南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

CS201 DISCRETE MATHEMATICS FOR COMPUTER SCIENCE

Dr. QI WANG

Department of Computer Science and Engineering
Office: Room413, CoE South Tower
Email: wangqi@sustech.edu.cn

Graph Concepts

- $G = (V, E)$, *simple* graph, *multigraph*, *pseudograph*
- *Undirected*, *directed* graph
- Special graphs

  K_n, C_n, W_n, Q_n, $K_{m,n}$

  Hall's Marriage Theorem on *bipartite* graphs

Graph Concepts

- $G = (V, E)$, *simple* graph, *multigraph*, *pseudograph*
- *Undirected*, *directed* graph
- Special graphs

  K_n, C_n, W_n, Q_n, $K_{m,n}$

  Hall's Marriage Theorem on *bipartite* graphs
- Representation of graphs

  adjacency list, *adjacency matrix*, *incidence matrix*

Isomorphism of Graphs

- **Definition** The simple graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are *isomorphic* if there is a one-to-one and onto function from V_1 to V_2 with the property that a and b are adjacent in G_1 if and only if $f(a)$ and $f(b)$ are adjacent in G_2, for all a and b in V_1. Such a function is called an *isomorphism*.

Isomorphism of Graphs

- **Definition** The simple graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are *isomorphic* if there is a one-to-one and onto function from V_1 to V_2 with the property that a and b are adjacent in G_1 if and only if $f(a)$ and $f(b)$ are adjacent in G_2, for all a and b in V_1. Such a function is called an *isomorphism*.

- Useful graph invariants include the number of vertices, number of edges, degree sequence, etc.

Cut Vertices and Cut Edges

- Sometimes the removal from a graph of a vertex and all incident edges disconnect the graph. Such vertices are called *cut vertices*. Similarly we may define *cut edges*.

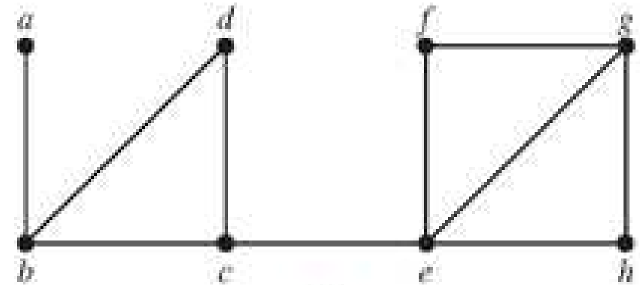
Cut Vertices and Cut Edges

- Sometimes the removal from a graph of a vertex and all incident edges disconnect the graph. Such vertices are called *cut vertices*. Similarly we may define *cut edges*.

  A set of edges E' is called an *edge cut* of G if the subgraph $G - E'$ is disconnected. The *edge connectivity* $\lambda(G)$ is the minimum number of edges in an edge cut of G.

- Sometimes the removal from a graph of a vertex and all incident edges disconnect the graph. Such vertices are called *cut vertices*. Similarly we may define *cut edges*.

  A set of edges E' is called an *edge cut* of G if the subgraph $G - E'$ is disconnected. The *edge connectivity* $\lambda(G)$ is the minimum number of edges in an edge cut of G.



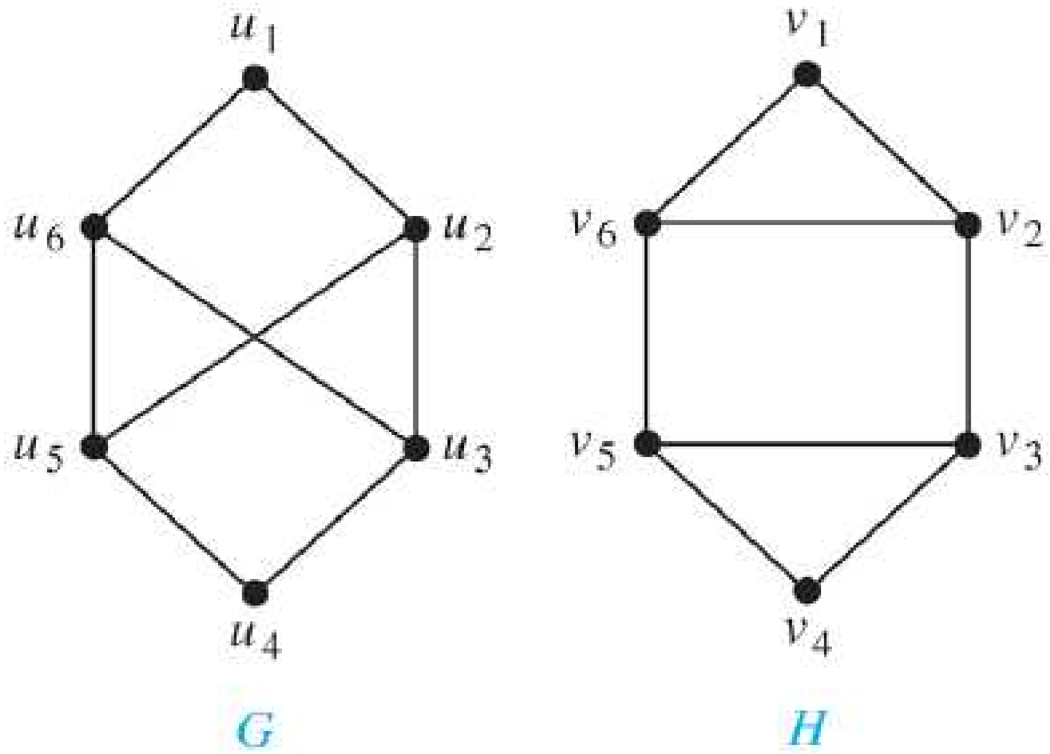
G_1

Paths and Isomorphism

- The existence of a simple circuit of length k is isomorphic invariant. In addition, paths can be used to construct mappings that may be isomorphisms.

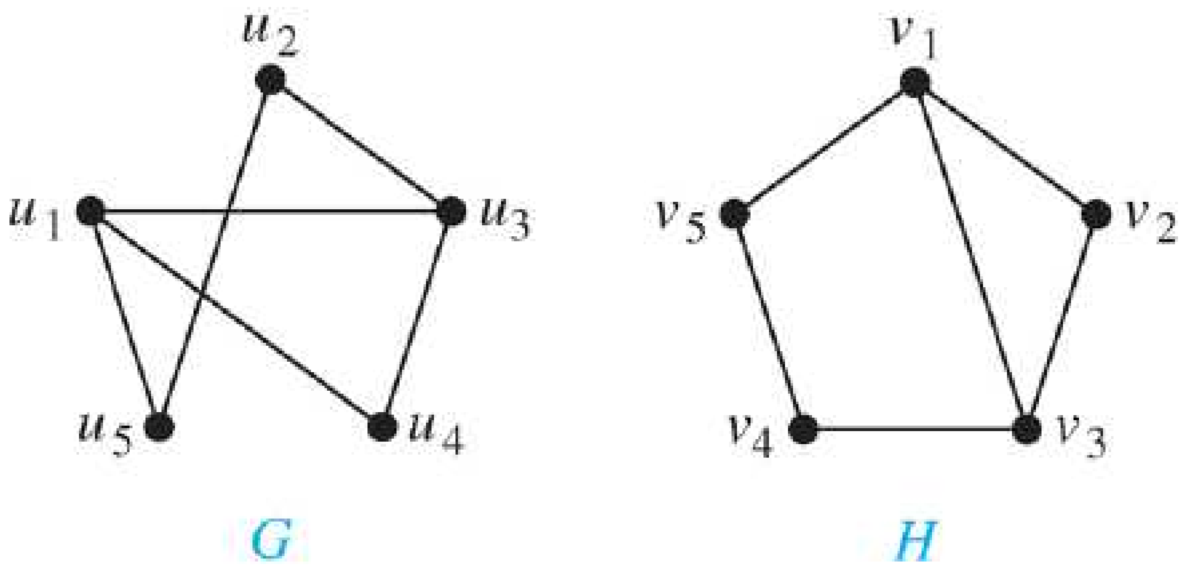
Paths and Isomorphism

- The existence of a simple circuit of length k is isomorphic invariant. In addition, paths can be used to construct mappings that may be isomorphisms.



Paths and Isomorphism

- The existence of a simple circuit of length k is isomorphic invariant. In addition, paths can be used to construct mappings that may be isomorphisms.



Counting Paths between Vertices

- **Theorem** Let G be a graph with adjacency matrix $\mathbf{A}$ with respect to the ordering $v_1, v_2, \ldots, v_n$ of vertices. The number of different paths of length r from v_i to v_j, where $r > 0$ is positive, equals the (i, j)-th entry of $\mathbf{A}^r$.

Counting Paths between Vertices

- **Theorem** Let G be a graph with adjacency matrix $\mathbf{A}$ with respect to the ordering $v_1, v_2, \ldots, v_n$ of vertices. The number of different paths of length r from v_i to v_j, where $r > 0$ is positive, equals the (i, j)-th entry of $\mathbf{A}^r$.

  Proof (by **induction**)

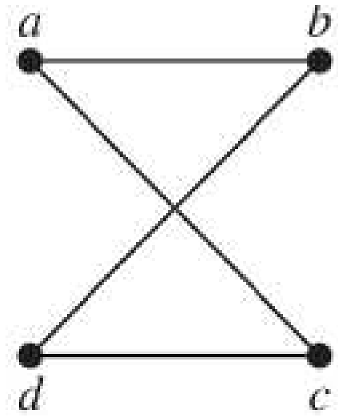
Counting Paths between Vertices

- **Theorem** Let G be a graph with adjacency matrix $\mathbf{A}$ with respect to the ordering $v_1, v_2, \ldots, v_n$ of vertices. The number of different paths of length r from v_i to v_j, where $r > 0$ is positive, equals the (i, j)-th entry of $\mathbf{A}^r$.

