Complex Networks:

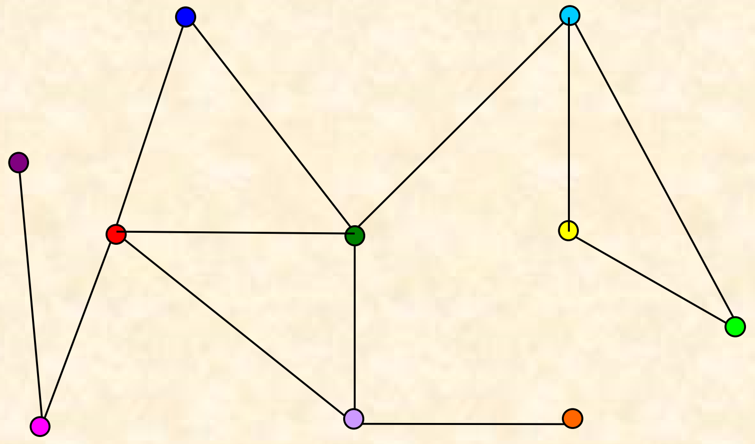
Lecture 3b: Introduction to Graph Theory

EE 6605

Instructor: G Ron Chen

Review

- **Walk**: A finite sequence of edges, one after another, in the form of $v_1v_2, v_2v_3, \ldots, v_{n-1}v_n$ where $N(G) = \{v_1, v_2, \ldots, v_n\}$ are nodes.
- A walk is denoted by $v_1 \to v_2 \to \cdots \to v_n$ and the number of edges in a walk is called its **length**.
- **Trail**: A walk in which all edges are distinct.
- **Path**: A trail in which all nodes are distinct, except perhaps $v_1 = v_n$ which is called a **closed path**, often called a **circuit** (or a **cycle, a loop**).
- **Tree**: A graph with no circuits.

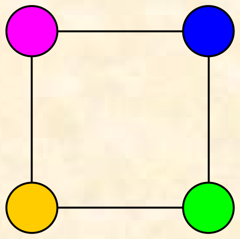


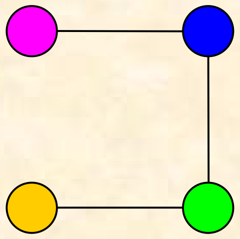
Eulerian Graphs

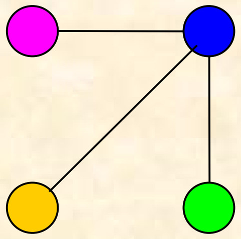
- **Eulerian graph:** If it has a closed trail that traverses every edge once and once only

- There may be more than one such trail, each of which is called an **Eulerian trail** (or, **Euler Circuit**)

- **Semi-Eulerian graph:** If it has a trail, need not be closed, that traverses each edge once and once only

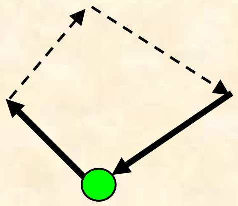




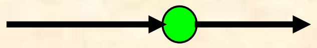


Main Results

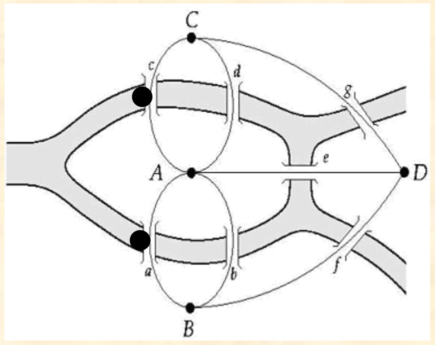
- **Theorem:** *A connected graph is Eulerian if and only if the every node in the graph has an even degree.*
- *Proof:* If a note is a starting point, then the trail has to return to it:



If a note has a trail enters it, then the trail has to leave it:

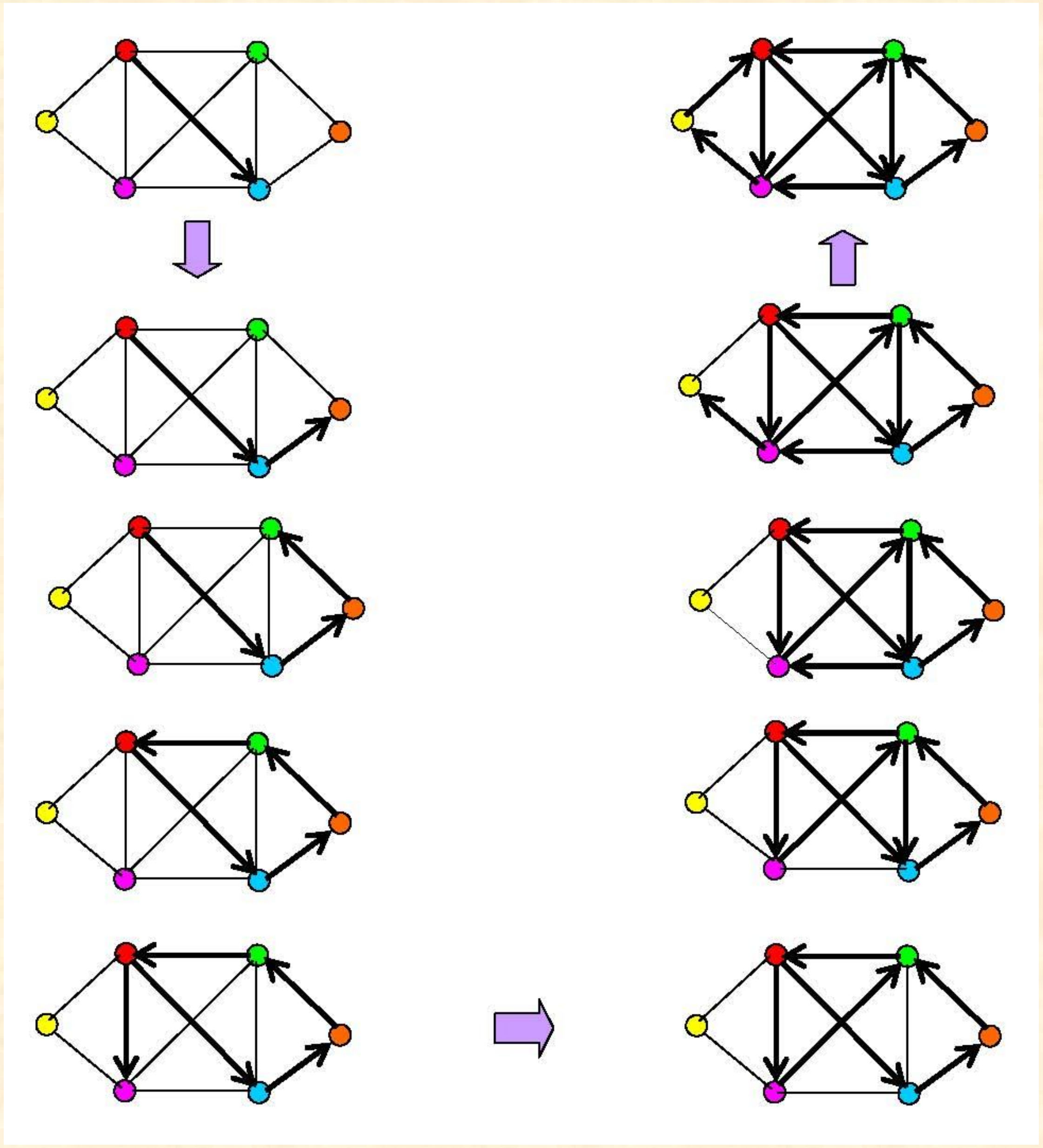


- **Corollary:** *The seven-bridge problem of Königsburg has* ***no*** *solutions.*



- *Proof:* It is not Eulerian.

