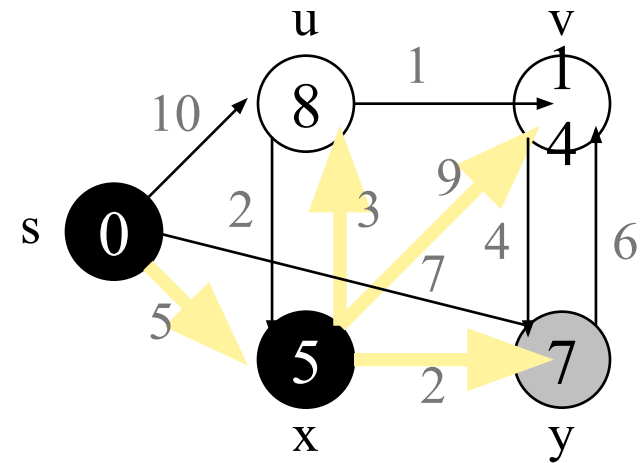
DIJKSTRA'S EXAMPLE

Dijkstra (G, s)

```

01 for each vertex  $u \in G.V()$ 
02      $u.setd(\infty)$ 
03      $u.setparent(NIL)$ 
04  $s.setd(0)$ 
05  $S \leftarrow \emptyset$ 
06  $Q.init(G.V())$ 
07 while not  $Q.isEmpty()$ 
08      $u \leftarrow Q.extractMin()$ 
09      $S \leftarrow S \cup \{u\}$ 
10     for each  $v \in u.adjacent()$  do
11         Relax ( $u, v, G$ )
12          $Q.modifyKey(v)$ 

```



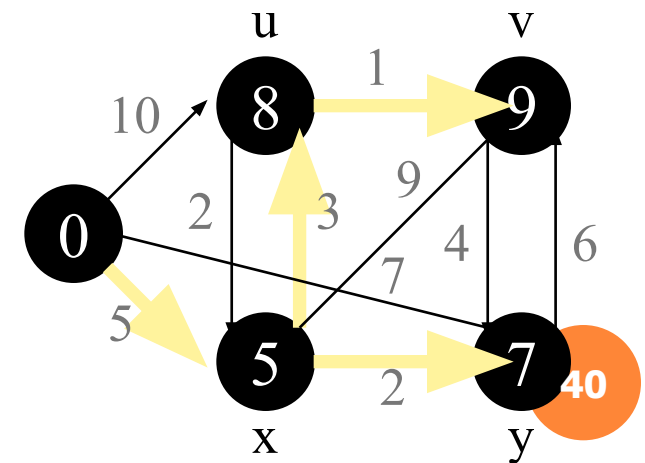
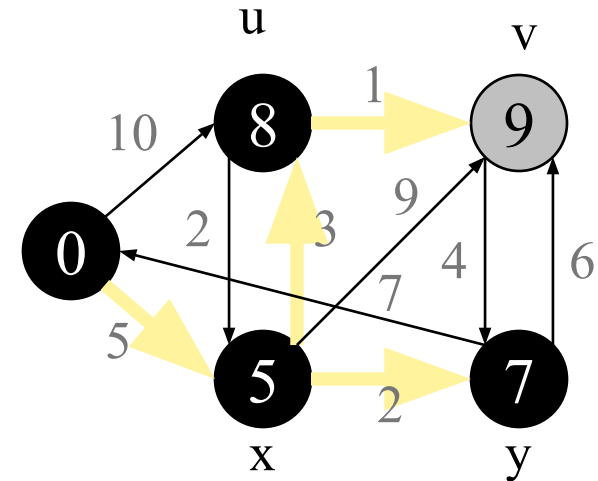
DIJKSTRA'S EXAMPLE

Dijkstra (G, s)

```

01 for each vertex  $u \in G.V()$ 
02      $u.setd(\infty)$ 
03      $u.setparent(NIL)$ 
04  $s.setd(0)$ 
05  $S \leftarrow \emptyset$ 
06  $Q.init(G.V())$ 
07 while not  $Q.isEmpty()$ 
08      $u \leftarrow Q.extractMin()$ 
09      $S \leftarrow S \cup \{u\}$ 
10     for each  $v \in u.adjacent()$  do
11         Relax ( $u, v, G$ )
12          $Q.modifyKey(v)$ 