Proof (by **induction**)

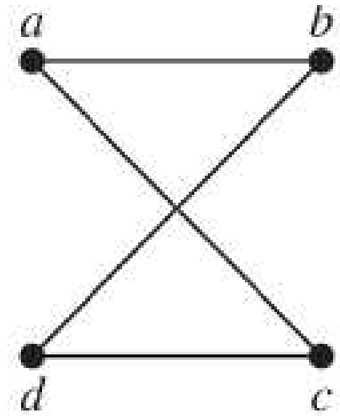
$\mathbf{A}^{r+1} = \mathbf{A}^r\mathbf{A}$, the (i, j)-th entry of $\mathbf{A}^{r+1}$ equals $b_{i1}a_{1j} + b_{i2}a_{2j} + \cdots + b_{in}a_{nj}$, where b_{ik} is the (i, k)-th entry of $\mathbf{A}^r$.

- **Example** How many paths of length 4 are there from a to d in the graph G?



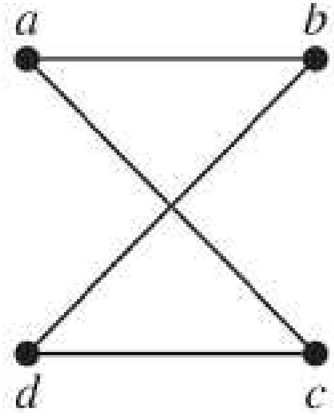
Counting Paths between Vertices

- **Example** How many paths of length 4 are there from a to d in the graph G?



$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

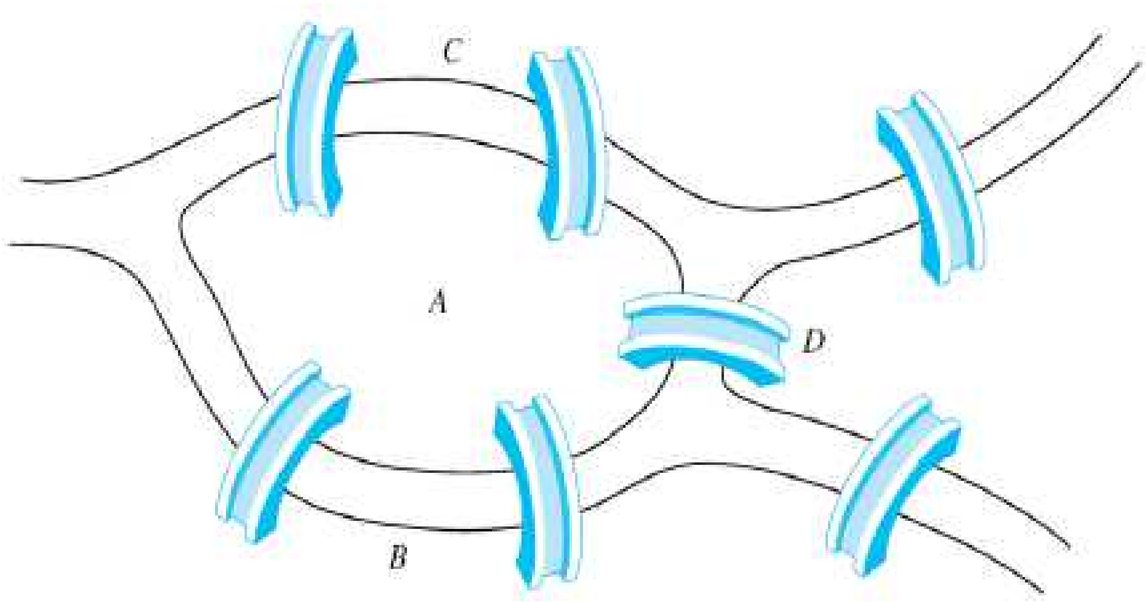
- **Example** How many paths of length 4 are there from a to d in the graph G?



$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \qquad \begin{bmatrix} 8 & 0 & 0 & 8 \\ 0 & 8 & 8 & 0 \\ 0 & 8 & 8 & 0 \\ 8 & 0 & 0 & 8 \end{bmatrix}$$

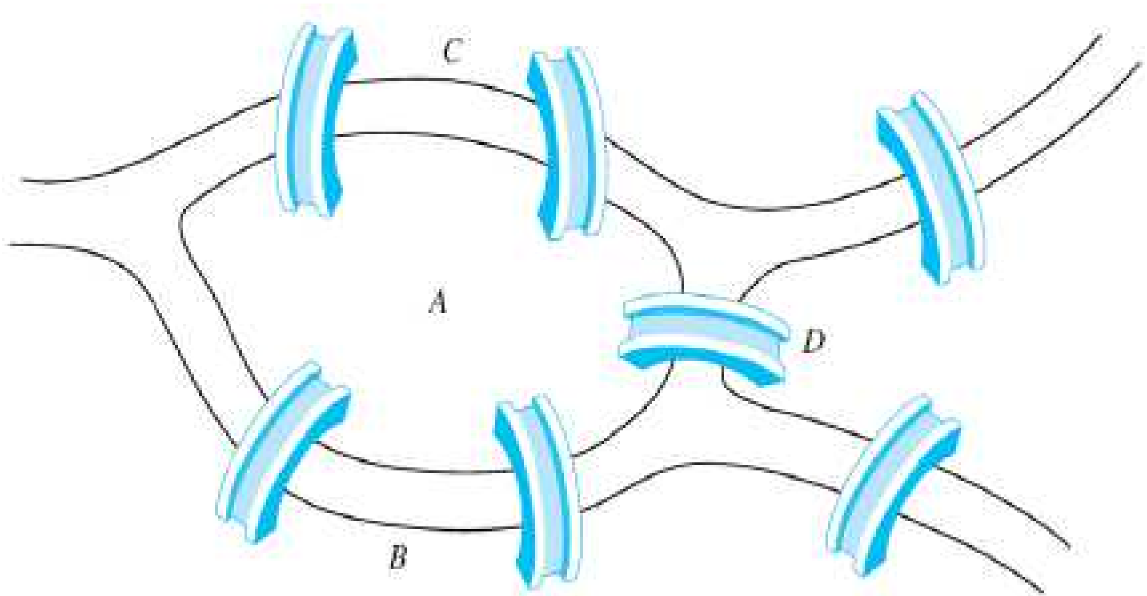
- **Königsberg seven-bridge problem**

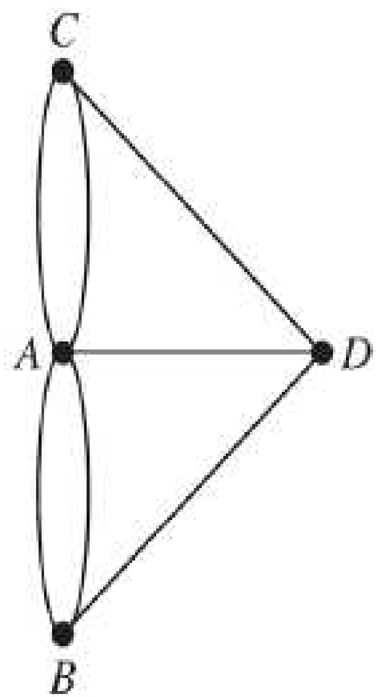
  People wondered whether it was possible to start at some location in the town, travel across all the bridges once without crossing any bridge twice, and return to the starting point.



- **Königsberg seven-bridge problem**

  People wondered whether it was possible to start at some location in the town, travel across all the bridges once without crossing any bridge twice, and return to the starting point.



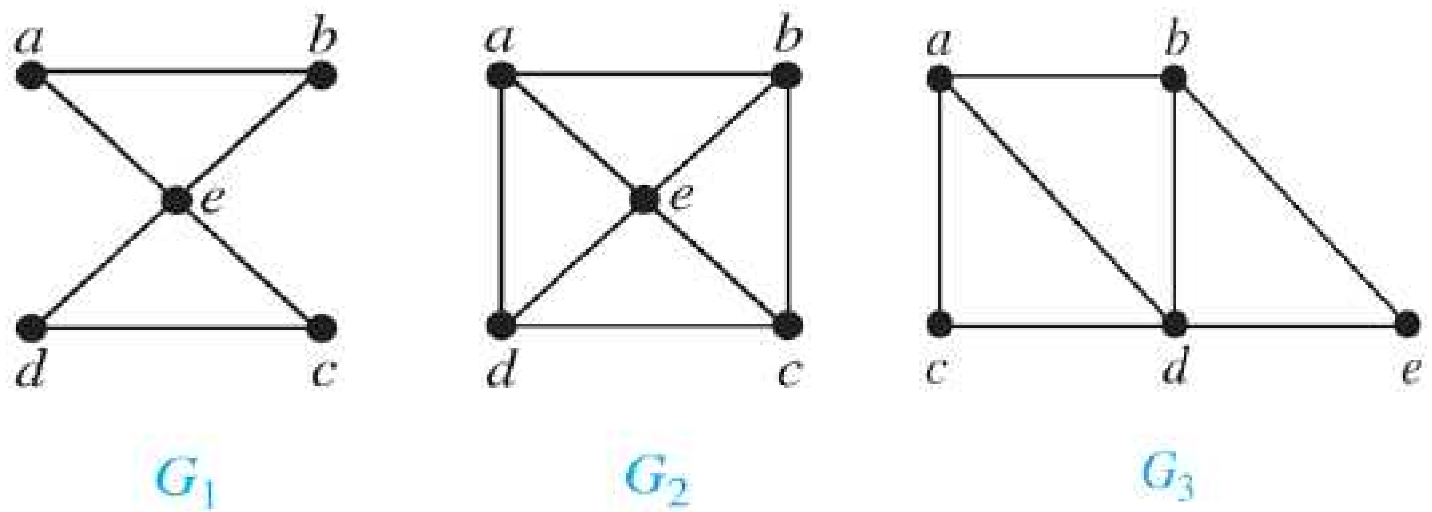


Euler Paths and Circuits

- **Definition** An *Euler circuit* in a graph G is a simple circuit containing every edge of G. An *Euler path* in G is a simple path containing every edge of G.