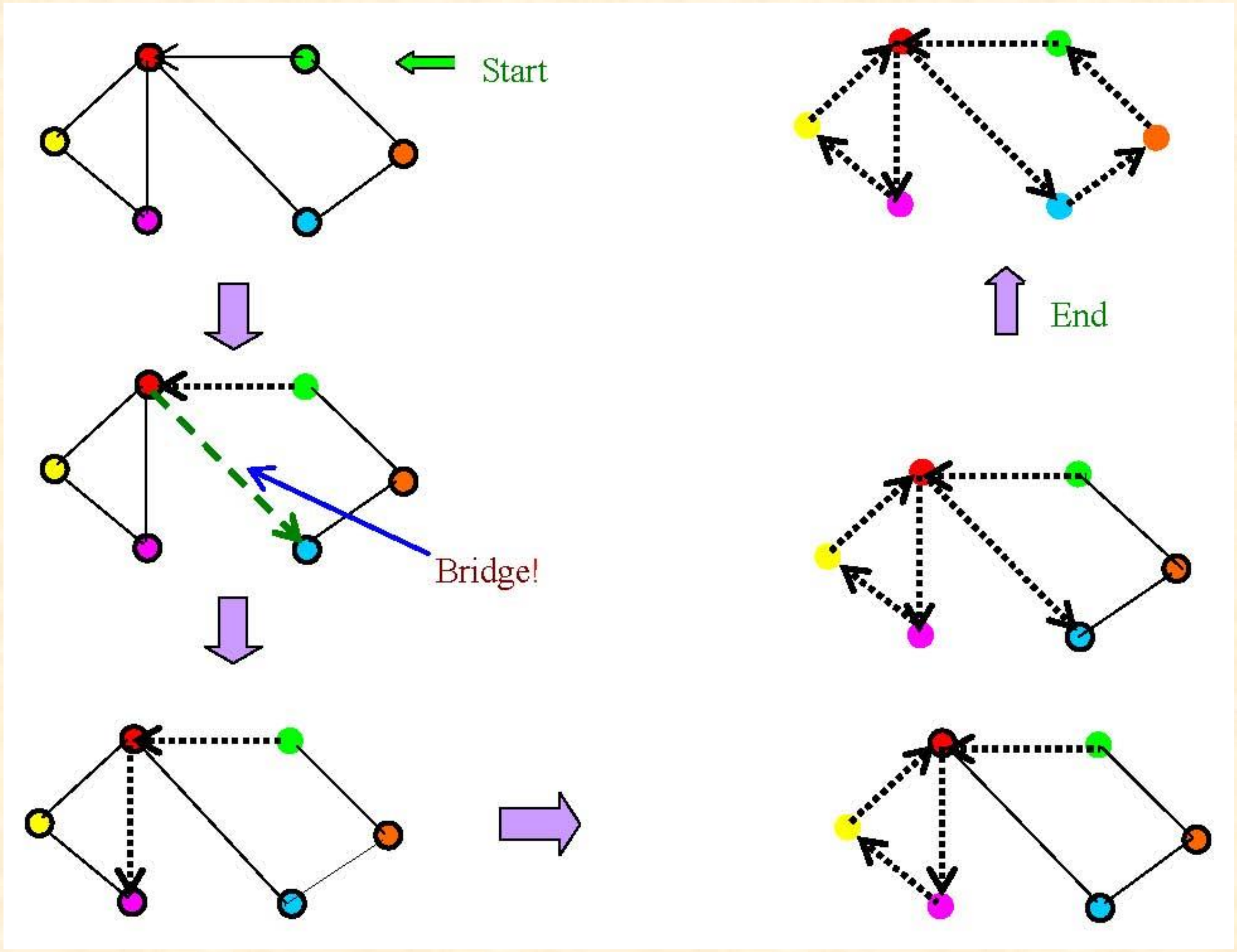
Example:



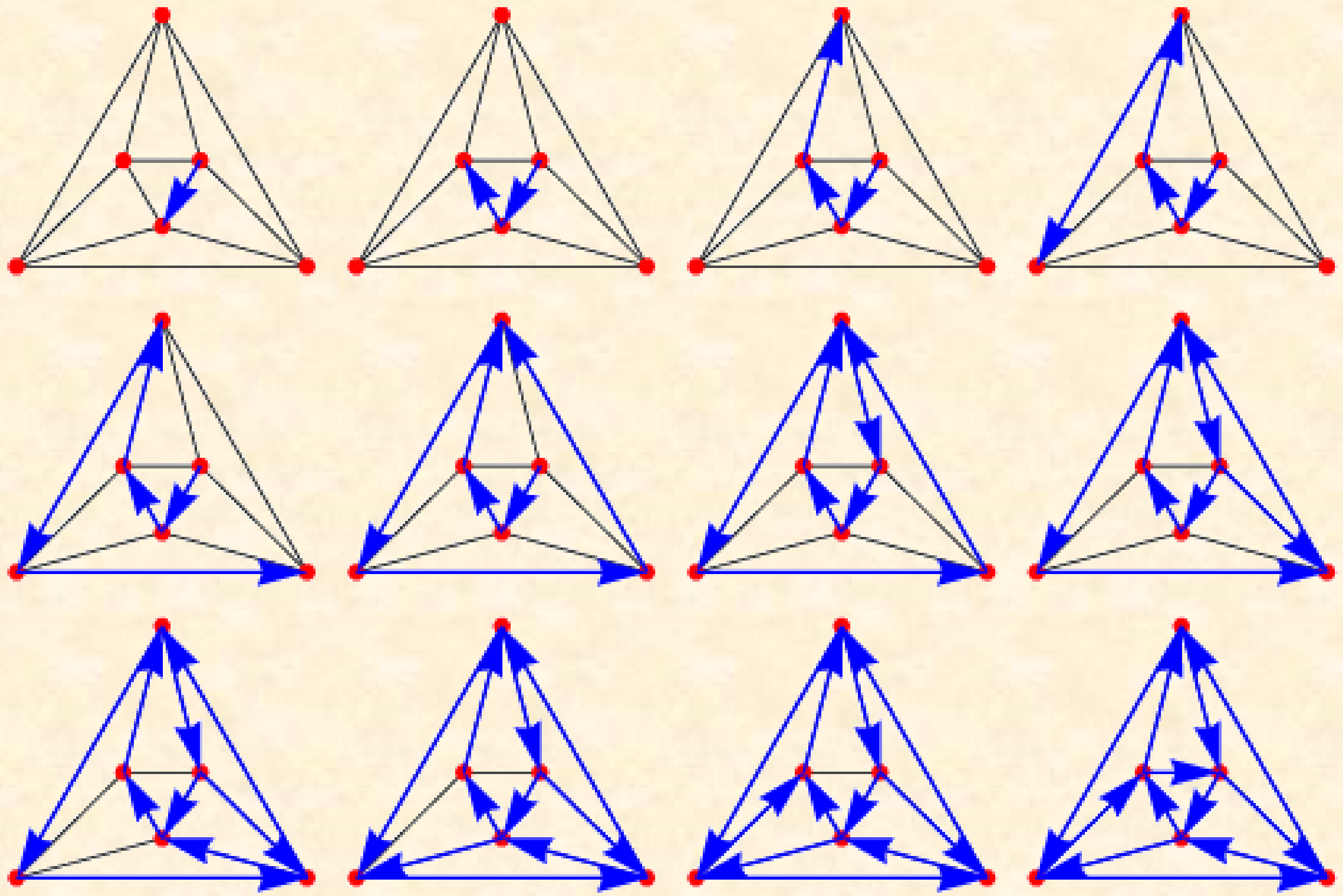
Algorithm to find Eulerian Graphs

- **Corollary** (Fleury Algorithm) *In any given Eulerian graph G, an Eulerian trail can be found by the following procedure***:**
- *Start from any node in G. Walk along the edges of G in an arbitrary manner, subject to the following rules***:**
  - *erase the edges as they are traversed;*
  - *erase the resulting isolated nodes;*
  - *walk through a bridge only if there are no other alternatives.*

Example:

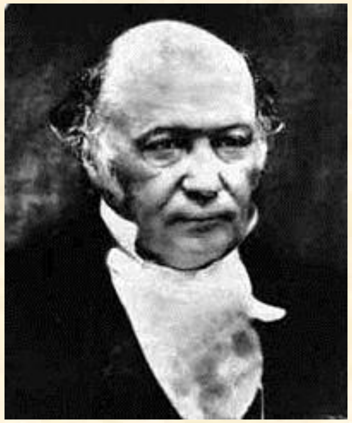


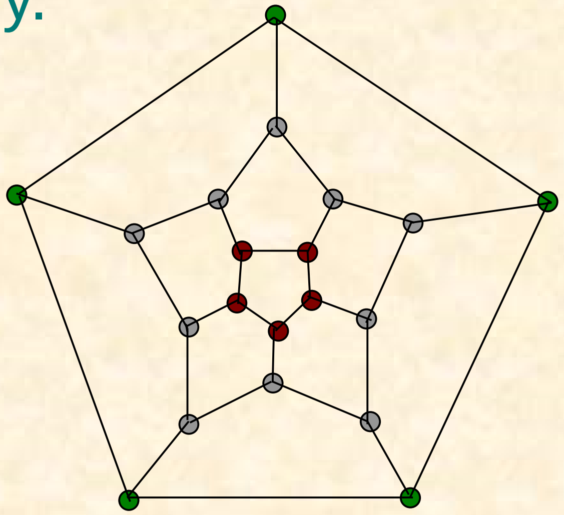
Example
(path usually not unique)



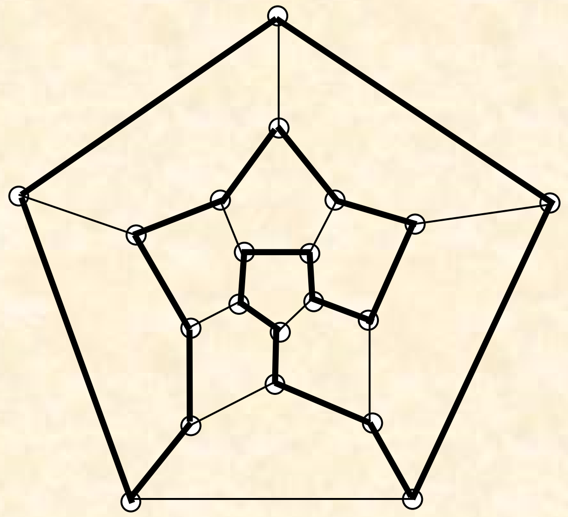
Sir William Rowan Hamilton (1805-1865)

In 1856, the English mathematician William R. Hamilton studied the world navigation problem and considered a map with 20 nodes representing cities connected by sailing routes, as depicted by the following figure. He wanted to find out if one can traverse through every city once and once only, and finally return to the starting city. His study eventually led to the establishment of the now-famous Hamiltonian graph theory.