```



DIJKSTRA'S ALGORITHM

- $O(n^2)$ operations
 - (n-1) iterations: 1 for each vertex added to the distinguished set S.
 - (n-1) iterations: for each adjacent vertex of the one added to the distinguished set.
- Note: it is single source – single destination algorithm

TRAVELING SALESMAN PROBLEM

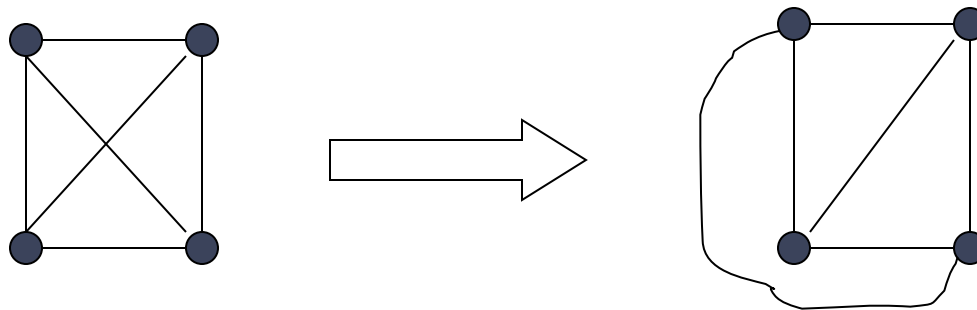
- Given a number of cities and the costs of traveling from one to the other, what is the cheapest roundtrip route that visits each city once and then returns to the starting city?
- An equivalent formulation in terms of graph theory is: Find the Hamiltonian cycle with the least weight in a weighted graph.
- It can be shown that the requirement of returning to the starting city does not change the computational complexity of the problem.
- A related problem is the (bottleneck TSP): Find the Hamiltonian cycle in a weighted graph with the minimal length of the longest edge.

PLANAR GRAPHS

- A graph (or multigraph) G is called *planar* if G can be drawn in the plane with its edges intersecting only at vertices of G , such a drawing of G is called an *embedding* of G in the plane.

Application Example: VLSI design (overlapping edges requires extra layers), Circuit design (cannot overlap wires on board)

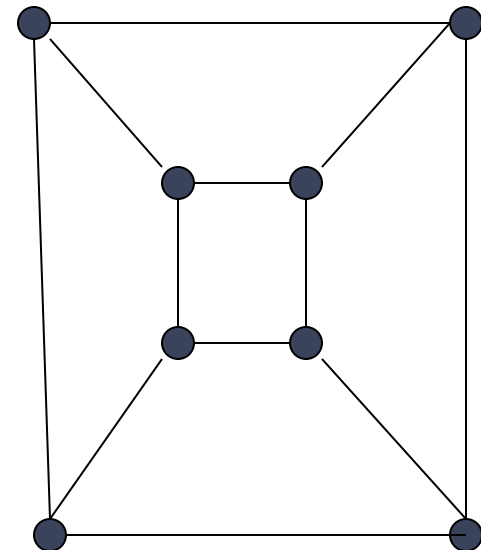
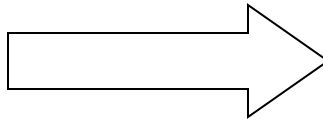
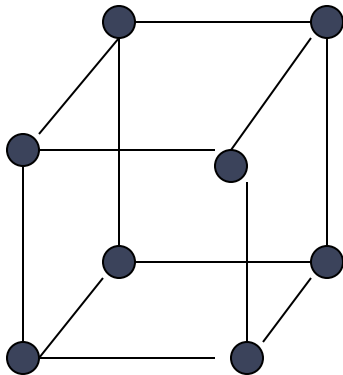
Representation examples: K_1, K_2, K_3, K_4 are planar, K_n for $n > 4$ are non-planar