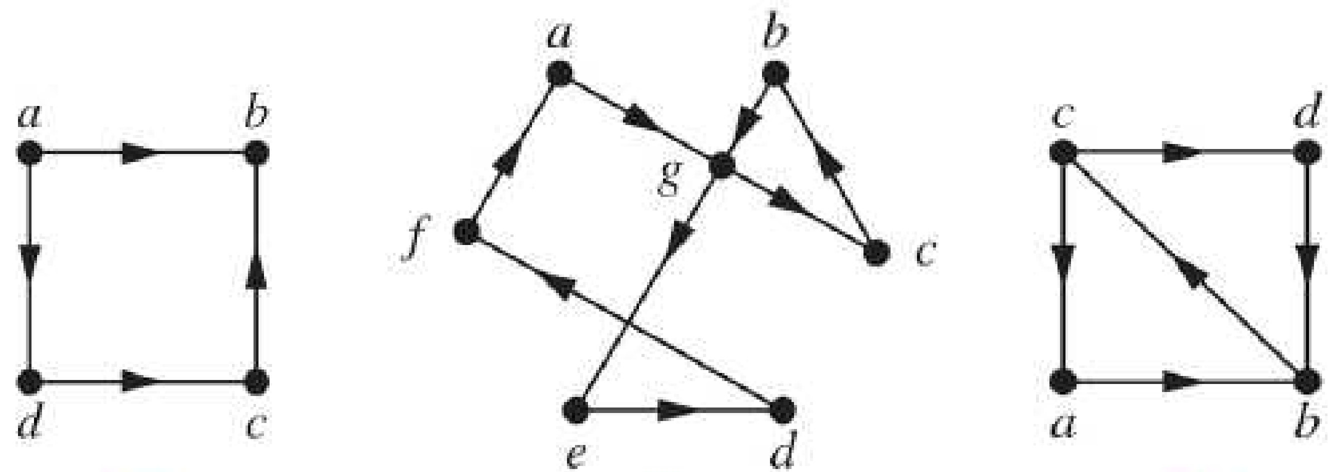
- **Definition** An *Euler circuit* in a graph G is a simple circuit containing every edge of G. An *Euler path* in G is a simple path containing every edge of G.

Example Which of the undirected graphs have an Euler circuit? Of those that do not, which have an Euler path?



- **Definition** An *Euler circuit* in a graph G is a simple circuit containing every edge of G. An *Euler path* in G is a simple path containing every edge of G.

Example Which of the directed graphs have an Euler circuit? Of those that do not, which have an Euler path?



H_1 H_2 H_3

Necessary Conditions for Euler Circuits and Paths

- Euler Circuit $\Rightarrow$ The degree of every vertex must be even

Necessary Conditions for Euler Circuits and Paths

- Euler Circuit $\Rightarrow$ The degree of every vertex must be even
  - Each time the circuit passes through a vertex, it contributes two to the vertex's degree.

Necessary Conditions for Euler Circuits and Paths

- Euler Circuit $\Rightarrow$ The degree of every vertex must be even
  - Each time the circuit passes through a vertex, it contributes two to the vertex's degree.
  - The circuit starts with a vertex a and ends at a, then contributes two to $\deg(a)$.

Necessary Conditions for Euler Circuits and Paths

- Euler Circuit $\Rightarrow$ The degree of every vertex must be even
  - ◇ Each time the circuit passes through a vertex, it contributes two to the vertex's degree.
  - ◇ The circuit starts with a vertex a and ends at a, then contributes two to $\deg(a)$.

Euler Path $\Rightarrow$ The graph has exactly two vertices of odd degree

Necessary Conditions for Euler Circuits and Paths

- Euler Circuit $\Rightarrow$ The degree of every vertex must be even
  - Each time the circuit passes through a vertex, it contributes two to the vertex's degree.
  - The circuit starts with a vertex a and ends at a, then contributes two to $\deg(a)$.

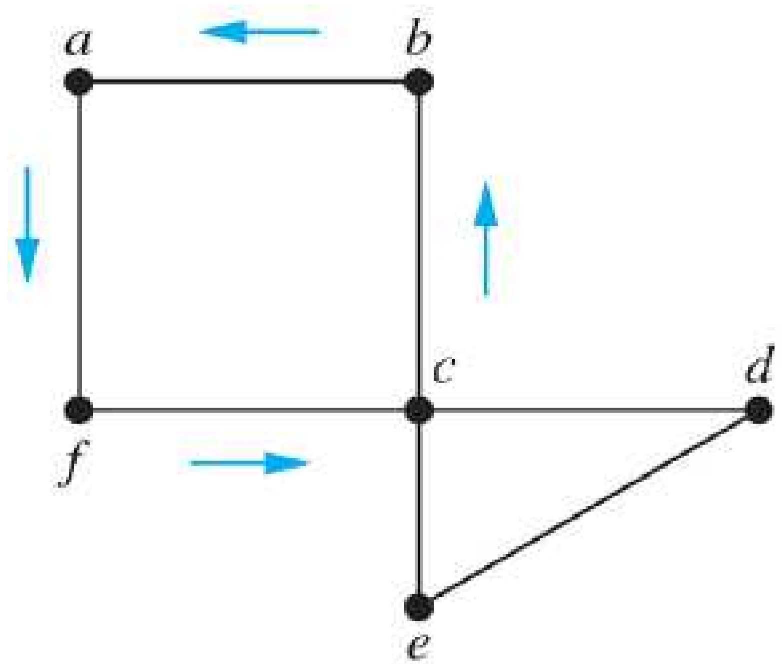
Euler Path $\Rightarrow$ The graph has exactly two vertices of odd degree

- The initial vertex and the final vertex of an Euler path have odd degree.

Sufficient Conditions for Euler Circuits and Paths

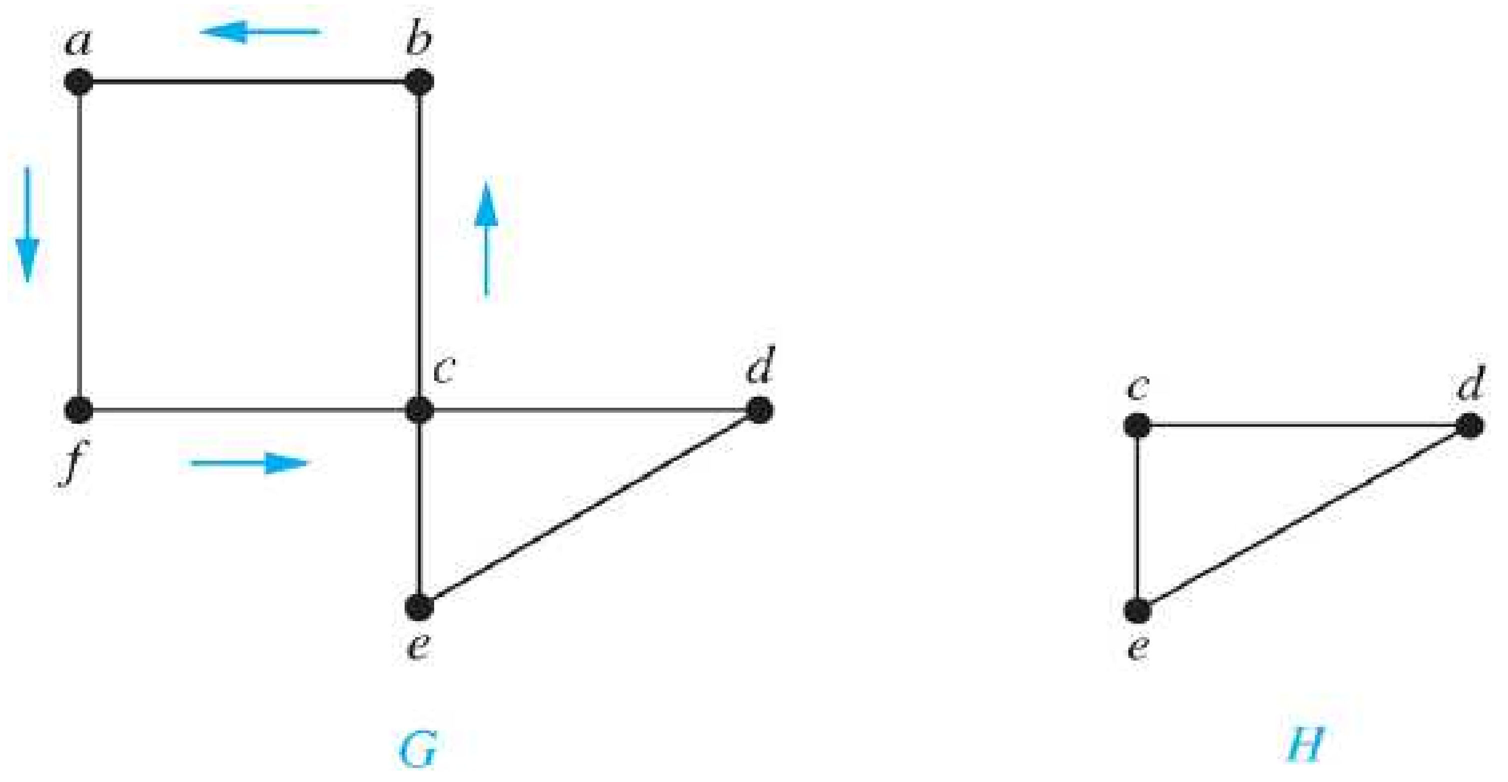
- Suppose that G is a connected multigraph with ≥ 2 vertices, all of even degree.