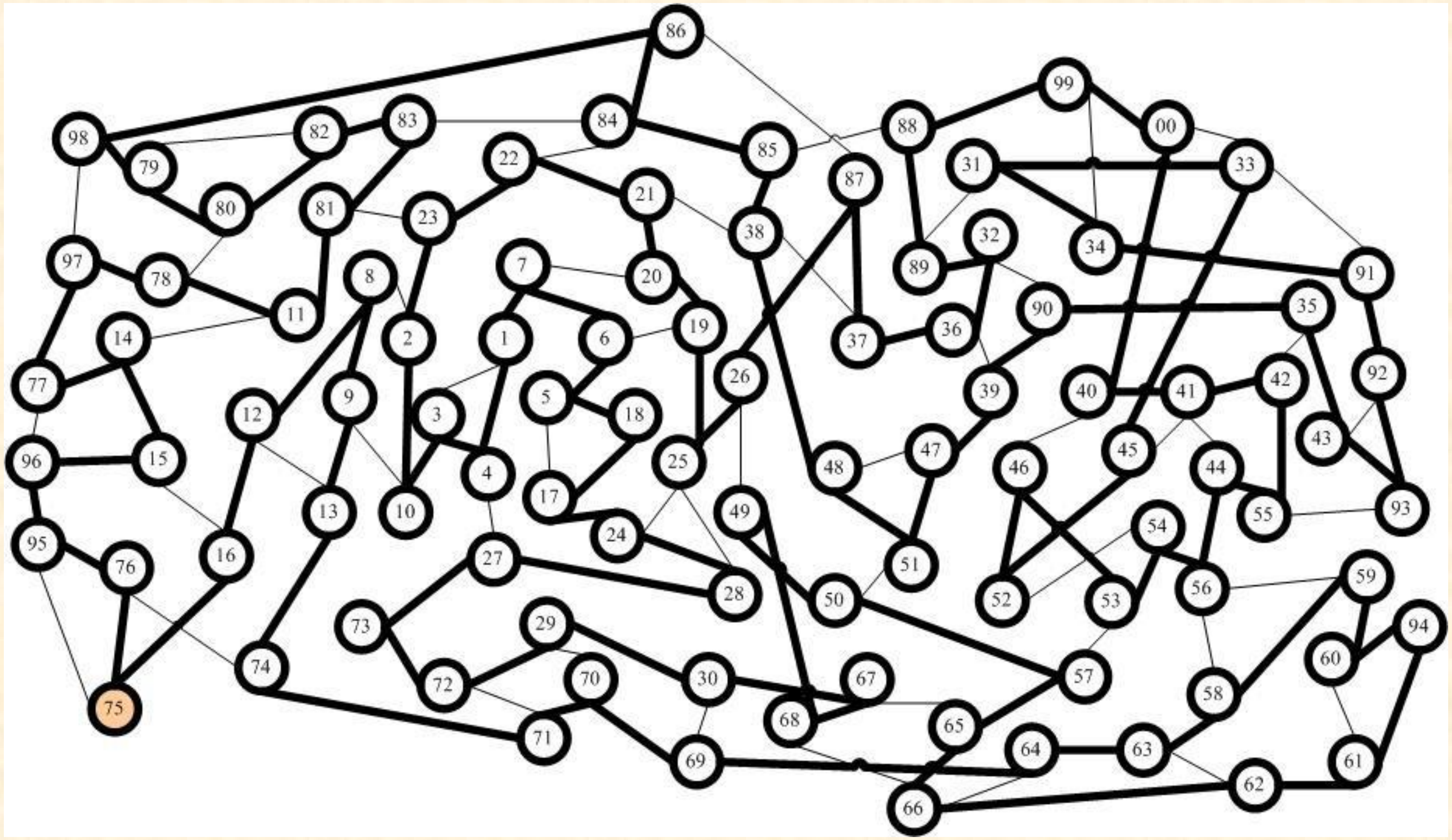


One solution



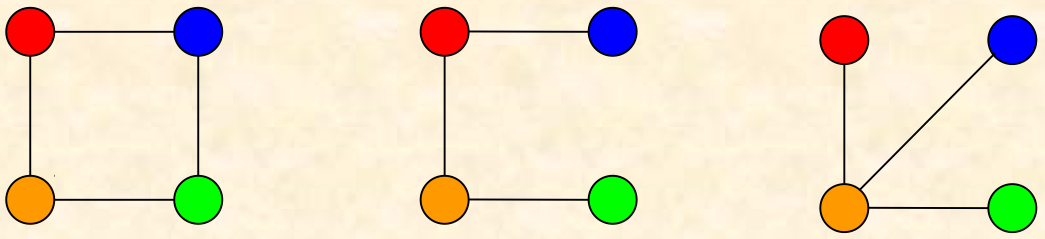
Solution not unique

Another Example



Hamiltonian Graphs

- **Hamiltonian graph:** If it has a closed trail that traverses each node once and once only
- A trivial case is a single node and a nontrivial Hamiltonian graph must be a circuit, called a **Hamiltonian circuit**
- **Semi-Hamiltonian graph:** If it has a trail, need not be closed, that traverses each node once and once only



Some Results

❖ **Theorem:** *Let G be a simple graph with $N\ (\geq 3)$ nodes. If, for every pair of non-adjacent nodes v and u, their degrees always satisfy $k(v)+k(u)\geq N$, then G is Hamiltonian.*

Proof. For a simple case with $N=3$, consider any path and any 3 adjacent nodes in it, for example:

$$v_1 - v_2 - v_3$$

Since $k(v_1)+k(v_3)\geq 3$, either v_1 or v_3 has another neighbor v_i, for example,

$$\begin{tikzcd}
 & & v_i \arrow[d, dash] \\
v_1 \arrow[r, dash] & v_2 \arrow[r, dash] & v_3
\end{tikzcd}$$

But then $k(v_1)+k(v_i)\geq 3$, so either v_1 or v_i has another neighbor, ……
Thus, the only way to satisfy $k(v_1)+k(v_i)\geq 3$ for non-adjacent v_1 and v_i is to close the path:

$$\begin{tikzcd}
 & -\ldots & v_i \arrow[d, dash] \\
v_1 \arrow[u, dash] \arrow[r, dash] & v_2 \arrow[r, dash] & v_3
\end{tikzcd}$$

This gives a Hamiltonian circuit. Since this is true for "every pair", the graph is a Hamiltonian graph.

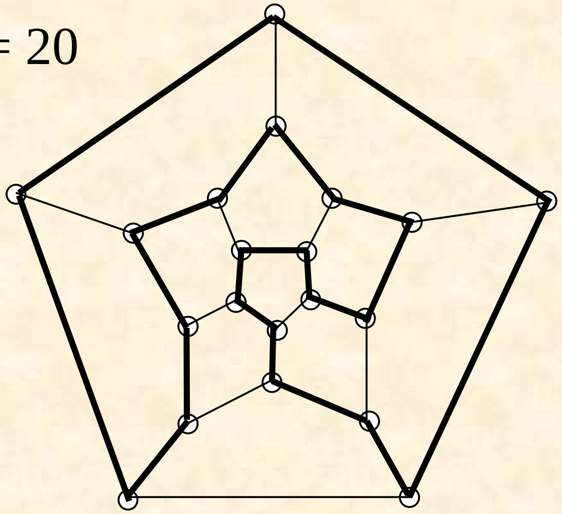
For another proof, see Wiki

Corollary: *Let G be a simple graph with $N(\geq 3)$ nodes.*
If every pair of nodes has degree $k(v)+k(u)\geq N$, then G is Hamiltonian. [Note: "every pair" contains "every non-adjacent pair"]

Corollary *Let G be a simple graph with $N(\geq 3)$ nodes. If every node has degree $k\geq N/2$, then G is Hamiltonian.*

The above conditions are sufficient but not necessary

For example: $N = 20$



$$k(v)+k(u)<N$$

$$k(v)<N/2$$

* There is no "if and only if" condition for Hamiltonian graphs so far

Travelling Salesman Problem

Travelling Salesman Problem -

Given a set of cities and the distances between each pair of cities, what is the shortest possible route that visits every city once and once only, and then returns to the starting city?