K

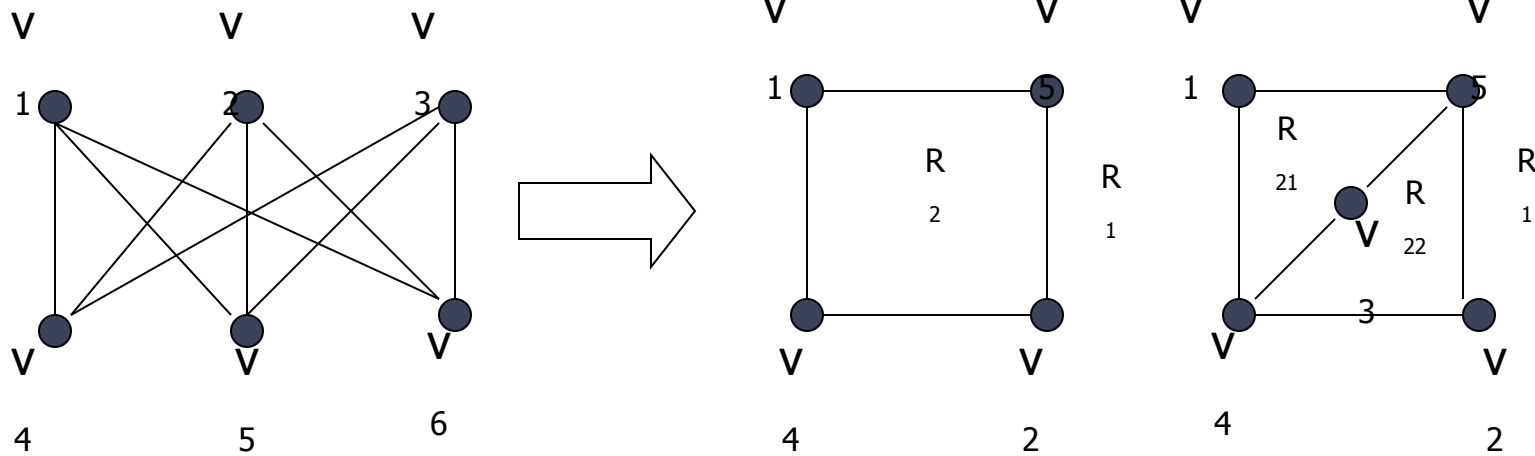
PLANAR GRAPHS

□ Representation examples: Q_3



PLANAR GRAPHS

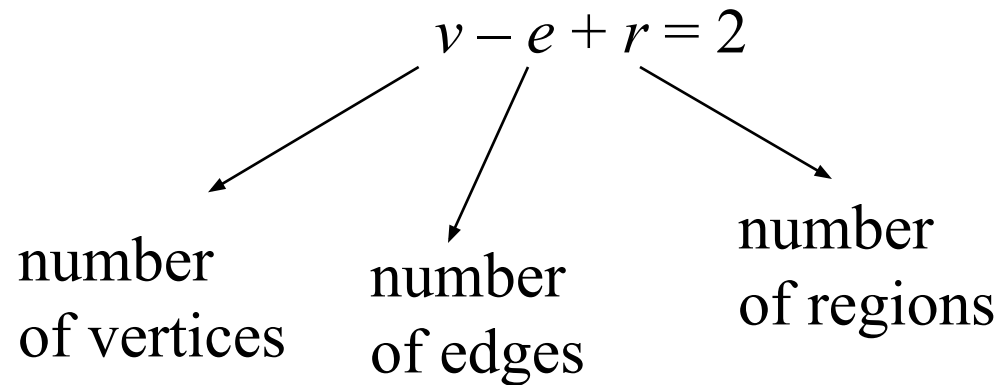
□ Representation examples: $K_{3,3}$ is Nonplanar