- Suppose that G is a connected multigraph with ≥ 2 vertices, all of even degree.



G

- Suppose that G is a connected multigraph with ≥ 2 vertices, all of even degree.



Algorithm for Constructing an Euler Circuit

procedure *Euler*(G: connected multigraph with all vertices of even degree)
circuit := a circuit in G beginning at an arbitrarily chosen vertex with edges
successively added to form a path that returns to this vertex.
H := G with the edges of this circuit removed
while H has edges
subciruit := a circuit in H beginning at a vertex in H that also is
an endpoint of an edge in circuit.
H := H with edges of *subciruit* and all isolated vertices removed
circuit := *circuit* with *subcircuit* inserted at the appropriate vertex.
return *circuit*{*circuit* is an Euler circuit}

Algorithm for Constructing an Euler Circuit

procedure *Euler*(G: connected multigraph with all vertices of even degree)
circuit := a circuit in G beginning at an arbitrarily chosen vertex with edges successively added to form a path that returns to this vertex.
$H := G$ with the edges of this circuit removed
while H has edges
  subciruit := a circuit in H beginning at a vertex in H that also is an endpoint of an edge in circuit.
  $H := H$ with edges of *subciruit* and all isolated vertices removed
  circuit := *circuit* with *subcircuit* inserted at the appropriate vertex.
return *circuit*{*circuit* is an Euler circuit}

Algorithm for Constructing an Euler Circuit

procedure *Euler*(G: connected multigraph with all vertices of even degree)
circuit := a circuit in G beginning at an arbitrarily chosen vertex with edges successively added to form a path that returns to this vertex.
H := G with the edges of this circuit removed
while H has edges
  subciruit := a circuit in H beginning at a vertex in H that also is an endpoint of an edge in circuit.
  H := H with edges of *subciruit* and all isolated vertices removed
  circuit := *circuit* with *subcircuit* inserted at the appropriate vertex.
return *circuit*{*circuit* is an Euler circuit}

Necessary and Sufficient Conditions

- **Theorem** A connected multigraph with at least two vertices has an *Euler circuit* if and only if each of its vertices has **even** degree.

Necessary and Sufficient Conditions

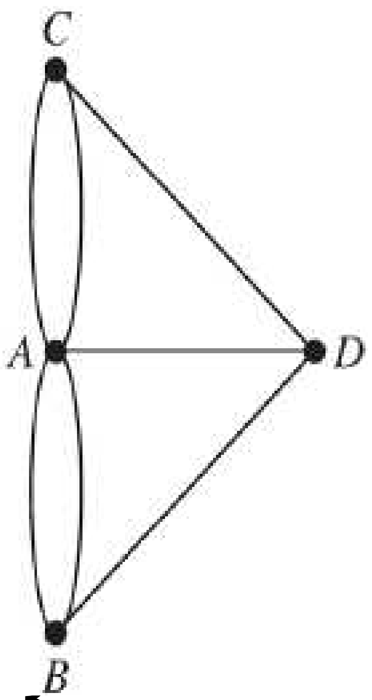
- **Theorem** A connected multigraph with at least two vertices has an *Euler circuit* if and only if each of its vertices has **even** degree.

  Theorem A connected multigraph has an *Euler path* but not an *Euler circuit* if and only if it has exactly two vertices of odd degree.

Necessary and Sufficient Conditions

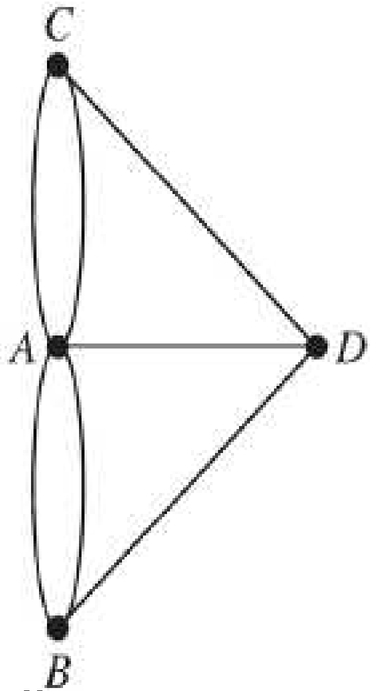
- **Theorem** A connected multigraph with at least two vertices has an *Euler circuit* if and only if each of its vertices has **even** degree.

  Theorem A connected multigraph has an *Euler path* but not an *Euler circuit* if and only if it has exactly two vertices of odd degree.



- **Theorem** A connected multigraph with at least two vertices has an *Euler circuit* if and only if each of its vertices has **even** degree.

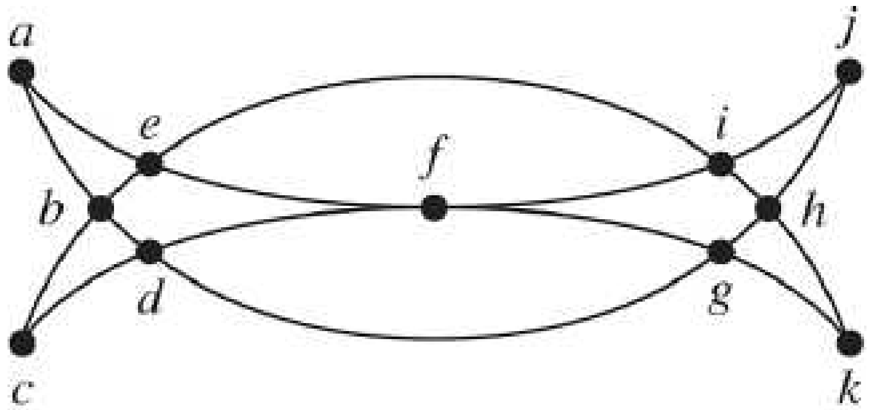
  Theorem A connected multigraph has an *Euler path* but not an *Euler circuit* if and only if it has exactly two vertices of odd degree.



No Euler circuit

Euler Circuits and Paths

- **Example**

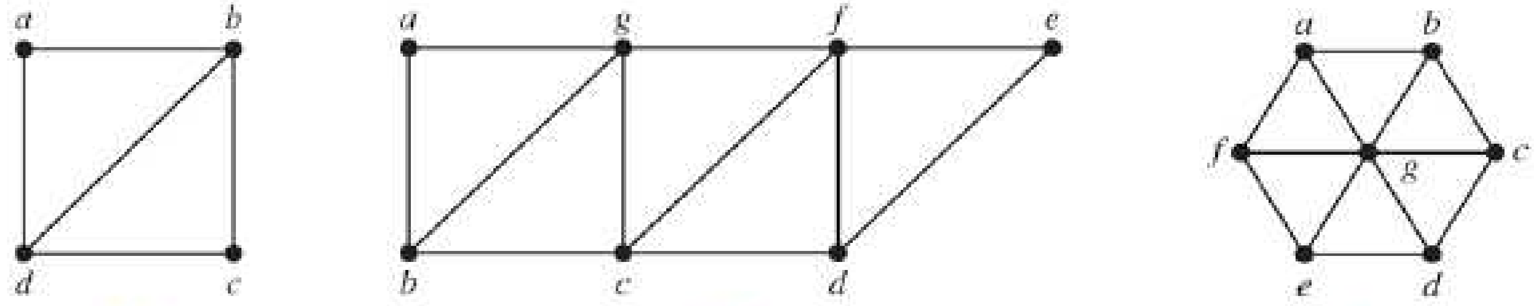


G

FIGURE 6 **Mohammed's Scimitars.**

Euler Circuits and Paths

Example



G_1 G_2 G_3

Applications of Euler Paths and Circuits

- Finding a path or circuit that traverses each
  - street in a neighborhood
  - road in a transportation network
  - link in a communication network
  - ...

Applications of Euler Paths and Circuits

- Finding a path or circuit that traverses each
  - ◇ street in a neighborhood
  - ◇ road in a transportation network
  - ◇ link in a communication network
  - ◇ ...

Chinese Postman Problem Meigu Guan [60']

Applications of Euler Paths and Circuits

- Finding a path or circuit that traverses each
  - ◇ street in a neighborhood
  - ◇ road in a transportation network
  - ◇ link in a communication network
  - ◇ ...

Chinese Postman Problem Meigu Guan [60']

Given a graph $G = (V, E)$, for every $e \in E$, there is a nonnegative weight $w(e)$. Find a circuit W such that

$$\sum_{e \in W} w(e) = \min$$

Applications of Euler Paths and Circuits

- Finding a path or circuit that traverses each
  - ◇ street in a neighborhood
  - ◇ road in a transportation network
  - ◇ link in a communication network
  - ◇ ...

Chinese Postman Problem Meigu Guan [60']

Given a graph $G = (V, E)$, for every $e \in E$, there is a nonnegative weight $w(e)$. Find a circuit W such that

$$\sum_{e \in W} w(e) = \min$$

k-Postman Chinese Postman Problem (k-PCPP)

Applications of Euler Paths and Circuits

- Finding a path or circuit that traverses each
  - street in a neighborhood
  - road in a transportation network
  - link in a communication network
  - ...