It is called an optimal Hamiltonian graph problem, which is NP-hard

A special case: The given cities are fully connected

Fully-connected graphs are called complete graphs

Travelling Salesman Problem: On complete graphs

Nearest-Neighbor Algorithm: To find a near-optimal solution

Start with an arbitrary node, v_0. Find one adjacent edge with the smallest weight, which connects to node v_1, so as to have
$v_0 - v_1$

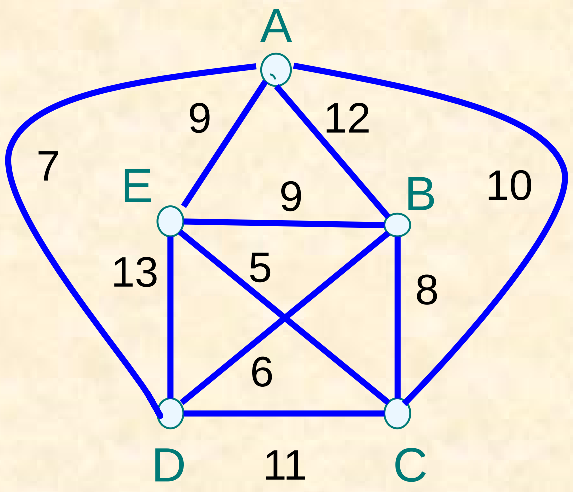
Start with v_1, repeat the same process, … until having
$v_0 - v_1 - \ldots - v_n$

If v_n still has nearest neighbor(s) not traversed, then continue;
If v_n does not have anymore nearest neighbor not traversed,
then connect v_n back to v_0

Solution: $v_0 - v_1 - \ldots - v_n - v_0$

Travelling Salesman Problem: A complete graph

Example:



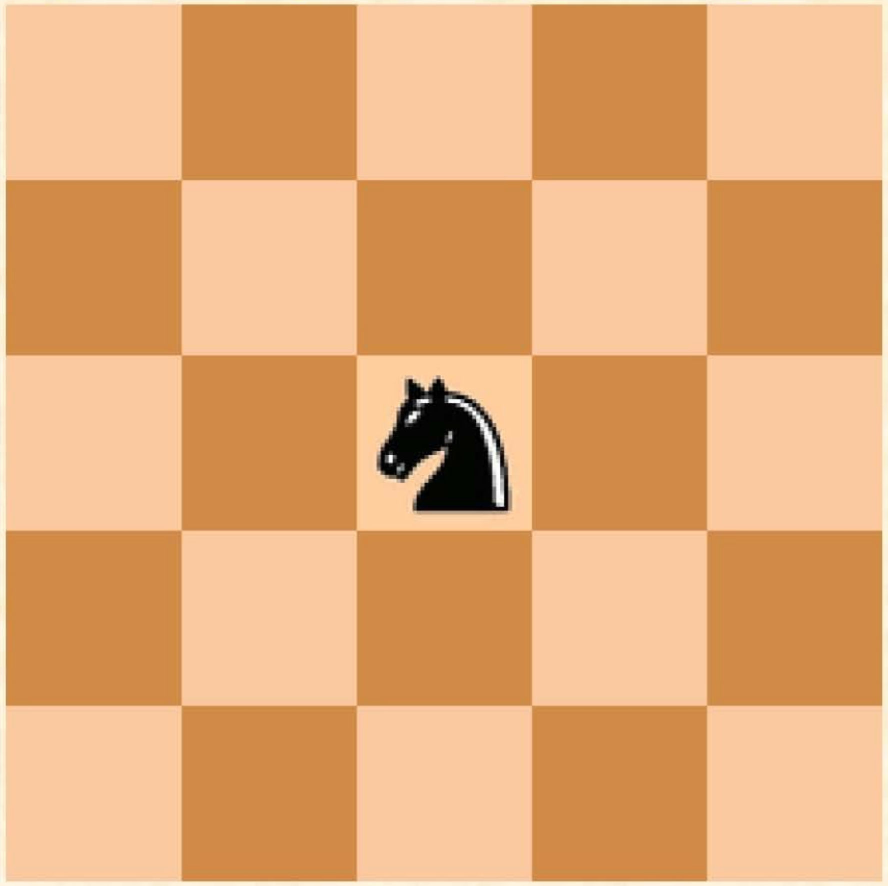
Start from A

A – D – B – C – E – A

Total path length = 7 + 6 + 8 + 5 + 9 = 35

Knight's Tour Game

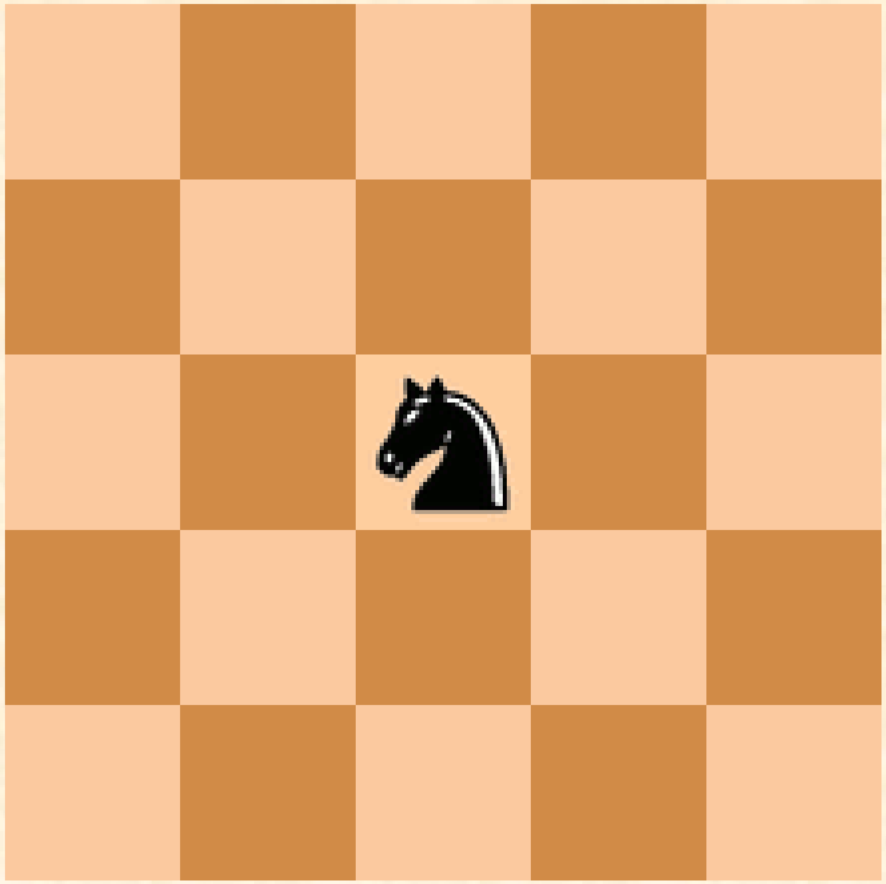
Can the knight visit every block, once and once only, and finally return to the starting point?



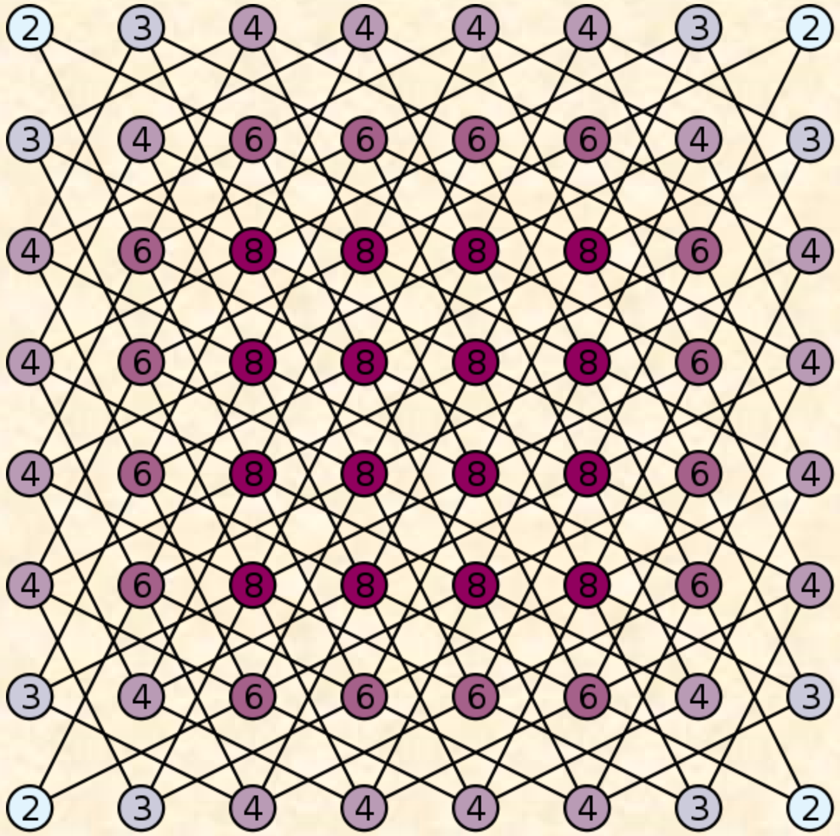
= Is this a Hamiltonian graph ?