Planar Graphs

Theorem : *Euler's planar graph theorem*

For a **connected** planar graph or multigraph:

$$v - e + r = 2$$


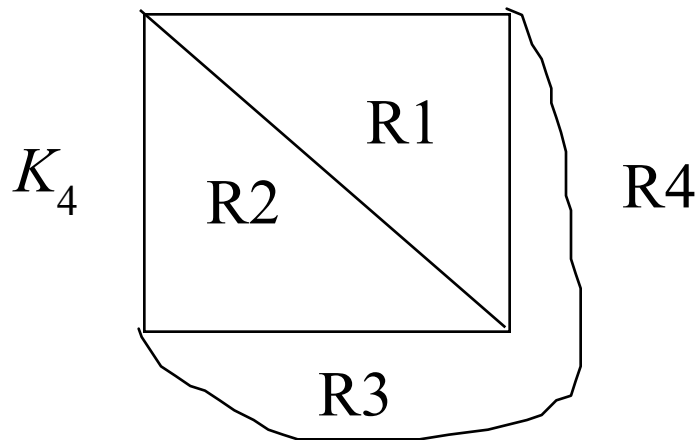
number
of vertices

number
of edges

number
of regions

PLANAR GRAPHS

Example of Euler's theorem



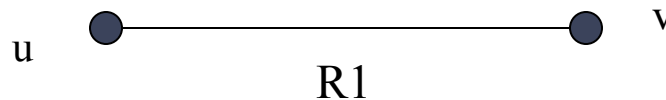
A planar graph divides the plane into several regions (faces), one of them is the infinite region.

$$v=4, e=6, r=4, v-e+r=2$$

PLANAR GRAPHS

□ Proof of Euler's formula: By Induction

Base Case: for G_1 , $e_1 = 1$, $v_1 = 2$ and $r_1 = 1$

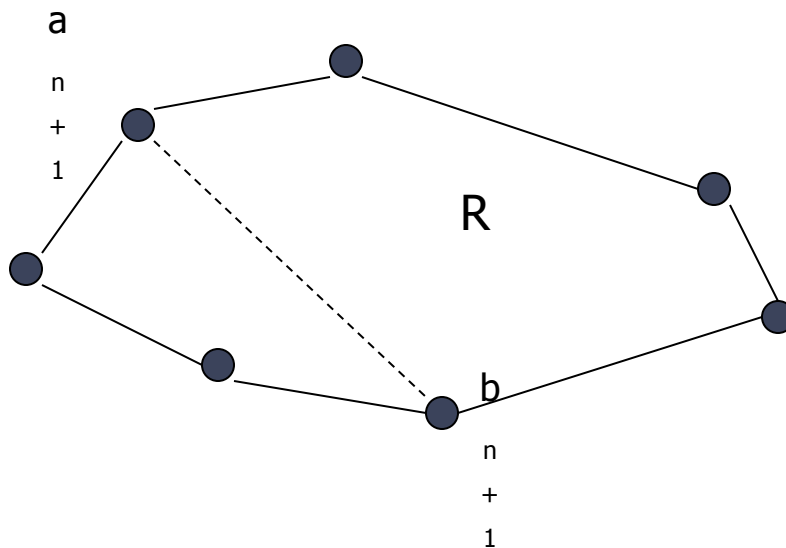


n+1 Case: Assume, $r_n = e_n - v_n + 2$ is true. Let $\{a_{n+1}, b_{n+1}\}$ be the edge that is added to G_n to obtain G_{n+1} and we prove that $r_n = e_n - v_n + 2$ is true. Can be proved using two cases.

PLANAR GRAPHS

□ Case 1:

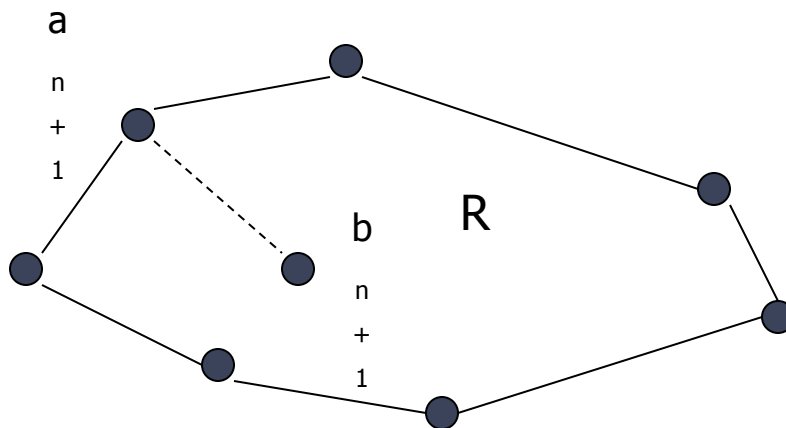
$$r_{n+1} = r_n + 1, e_{n+1} = e_n + 1, v_{n+1} = v_n \Rightarrow r_{n+1} = e_{n+1} - v_{n+1} + 2$$