Chinese Postman Problem Meigu Guan [60']

Given a graph $G = (V, E)$, for every $e \in E$, there is a nonnegative weight $w(e)$. Find a circuit W such that

$$\sum_{e \in W} w(e) = \min$$

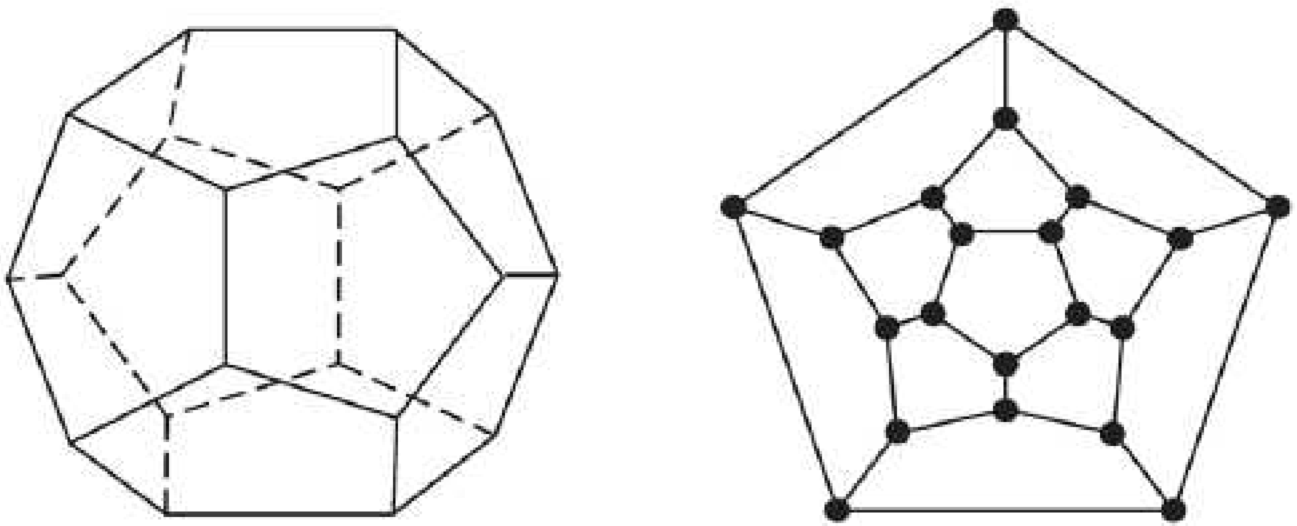
k-Postman Chinese Postman Problem (k-PCPP) $\in$ NPC

Hamilton Paths and Circuits

- Euler paths and circuits contained every edge only once. What about containing every vertex exactly once?

Hamilton Paths and Circuits

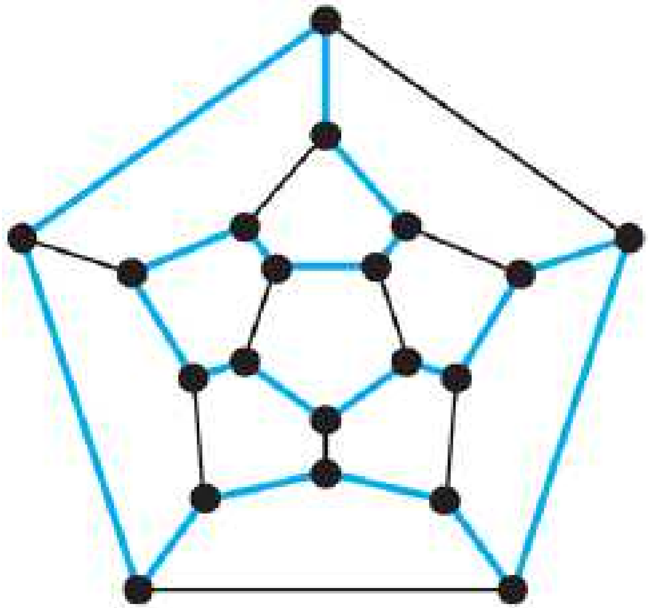
- Euler paths and circuits contained every edge only once. What about containing every vertex exactly once?



(a) (b)

Hamilton Paths and Circuits

- Euler paths and circuits contained every edge only once. What about containing every vertex exactly once?

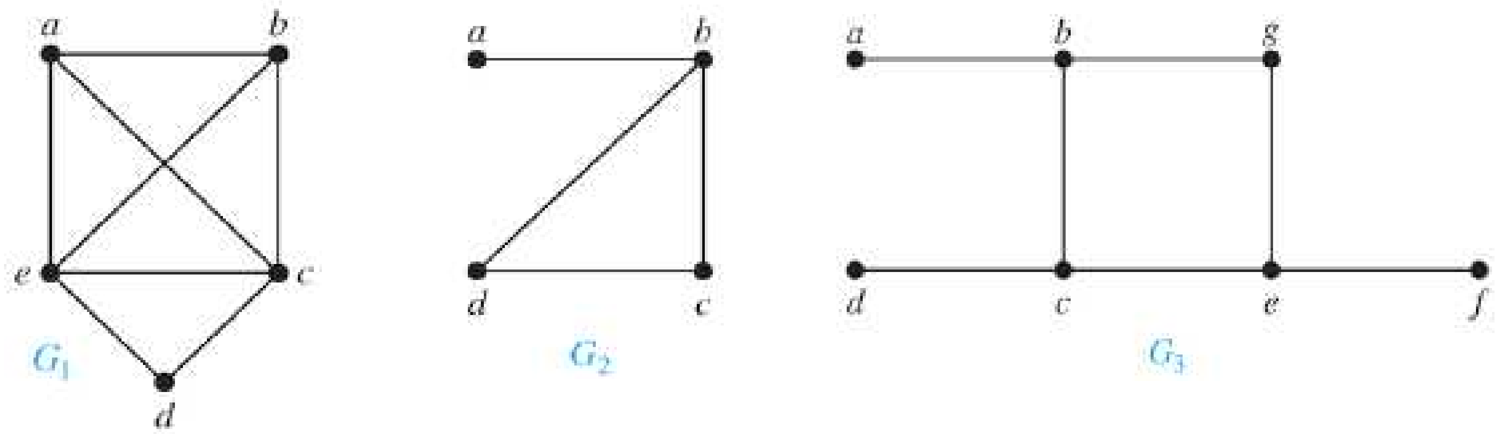


Hamilton Paths and Circuits

- **Definition**: A simple path in a graph G that passes through every vertex exactly once is called a *Hamilton path*, and a simple circuit in a graph G that passes through every vertex exactly once is called a *Hamilton circuit*.

- **Definition**: A simple path in a graph G that passes through every vertex exactly once is called a *Hamilton path*, and a simple circuit in a graph G that passes through every vertex exactly once is called a *Hamilton circuit*.

  Example Which of these simple graphs has a Hamilton circuit or, if not, a Hamilton path?



Sufficient Conditions for Hamilton Circuits

- No simple necessary and sufficient conditions are known for the existence of a Hamilton circuit.

Sufficient Conditions for Hamilton Circuits

- No simple necessary and sufficient conditions are known for the existence of a Hamilton circuit.

  But, there are some useful sufficient conditions.

Sufficient Conditions for Hamilton Circuits

- No simple necessary and sufficient conditions are known for the existence of a Hamilton circuit.

  But, there are some useful sufficient conditions.

  Dirac's Theorem If G is a simple graph with $n \geq 3$ vertices such that the degree of every vertex in G is $\geq n/2$, then G has a Hamilton circuit.

Sufficient Conditions for Hamilton Circuits

- No simple necessary and sufficient conditions are known for the existence of a Hamilton circuit.

  But, there are some useful sufficient conditions.

  Dirac's Theorem If G is a simple graph with $n \geq 3$ vertices such that the degree of every vertex in G is $\geq n/2$, then G has a Hamilton circuit.

  Ore's Theorem If G is a simple graph with $n \geq 3$ vertices such that $\deg(u) + \deg(v) \geq n$ for every pair of nonadjacent vertices, then G has a Hamilton circuit.

Sufficient Conditions for Hamilton Circuits

- No simple necessary and sufficient conditions are known for the existence of a Hamilton circuit.

  But, there are some useful sufficient conditions.

  Dirac's Theorem If G is a simple graph with $n \geq 3$ vertices such that the degree of every vertex in G is $\geq n/2$, then G has a Hamilton circuit.

  Ore's Theorem If G is a simple graph with $n \geq 3$ vertices such that $\deg(u) + \deg(v) \geq n$ for every pair of nonadjacent vertices, then G has a Hamilton circuit.

  Hamilton path problem $\in$ NPC

Applications of Hamilton Paths and Circuits

- A path or a circuit that visits each city, or each node in a communication network exactly once, can be solved by finding a Hamilton path.

Applications of Hamilton Paths and Circuits

- A path or a circuit that visits each city, or each node in a communication network exactly once, can be solved by finding a Hamilton path.

  Traveling Salesperson Problem (TSP) asks for the shortest route a traveling salesperson should take to visit a set of cities.