Knight's Tour Game

Can the knight visit every block, once and once only, and finally return to the starting point?



5 X 5 has open solution (semi-Hamiltonian) but no closed solution (Hamiltonian)



8 X 8 – Yes, it always has a closed solution

Namely, it is Hamiltonian (proved by Euler in 1759)

There are 26,534,728,821,064 Hamiltonian paths

Closed Knight's Tour

By Dan Thomasson, August 17, 2001

DTHOMASSON@carolina.rr.com

8 X 8 – Yes, it is Hamiltonian

This shows one solution

Theorem (Cull and de Curtins,1978) On any rectangular board whose smaller dimension is at least 5, there exists a (possibly open) knight's tour (= semi-Hamiltonian)

Theorem (Schwenk, 1991) For any $m \times n$ board with $m \leq n$ a closed knight's tour is always possible (= Hamiltonian), except for the following few cases:

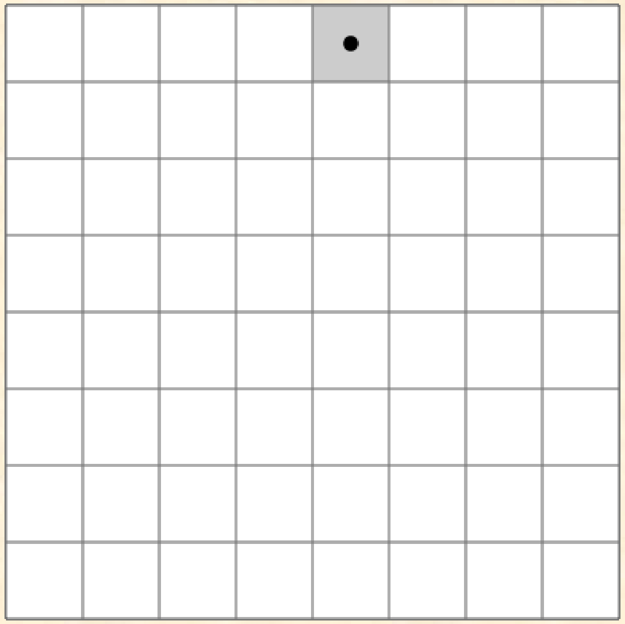
(1) m and n are both odd

(2) m = 1, 2, or 4

(3) m = 3 and n = 4, 6, or 8

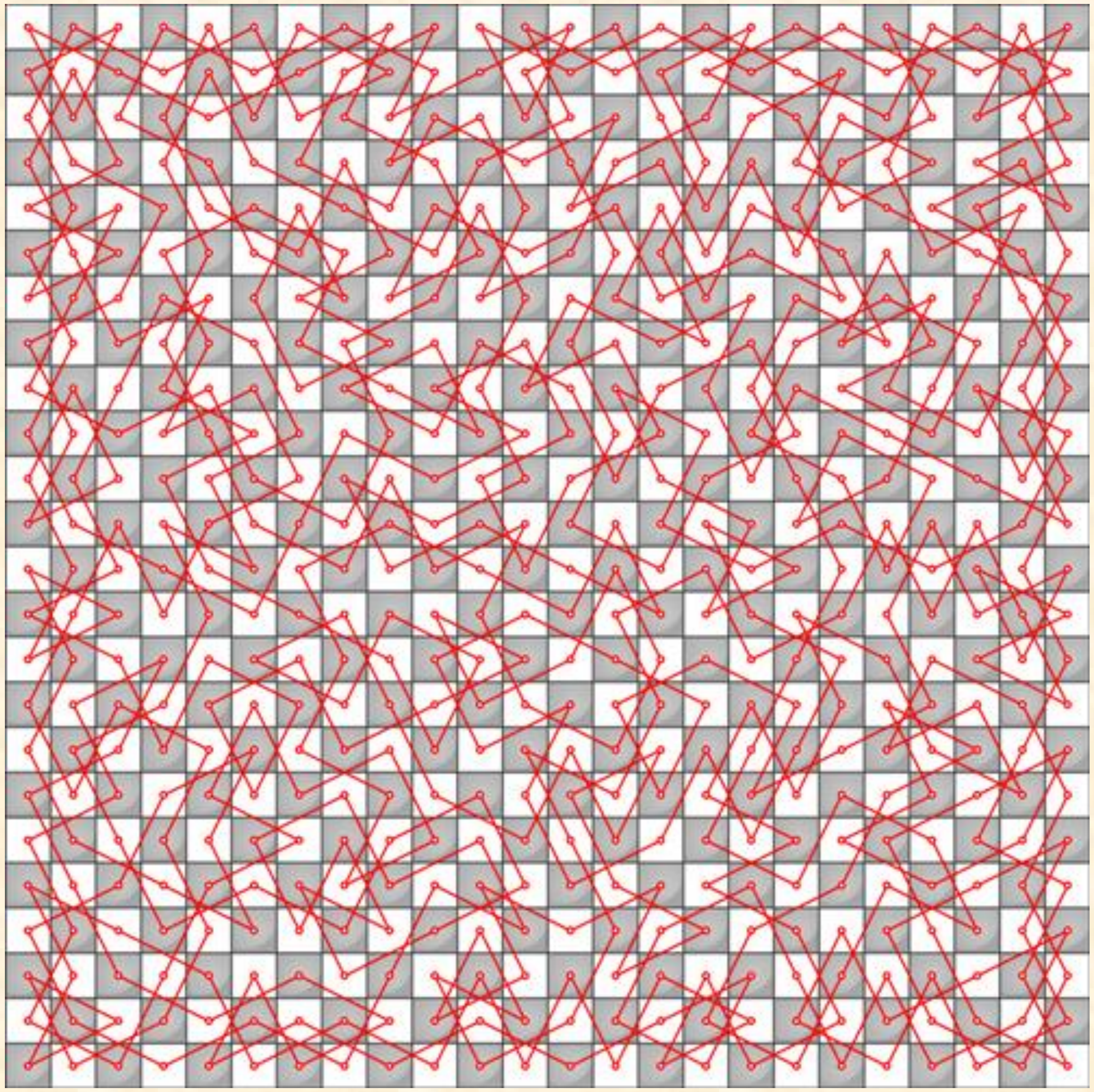
Note:
There is always a solution, which does not mean that starting anywhere will be a solution (Reason: 8X8 is a square, but the walking step of a knight is not a square)