PLANAR GRAPHS

□ Case 2:

$$r_{n+1} = r_n, e_{n+1} = e_n + 1, v_{n+1} = v_n + 1 \Rightarrow r_{n+1} = e_{n+1} - v_{n+1} + 2$$



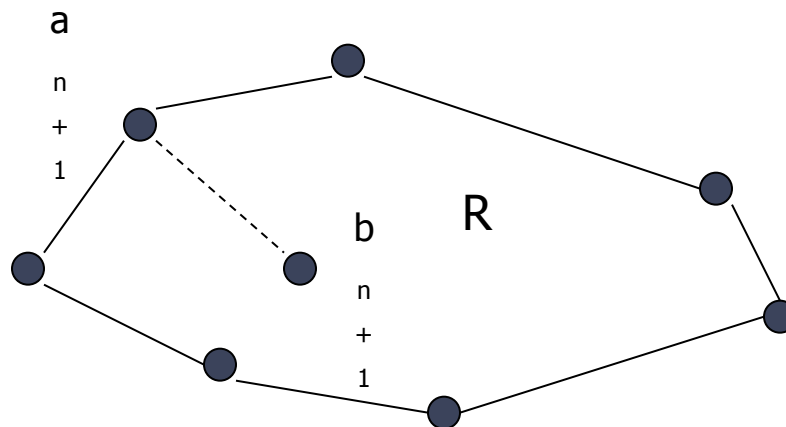
PLANAR GRAPHS

Corollary 1: Let $G = (V, E)$ be a connected simple planar graph with $|V| = v$, $|E| = e > 2$, and r regions. Then $3r \leq 2e$ and $e \leq 3v - 6$

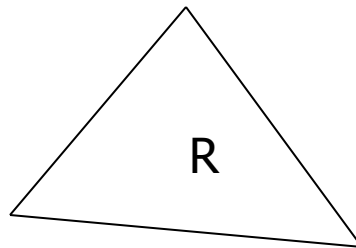
Proof: Since G is loop-free and is not a multigraph, the boundary of each region (including the infinite region) contains at least three edges. Hence, each region has degree ≥ 3 .

Degree of region: No. of edges on its boundary; 1 edge may occur twice on boundary \rightarrow contributes 2 to the region degree.

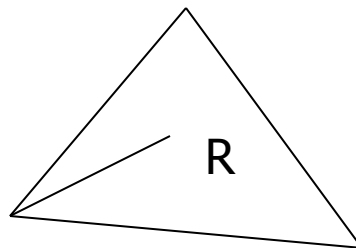
Each edge occurs exactly twice: either in the same region or in 2 different regions



REGION DEGREE



Degree of R = 3



Degree of R = ?

PLANAR GRAPHS

Each edge occurs exactly twice: either in the same region or in 2 different regions

$\Rightarrow 2e = \text{sum of degree of } r \text{ regions determined by } 2e$

$\Rightarrow 2e \geq 3r$. (since each region has a degree of at least 3)

$\Rightarrow r \leq (2/3) e$

\Rightarrow From Euler's theorem, $2 = v - e + r$

$\Rightarrow 2 \leq v - e + 2e/3$

$\Rightarrow 2 \leq v - e/3$

\Rightarrow So $6 \leq 3v - e$

\Rightarrow or $e \leq 3v - 6$

PLANAR GRAPHS

Corollary 2: Let $G = (V, E)$ be a connected simple planar graph then G has a vertex degree that does not exceed 5

Proof: If G has one or two vertices the result is true

If G has 3 or more vertices then by Corollary 1, $e \leq 3v - 6$

$$\Rightarrow 2e \leq 6v - 12$$

If the degree of every vertex were at least 6:

by Handshaking theorem: $2e = \text{Sum}(\text{deg}(v))$

$$\Rightarrow 2e \geq 6v. \text{ But this contradicts the inequality } 2e \leq 6v - 12$$