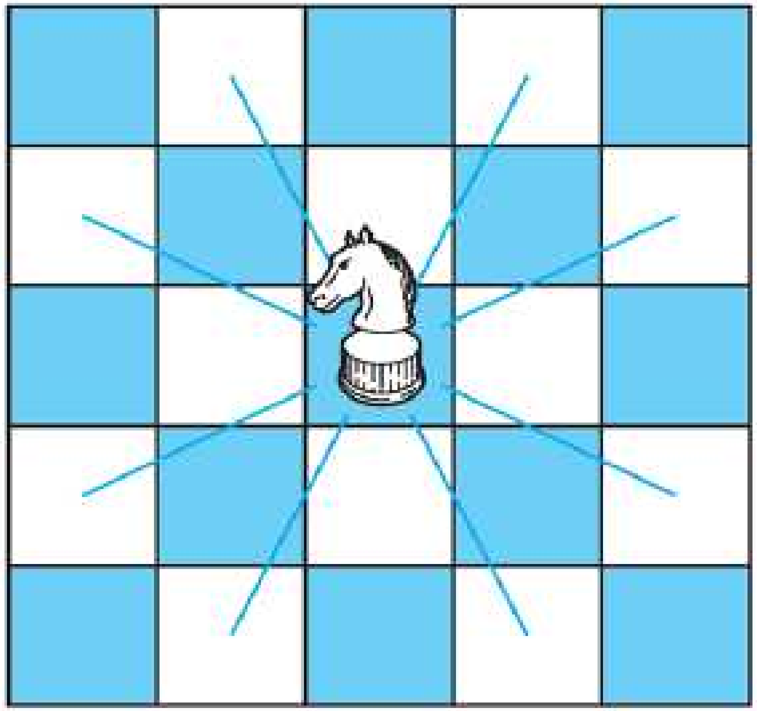
Applications of Hamilton Paths and Circuits

- A path or a circuit that visits each city, or each node in a communication network exactly once, can be solved by finding a Hamilton path.

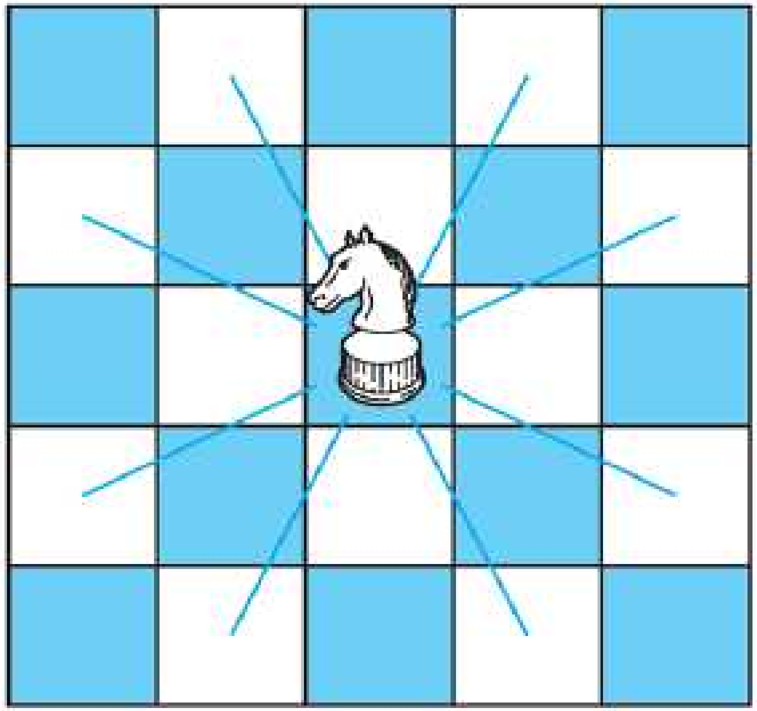
  Traveling Salesperson Problem (TSP) asks for the shortest route a traveling salesperson should take to visit a set of cities.

  the decision version of the TSP $\in$ NPC

- Can we traverse every space (and come back) in the 5×5 chessboard?



- Can we traverse every space (and come back) in the 5×5 chessboard?



What about in 6×6 chessboard?

Shortest Path Problems

- Using graphs with weights assigned to their edges

Shortest Path Problems

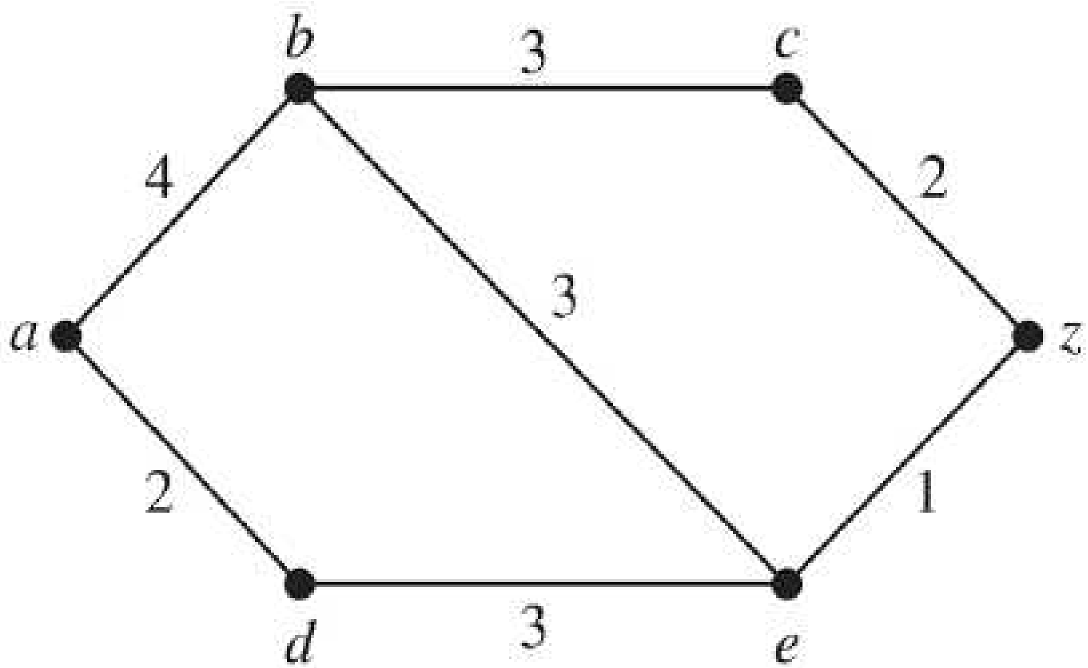
- Using graphs with weights assigned to their edges

  Such graphs are called *weighted graphs* and can model lots of questions involving distance, time consuming, fares, etc.

Shortest Path Problems

- Using graphs with weights assigned to their edges

  Such graphs are called *weighted graphs* and can model lots of questions involving distance, time consuming, fares, etc.



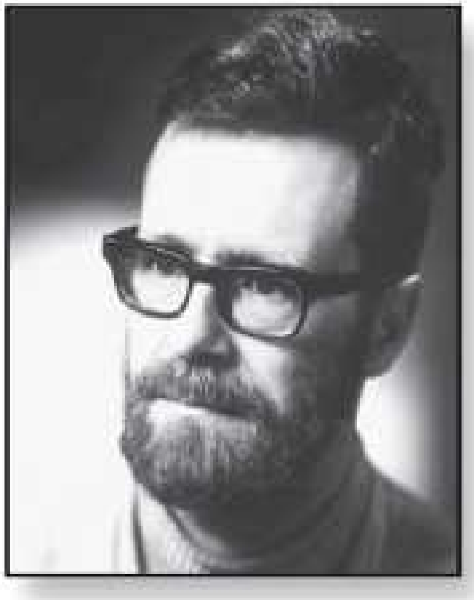
- **Definition** Let G^α be an weighted graph, with a weight function $\alpha : E \to \mathbf{R}^+$ on its edges. If $P = e_1 e_2 \cdots e_k$ is a path, then its weight is $\alpha(P) = \sum_{i=1}^{k} \alpha(e_i)$. The *minimum weighted distance* between two vertices is

$$d(u, v) = \min\{\alpha(P) | P : u \to v\}$$

Shortest Path Problems

- **Definition** Let G^α be an weighted graph, with a weight function $\alpha : E \to \mathbf{R}^+$ on its edges. If $P = e_1 e_2 \cdots e_k$ is a path, then its weight is $\alpha(P) = \sum_{i=1}^{k} \alpha(e_i)$. The *minimum weighted distance* between two vertices is

$$d(u, v) = \min\{\alpha(P) | P : u \to v\}$$



Edsger Wybe Dijkstra

Dijkstra's Algorithm

- (i) Set $d(v_0) = 0$ and $d(v) = \infty$ for all $v \neq v_0$, $S = \emptyset$

Dijkstra's Algorithm

- (i) Set $d(v_0) = 0$ and $d(v) = \infty$ for all $v \neq v_0$, $S = \emptyset$

  (ii) while $S \neq V$

  let $v \notin S$ be the vertex with the least value $d(v)$,

  $S = S \cup \{v\}$ for each $u \notin S$, replace $d(u)$ by

  $$\min\{d(u), d(v) + \alpha(u, v)\}$$

Dijkstra's Algorithm

- (i) Set $d(v_0) = 0$ and $d(v) = \infty$ for all $v \neq v_0$, $S = \emptyset$

  (ii) while $S \neq V$

  let $v \notin S$ be the vertex with the least value $d(v)$,

  $S = S \cup \{v\}$ for each $u \notin S$, replace $d(u)$ by

  $$\min\{d(u), d(v) + \alpha(u, v)\}$$

  (iii) return all $d(v)$'s

Dijkstra's Algorithm

- (i) Set $d(v_0) = 0$ and $d(v) = \infty$ for all $v \neq v_0$, $S = \emptyset$

  (ii) while $S \neq V$

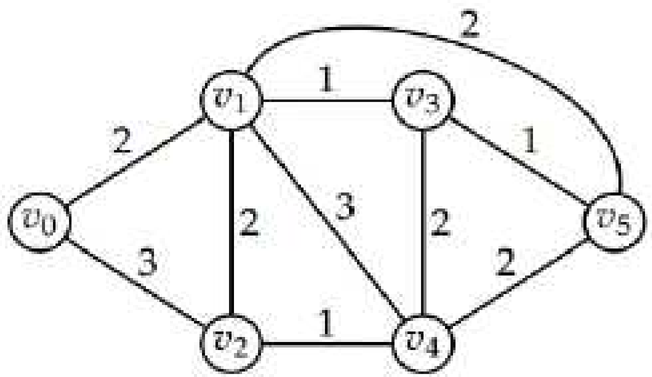
  let $v \notin S$ be the vertex with the least value $d(v)$,