Example



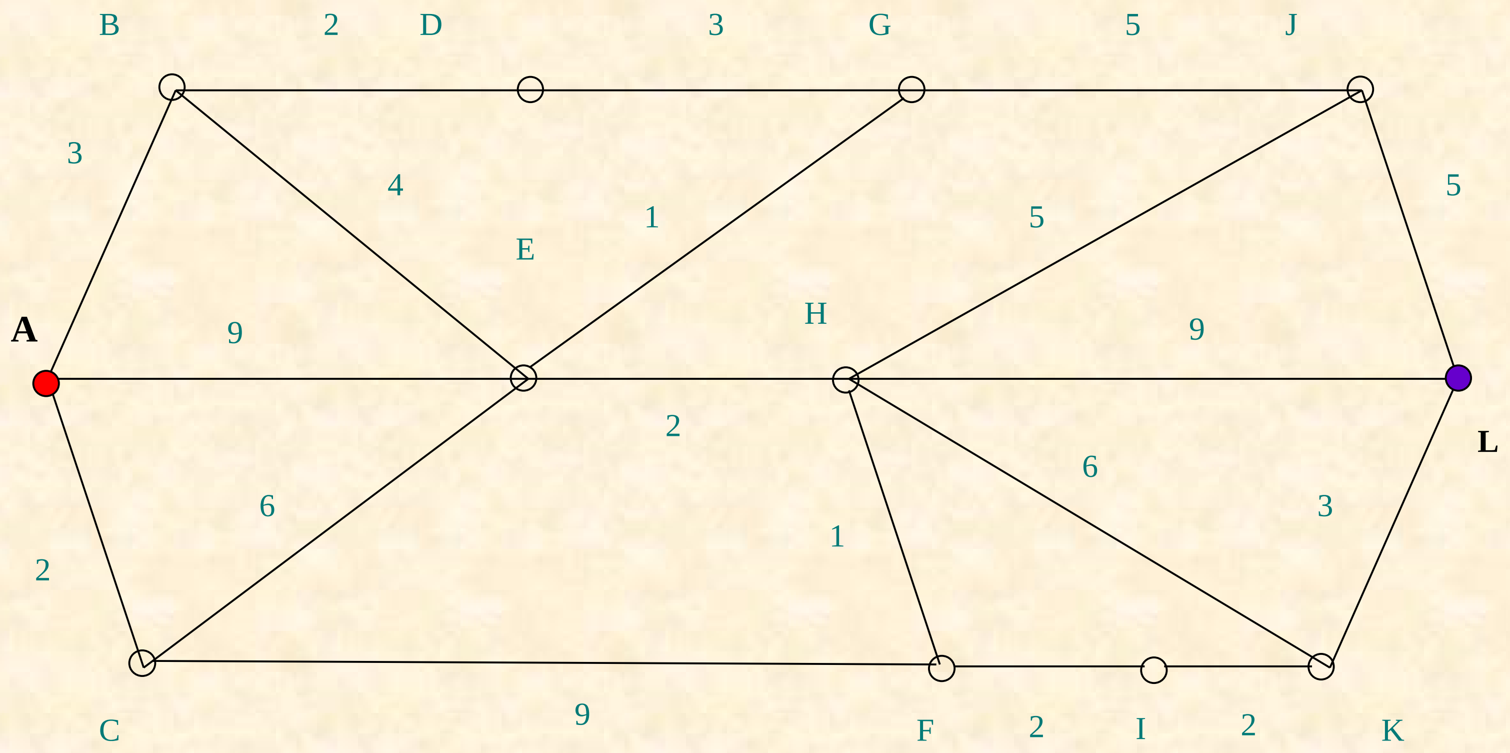
Wiki

24 x 24



Shortest Path Length Problem

Q: What is the shortest path length from A to L ?



Solving the Shortest Path Length Problem

Dijkstra’s Algorithm

Dijkstra's algorithm is a greedy algorithm (which needs to explore all nodes and all edges).

Dijkstra's algorithm is adopted by edge-state routing protocols, OSPF (Open Shortest Path First), IS-IS (Intermediate System to Intermediate System), etc.

Dijkstra's algorithm cannot handle negative edge weights (for which Bellman–Ford algorithm works)

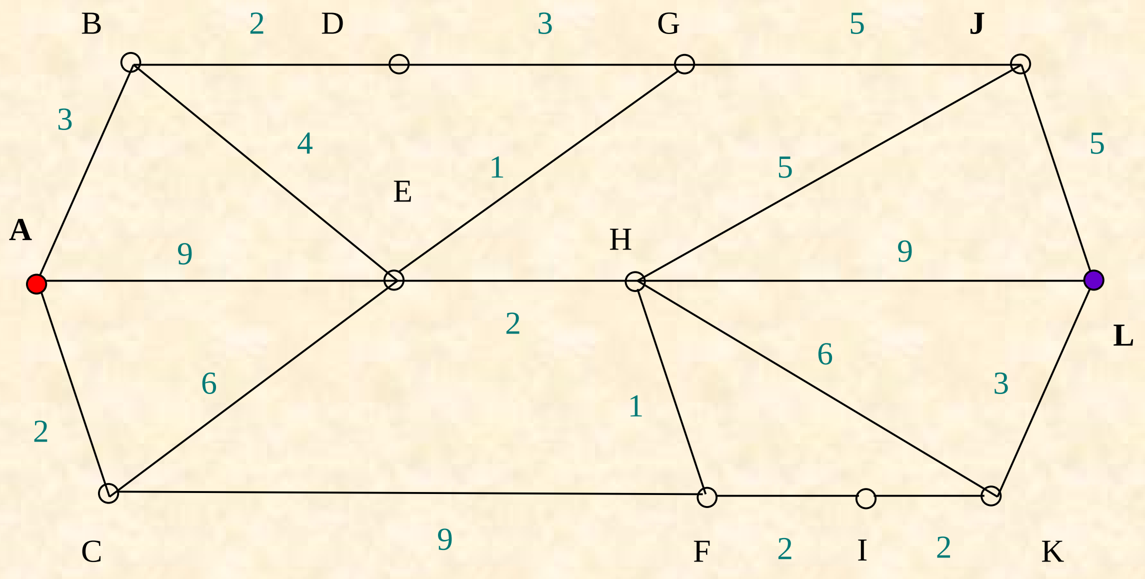
Edsger Wybe Dijkstra
(1930-2002)

Solving the Shortest Path Length Problem

- Dijkstra's Algorithm: Moving from A toward L and associating each intermediate node V with a number $l(V)$ that is equal to the shortest path length from A to V.
- Example: From A to J, one has $l(J) = l(G) + 5$ or

  $l(J) = l(H) + 5$

  whichever is shorter.

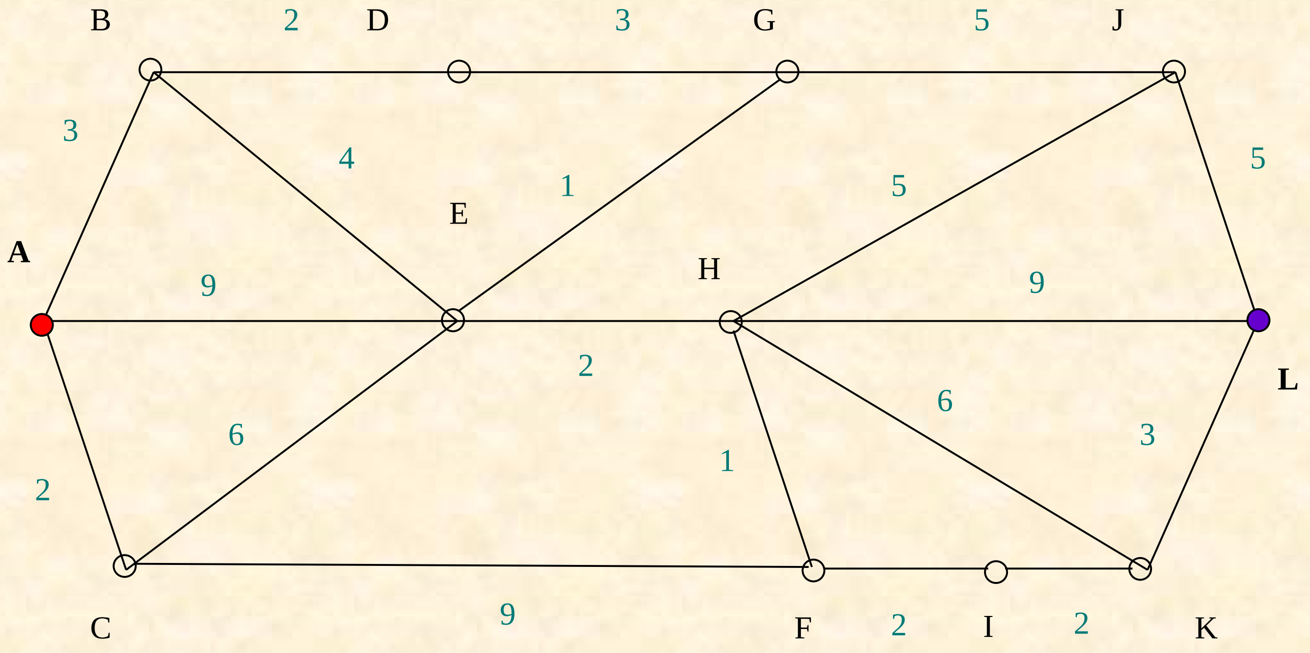


Solving the Shortest Path Length Problem

- Starting from A with $l(A)=0$, moving from A toward L step by step, one has

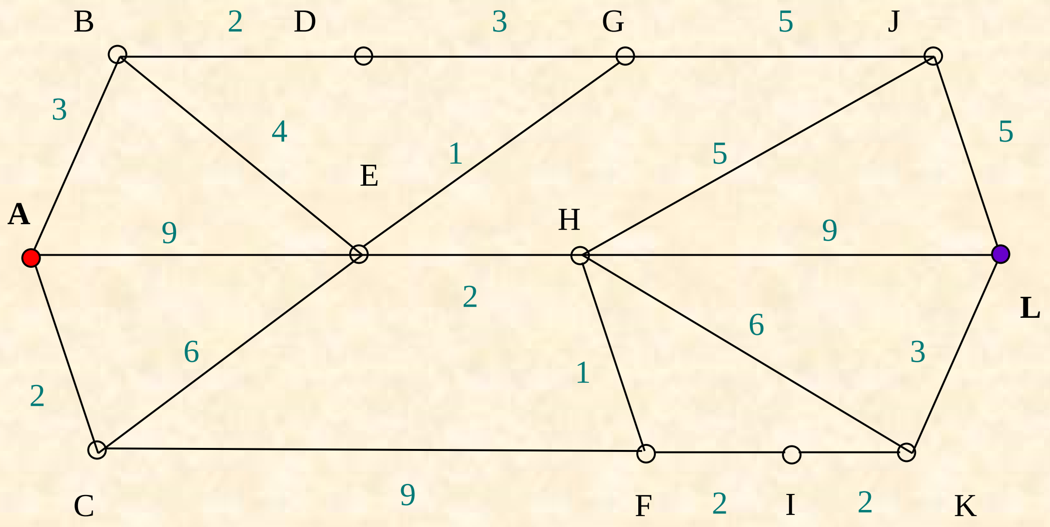
  $l(B)=l(A)+3=3$ $\quad l(E)=l(A)+9=9$ $\quad l(C)=l(A)+2=2$