\Rightarrow There must be at least one vertex with degree no greater than 5

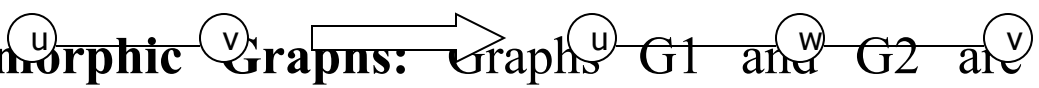
PLANAR GRAPHS

Corollary 3: Let $G = (V, E)$ be a connected simple planar graph with v vertices ($v \geq 3$), e edges, and no circuits of length 3 then $e \leq 2v - 4$

Proof: Similar to Corollary 1 except the fact that no circuits of length 3 imply that degree of region must be at least 4.

PLANAR GRAPHS

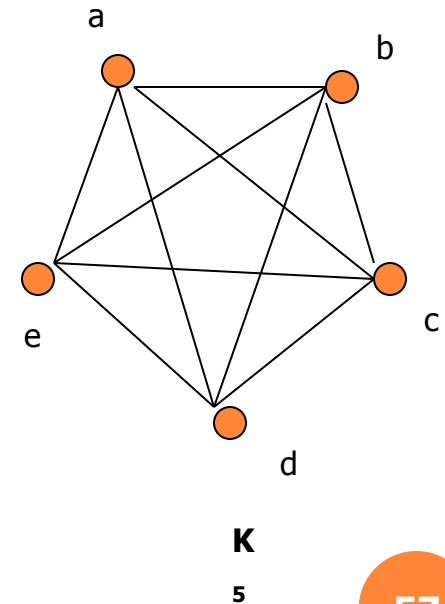
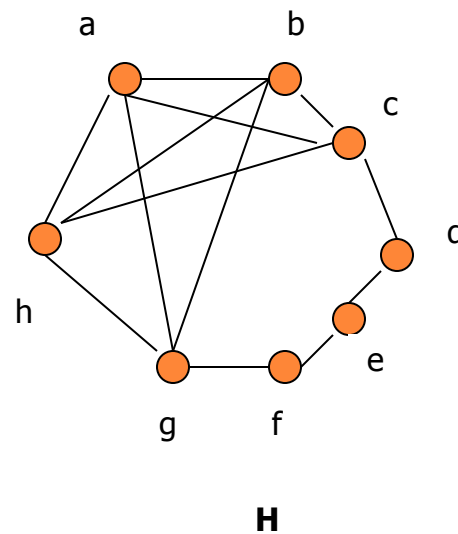
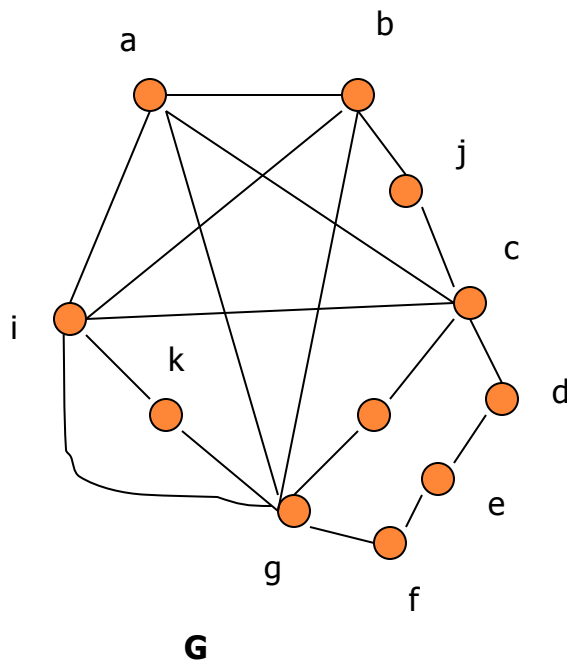
- **Elementary sub-division:** Operation in which a graph are obtained by removing an edge $\{u, v\}$ and adding the vertex w and edges $\{u, w\}$, $\{w, v\}$

- **Homeomorphic Graphs:**  Graphs G_1 and G_2 are termed as homeomorphic if they are obtained by sequence of elementary sub-divisions.