  $S = S \cup \{v\}$ for each $u \notin S$, replace $d(u)$ by

  $$\min\{d(u), d(v) + \alpha(u, v)\}$$

  (iii) return all $d(v)$'s

  Example



Dijkstra's Algorithm

- (i) Set $d(v_0) = 0$ and $d(v) = \infty$ for all $v \neq v_0$, $S = \emptyset$

  (ii) while $S \neq V$

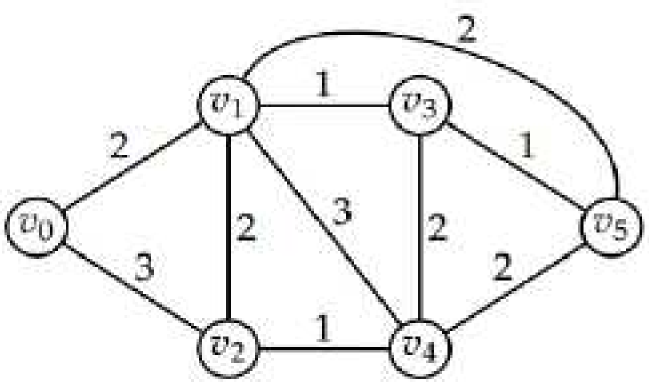
  let $v \notin S$ be the vertex with the least value $d(v)$,

  $S = S \cup \{v\}$ for each $u \notin S$, replace $d(u)$ by

  $$\min\{d(u), d(v) + \alpha(u, v)\}$$

  (iii) return all $d(v)$'s

  Example



$d(v_0) = 0$, all other $d(v) = \infty$

Dijkstra's Algorithm

- (i) Set $d(v_0) = 0$ and $d(v) = \infty$ for all $v \neq v_0$, $S = \emptyset$

  (ii) while $S \neq V$

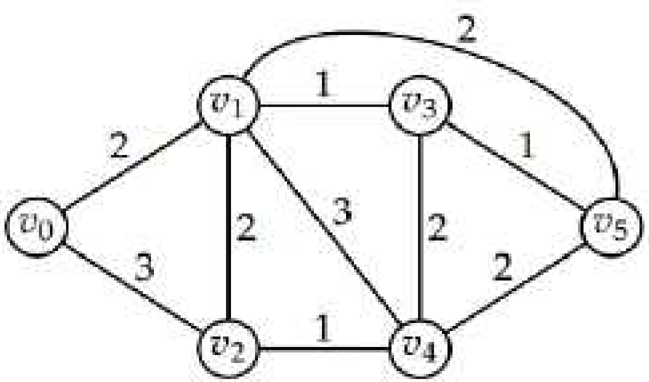
  let $v \notin S$ be the vertex with the least value $d(v)$,

  $S = S \cup \{v\}$ for each $u \notin S$, replace $d(u)$ by

  $$\min\{d(u), d(v) + \alpha(u, v)\}$$

  (iii) return all $d(v)$'s

  Example



v_0	v_1	v_2	v_3	v_4	v_5
0	∞	∞	∞	∞	∞

$d(v_0) = 0$, all other $d(v) = \infty$

- (i) Set $d(v_0) = 0$ and $d(v) = \infty$ for all $v \neq v_0$, $S = \emptyset$

  (ii) while $S \neq V$

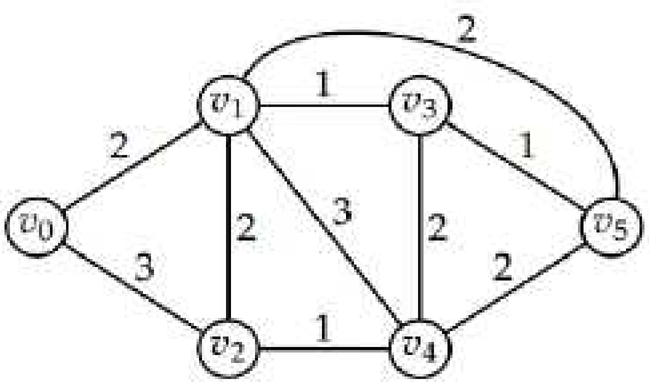
  let $v \notin S$ be the vertex with the least value $d(v)$,

  $S = S \cup \{v\}$ for each $u \notin S$, replace $d(u)$ by

  $$\min\{d(u), d(v) + \alpha(u, v)\}$$

  (iii) return all $d(v)$'s

  Example



v_0	v_1	v_2	v_3	v_4	v_5
0	∞	∞	∞	∞	∞

$i = 0$

$d(v_1) = \min\{\infty, 2\} = 2$, $d(v_2) = \min\{\infty, 3\} = 3$

Dijkstra's Algorithm

- (i) Set $d(v_0) = 0$ and $d(v) = \infty$ for all $v \neq v_0$, $S = \emptyset$

  (ii) while $S \neq V$

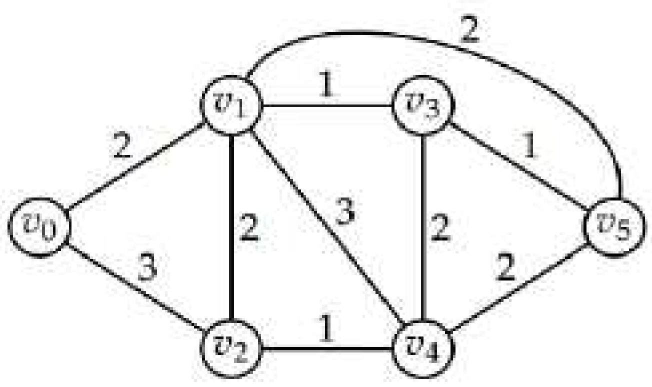
  let $v \notin S$ be the vertex with the least value $d(v)$,

  $S = S \cup \{v\}$ for each $u \notin S$, replace $d(u)$ by

  $$\min\{d(u), d(v) + \alpha(u, v)\}$$

  (iii) return all $d(v)$'s

Example



v_0	v_1	v_2	v_3	v_4	v_5
0	2	3	∞	∞	∞

$i = 0$

$d(v_1) = \min\{\infty, 2\} = 2$, $d(v_2) = \min\{\infty, 3\} = 3$

Dijkstra's Algorithm

- (i) Set $d(v_0) = 0$ and $d(v) = \infty$ for all $v \neq v_0$, $S = \emptyset$

  (ii) while $S \neq V$

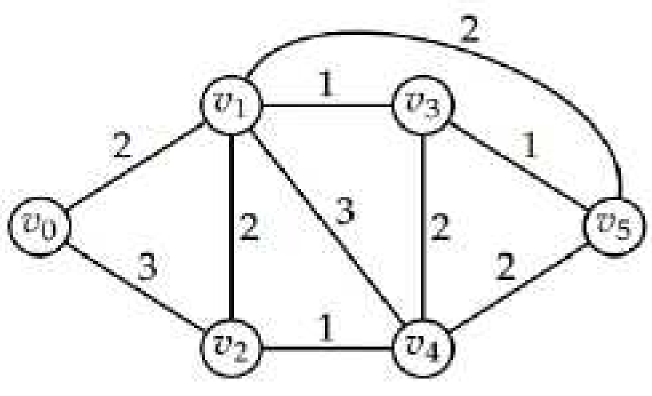
  let $v \notin S$ be the vertex with the least value $d(v)$,

  $S = S \cup \{v\}$ for each $u \notin S$, replace $d(u)$ by

  $$\min\{d(u), d(v) + \alpha(u, v)\}$$

  (iii) return all $d(v)$'s

Example



v_0	v_1	v_2	v_3	v_4	v_5
0	2	3	∞	∞	∞

$i = 1$

$d(v_2) = \min\{3, d(v_1) + \alpha(v_1 v_2)\} = \min\{3, 4\} = 3$,

$d(v_3) = 2 + 1 = 3$, $d(v_4) = 2 + 3 = 5$,

$d(v_5) = 2 + 2 = 4$

Dijkstra's Algorithm

- (i) Set $d(v_0) = 0$ and $d(v) = \infty$ for all $v \neq v_0$, $S = \emptyset$

  (ii) while $S \neq V$

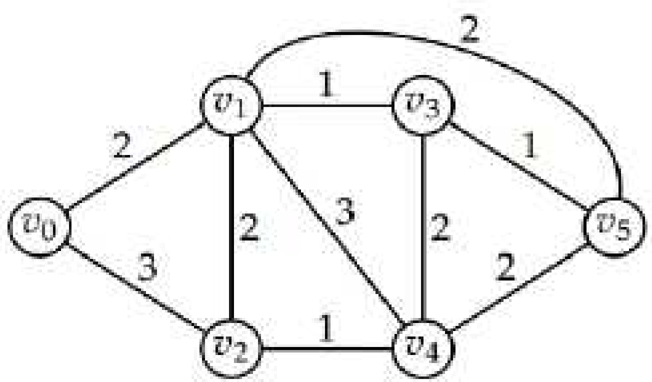
  let $v \notin S$ be the vertex with the least value $d(v)$,