- Therefore, node B is chosen, with $l(B)=3$, node C is chosen, with $l(C)=2$ and node E is chosen with $l(E)=9$



Solving the Shortest Path Length Problem

- Looking at the nodes adjacent to B, one has
  $l(D) = l(B) + 2 = 5 \quad l(E) = l(B) + 4 = 7$
- Looking at the nodes adjacent to C, one has
  $l(E) = l(C) + 6 = 8 \quad l(F) = l(C) + 9 = 11$
- So, $l(D) = 5$ is uniquely determined.
- But $l(E) = 7$, or 8 or 9. Hence, node E is chosen with $l(E) = l(B) + 4 = 7$
  so that $l(H) = l(E) + 2 = 9$
- F is chosen with $l(F) = 11$
- But also $l(F) = l(H) + 1$ so F is chosen with $l(F) = l(H) + 1 = 10$



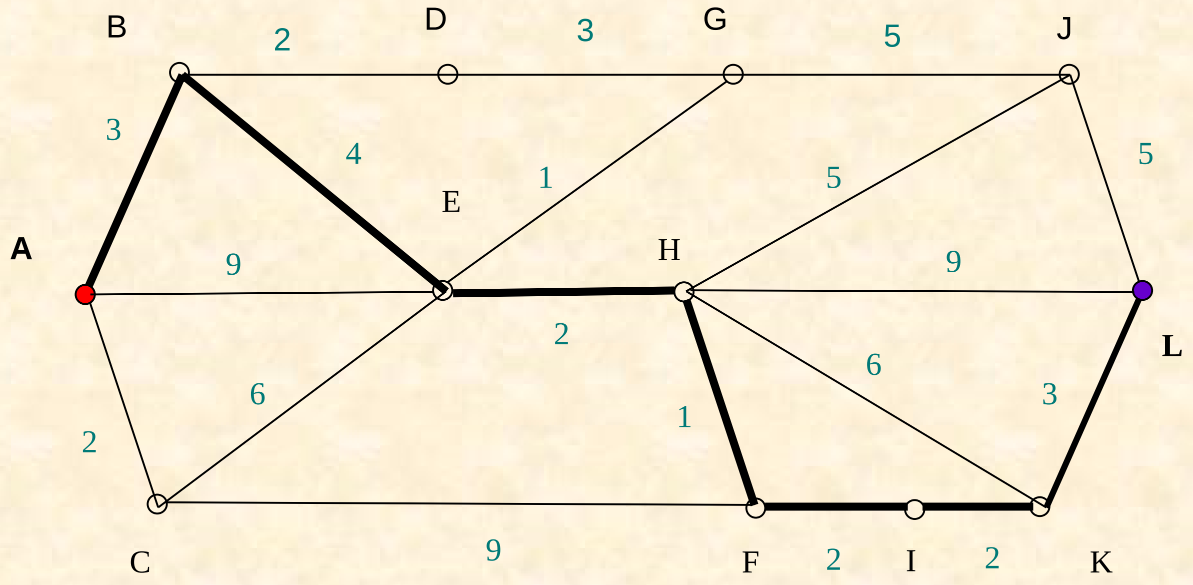
Solving the Shortest Path Length Problem

- Continuing this way, one finally obtains

$l(A)=0 \quad l(B)=3 \quad l(C)=2 \quad l(D)=5 \quad l(E)=7 \quad l(F)=10$

$l(G)=8 \quad l(H)=9 \quad l(I)=12 \quad l(J)=13 \quad l(K)=14 \quad l(L)=17$

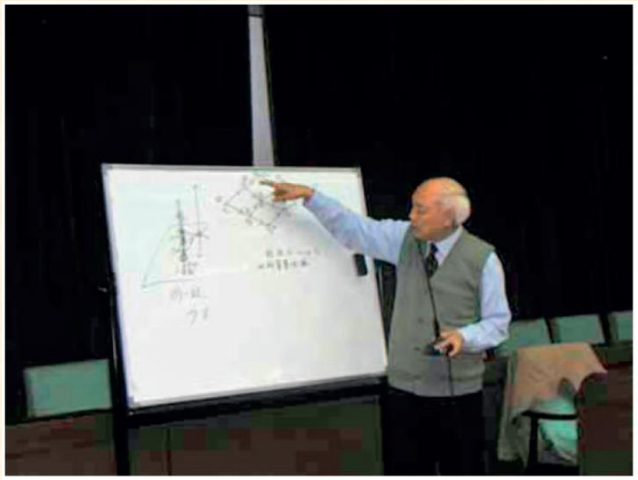
- Result: total shortest path length $=17$ from A to L :



- The above provides all shortest lengths from A to any other nodes
- Solutions may not be unique, but all will be shortest

Chinese Postman Problem

- The problem is for a postman to deliver letters in such a way that he passes every street at least once and finally returns to the starting point (the post office), traversing a shortest possible total path-length.
- First solution given by Prof Mei-ko Kwan **(1934—)** (管梅谷 from Shanghai)



Chinese Postman Problem

Ideas:

To have a solution, the graph must be Eulerian

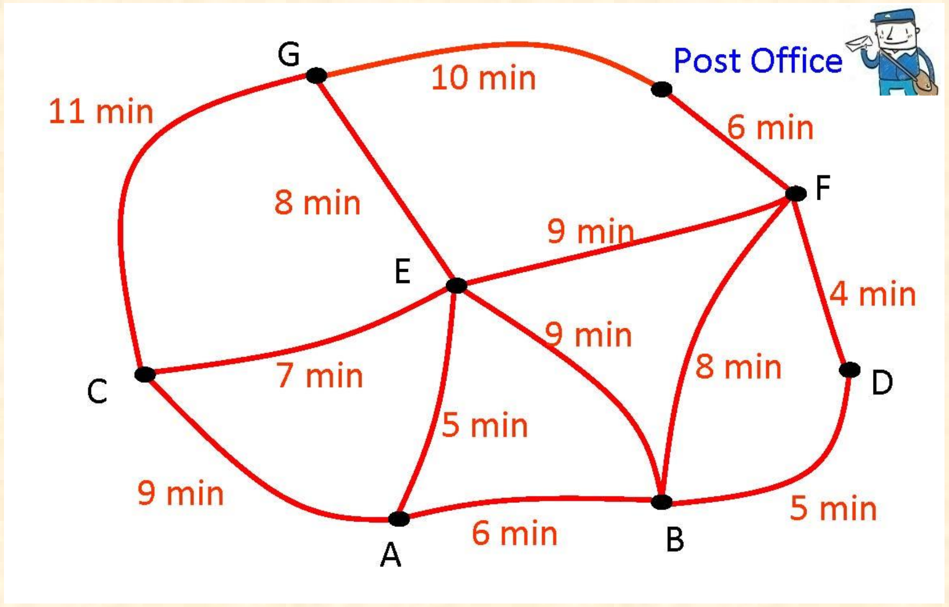
To make the graph be Eulerian, some edges need to be added to nodes with odd degrees

The postman must traverse on every added edge (otherwise, they do not need to be added).

The choice with the least total length of added edges provides an optimal solution.