PLANAR GRAPHS

- ▣ **Kuwratoski's Theorem:** A graph is non-planar if and only if it contains a subgraph homeomorphic to $K_{3,3}$ or K_5

Representation Example: G is Nonplanar

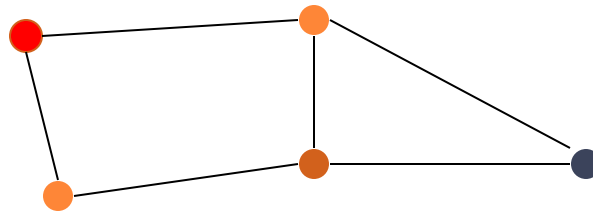


GRAPH COLORING PROBLEM

- **Graph coloring** is an assignment of "*colors*", almost always taken to be consecutive integers starting from 1 without loss of generality, to certain objects in a graph. Such objects can be vertices, edges, faces, or a mixture of the above.
- **Application examples:** scheduling, register allocation in a microprocessor, frequency assignment in mobile radios, and pattern matching

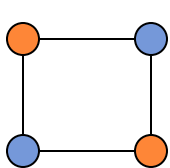
VERTEX COLORING PROBLEM

- Assignment of colors to the vertices of the graph such that proper coloring takes place (no two adjacent vertices are assigned the same color)
- **Chromatic number**: least number of colors needed to color the graph
- A graph that can be assigned a (proper) k -coloring is **k -colorable**, and it is **k -chromatic** if its chromatic number is exactly k .



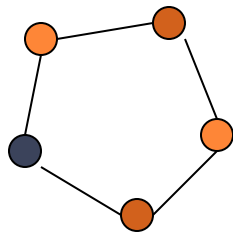
VERTEX COLORING PROBLEM

- The problem of finding a minimum coloring of a graph is NP-Hard
- The corresponding decision problem (Is there a coloring which uses at most k colors?) is NP-complete
- The chromatic number for $C_n = 3$ (n is odd) or 2 (n is even), $K_n = n$, $K_{m,n} = 2$
- C_n : cycle with n vertices; K_n : fully connected graph with n vertices; $K_{m,n}$: complete bipartite graph



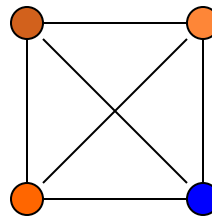
C

4



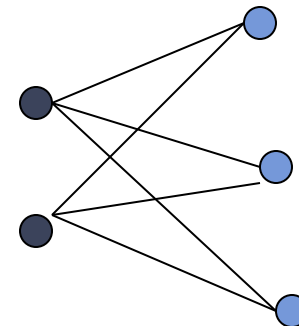
C

5



K

4

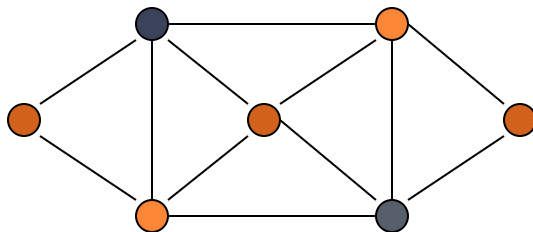


$K_{2,3}$

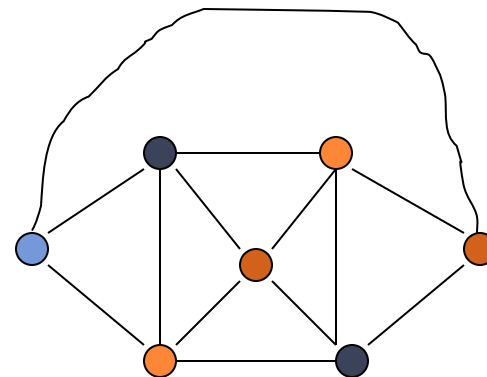
3

VERTEX COVERING PROBLEM

- **The Four color theorem:** the chromatic number of a planar graph is no greater than 4
- Example: G1 chromatic number = 3, G2 chromatic number = 4
- (Most proofs rely on case by case analysis).



G
1



G
2