  $S = S \cup \{v\}$ for each $u \notin S$, replace $d(u)$ by

  $$\min\{d(u), d(v) + \alpha(u, v)\}$$

  (iii) return all $d(v)$'s

Example



v_0	v_1	v_2	v_3	v_4	v_5
0	2	3	3	5	4

$i = 1$

$d(v_2) = \min\{3, d(v_1) + \alpha(v_1 v_2)\} = \min\{3, 4\} = 3,$

$d(v_3) = 2 + 1 = 3$, $d(v_4) = 2 + 3 = 5,$

$d(v_5) = 2 + 2 = 4$

Dijkstra's Algorithm

- (i) Set $d(v_0) = 0$ and $d(v) = \infty$ for all $v \neq v_0$, $S = \emptyset$

  (ii) while $S \neq V$

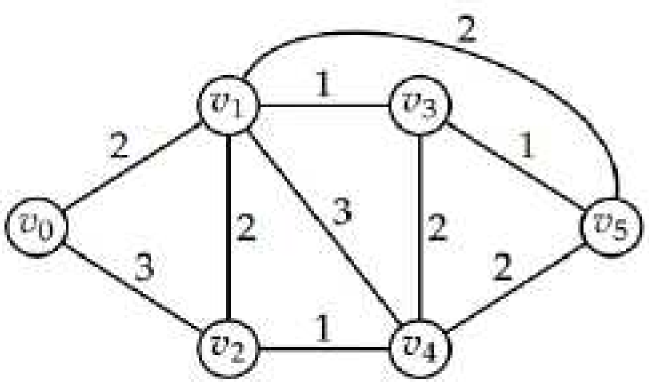
  let $v \notin S$ be the vertex with the least value $d(v)$,

  $S = S \cup \{v\}$ for each $u \notin S$, replace $d(u)$ by

  $$\min\{d(u), d(v) + \alpha(u, v)\}$$

  (iii) return all $d(v)$'s

  Example



v_0	v_1	v_2	v_3	v_4	v_5
0	2	3	3	5	4

$i = 2$

$d(v_3) = \min\{3, \infty\} = 3,$

$d(v_4) = \min\{5, 3 + 1\} = 4,$

$d(v_5) = \min\{4, \infty\} = 4$

Dijkstra's Algorithm

- (i) Set $d(v_0) = 0$ and $d(v) = \infty$ for all $v \neq v_0$, $S = \emptyset$

  (ii) while $S \neq V$

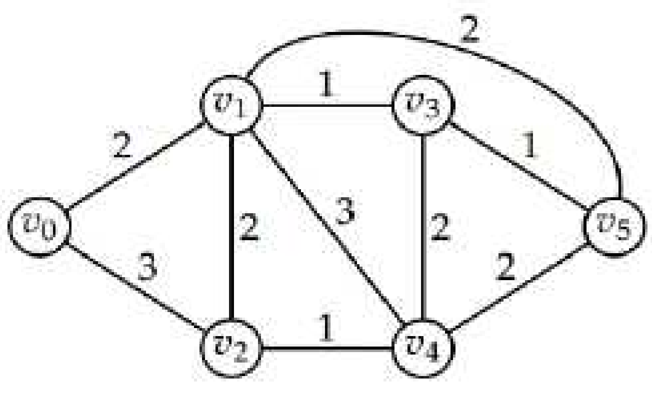
  let $v \notin S$ be the vertex with the least value $d(v)$,

  $S = S \cup \{v\}$ for each $u \notin S$, replace $d(u)$ by

  $$\min\{d(u), d(v) + \alpha(u, v)\}$$

  (iii) return all $d(v)$'s

  Example



v_0	v_1	v_2	v_3	v_4	v_5
0	2	3	3	4	4

$i = 2$

$d(v_3) = \min\{3, \infty\} = 3$,

$d(v_4) = \min\{5, 3 + 1\} = 4$,

$d(v_5) = \min\{4, \infty\} = 4$

Dijkstra's Algorithm

- (i) Set $d(v_0) = 0$ and $d(v) = \infty$ for all $v \neq v_0$, $S = \emptyset$

  (ii) while $S \neq V$

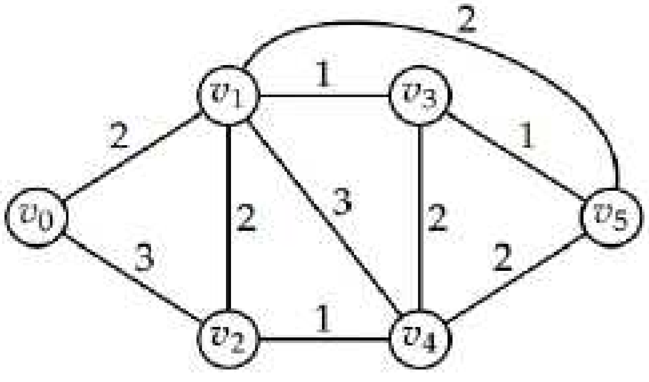
  let $v \notin S$ be the vertex with the least value $d(v)$,

  $S = S \cup \{v\}$ for each $u \notin S$, replace $d(u)$ by

  $$\min\{d(u), d(v) + \alpha(u, v)\}$$

  (iii) return all $d(v)$'s

Example



v_0	v_1	v_2	v_3	v_4	v_5
0	2	3	3	4	4

$i = 3$

$d(v_4) = \min\{4, 3+2\} = 4,$

$d(v_5) = \min\{4, 3+1\} = 4$

Dijkstra's Algorithm

- (i) Set $d(v_0) = 0$ and $d(v) = \infty$ for all $v \neq v_0$, $S = \emptyset$

  (ii) while $S \neq V$

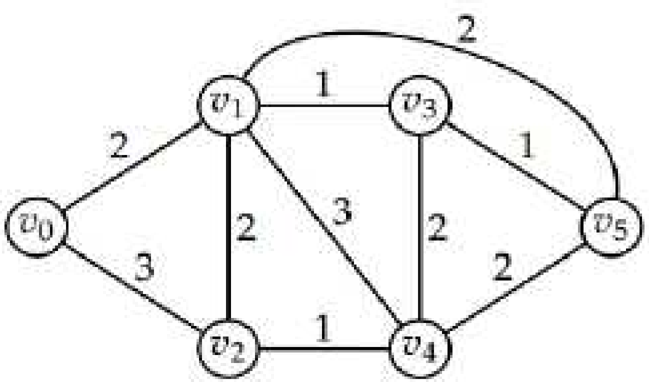
  let $v \notin S$ be the vertex with the least value $d(v)$,

  $S = S \cup \{v\}$ for each $u \notin S$, replace $d(u)$ by

  $\min\{d(u), d(v) + \alpha(u, v)\}$

  (iii) return all $d(v)$'s

Example



v_0	v_1	v_2	v_3	v_4	v_5
0	2	3	3	4	4

$i = 4$

$d(v_5) = \min\{4, 4 + 2\} = 4$

Dijkstra's Algorithm

- **Theorem** Dijkstra's algorithm finds the length of a shortest path between two vertices in a connnected simple undirected weighted graph.

Dijkstra's Algorithm

- **Theorem** Dijkstra's algorithm finds the length of a shortest path between two vertices in a connnected simple undirected weighted graph.

  Correctness

Dijkstra's Algorithm

- **Theorem** Dijkstra's algorithm finds the length of a shortest path between two vertices in a connnected simple undirected weighted graph.

  Correctness

  Theorem Dijkstra's algorithm uses $O(n^2)$ operations (additions and comparisons) in a connected simple undirected weighted graph with n vertices.

Dijkstra's Algorithm

- **Theorem** Dijkstra's algorithm finds the length of a shortest path between two vertices in a connnected simple undirected weighted graph.

  Correctness

  Theorem Dijkstra's algorithm uses $O(n^2)$ operations (additions and comparisons) in a connected simple undirected weighted graph with n vertices.

  Complexity

Dijkstra's Algorithm

- **Theorem** Dijkstra's algorithm finds the length of a shortest path between two vertices in a connnected simple undirected weighted graph.

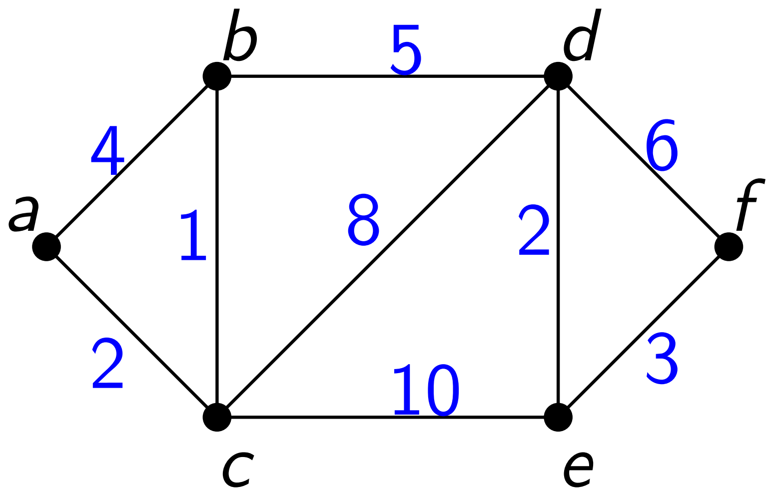
  Correctness

  Theorem Dijkstra's algorithm uses $O(n^2)$ operations (additions and comparisons) in a connected simple undirected weighted graph with n vertices.

  Complexity

  read the Textbook p.712 – p.714

Another Example



Next Lecture

- Graph theory III ...