Chinese Postman Problem

Example:



There are 3 options to make all node degrees become even:

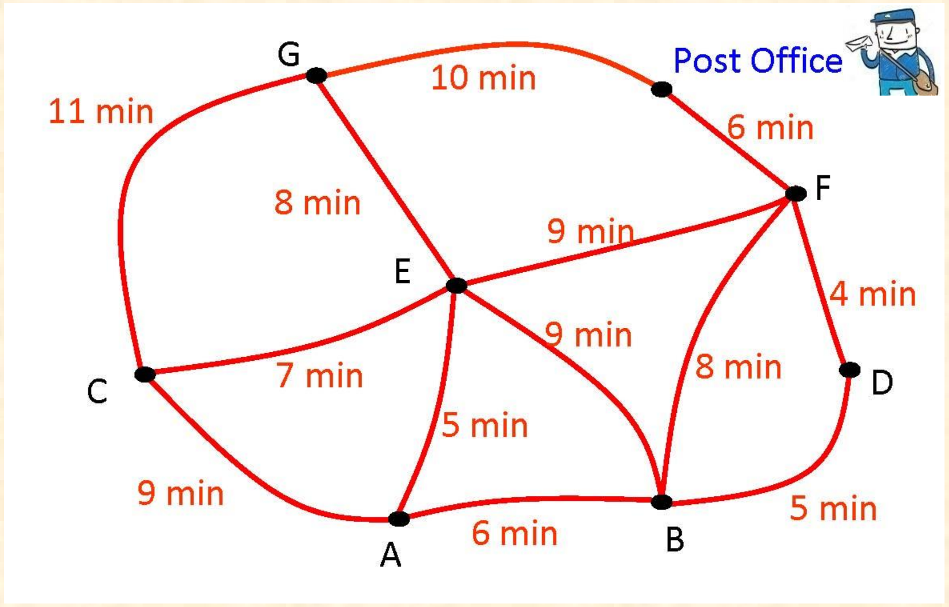
(1) add edges AC and EG

(2) add edges AEG and CE

(3) add edges AE and CG

Note:
Adding an edge does not mean to build a street, but just to walk through it twice

Chinese Postman Problem



Let T be the total time for the postman to walk through all original streets (T = 97 mins is not important in this analysis)

(1) T + (AC + EG) = T + 17 mins
(2) T + (AEG + CE) = T + 20 mins
(3) T + (AE + CG) = T + 16 mins ← optimal solution

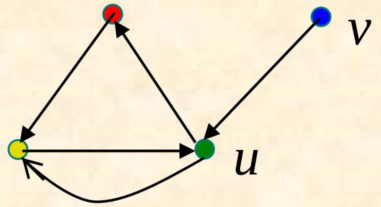
Kuan Mei-Ko (2016)

BREAK

10 minutes

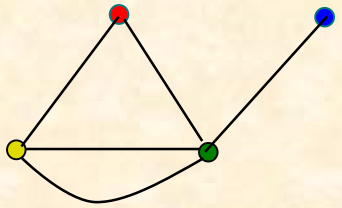
Directed Graphs

Digraph (= directed graph): edge (u,v): $u \leftarrow v$



simple digraph

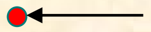
Underlying graph:

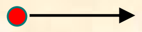


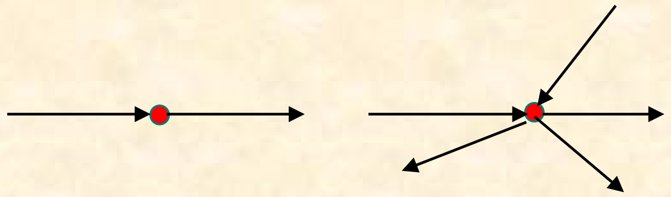
non-simple graph

Two digraphs are isomorphic if their underlying graphs are isomorphic

Degrees

In-degree: d_in 

Out-degree: d_out 

Total degree: $d = d\_in + d\_out$ 

Balanced, if $d\_in = d\_out$

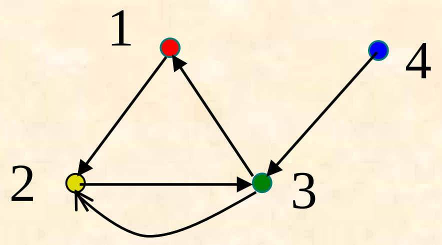
The following can be similarly defined:

- Directed walk
- Directed trail
- Directed path
- Directed circuit (cycle)
- Directed tree
- Directed Eulerian graph
- Directed Hamiltonian graph

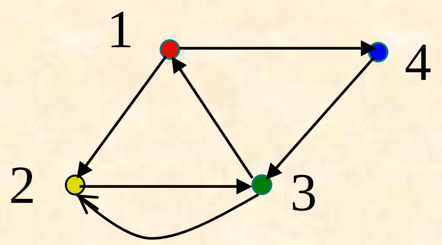
Connectivity

A digraph is connected if its underlying graph is connected.

A digraph is strongly connected if there is a directed path from any node to any other node.



← connected but not strongly connected

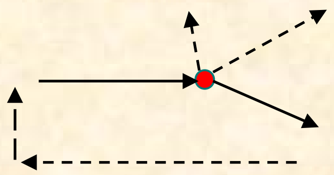


← strongly connected

Some Results

Theorem: Let G be a connected digraph. If all its nodes have $d\_in = 1$, regardless of their d_out, then G has one and only one directed circuit.

Proof.

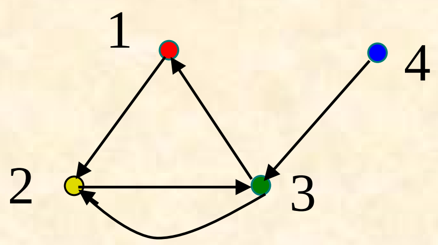


Corollary: Let G be a connected digraph. If all its nodes have $d\_out = 1$, regardless of their d_in, then G has one and only one directed circuit.

Theorem: A connected (sub)digraph is a directed circuit if and only if all its nodes satisfy $d\_in = d\_out$.

Corollary: A connected digraph contains a directed circuit if and only if it has a subgraph with all nodes satisfying $d\_in = d\_out$.

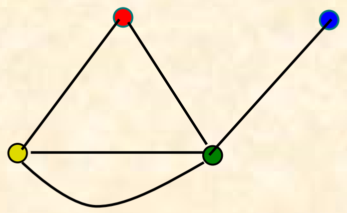
Adjacency Matrix



$$A_d = [a_{ij}] = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

(asymmetric)

a_{ij} -- from i to j



Underlying network
A is not well-defined
due to multiple edges

More Results

Directed Eulerian graphs

A directed graph is said to be balanced if for all nodes:

$$\mathbf{d}\_in = \mathbf{d}\_out$$

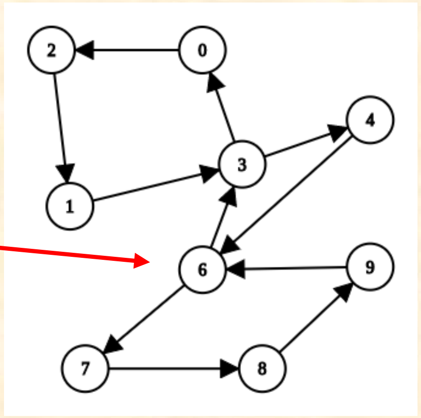
Theorem: A strongly connected digraph is directed Eulerian if and only if it is balanced.

Proof. If a node is not the end, then after

walking into it you must walk out from it,

which makes $\mathbf{d}\_in = \mathbf{d}\_out$ (may be repeated).

Continue, until returning to the starting node.



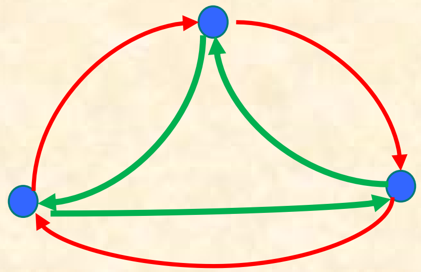
More Results

Directed Hamiltonian graphs

Theorem: Let G be a strongly connected digraph with N nodes. If all nodes satisfy both

$$d_{in} \geq N/2 \quad \text{and} \quad d_{out} \geq N/2$$

Then, G is directed Hamiltonian (sufficient condition)



$$d_{in} = 2 > \frac{3}{2}$$

$$d_{out} = 2 > \frac{3}{2}$$

Density of Graph

Recall undirected graph:

$$Den = \frac{\overset{\text{total number of edges}}{L}}{N(N-1)/2} = \frac{2\sum_{i>j=1}^{N} a_{ij}}{N(N-1)}$$

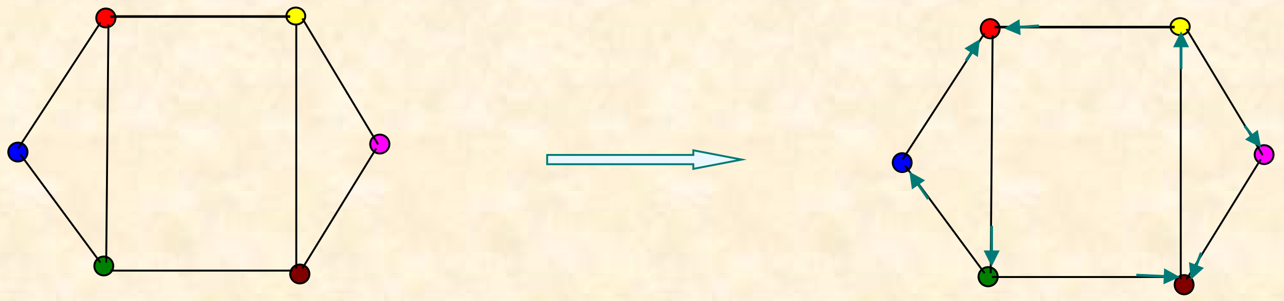
- Fully connected graph: $\sum_{i>j=1}^{N} a_{ij} = N(N-1)/2$, so $Den = 1$
- Isolated nodes without edges: $\sum_{i>j=1}^{N} a_{ij} = 0$, so $Den = 0$

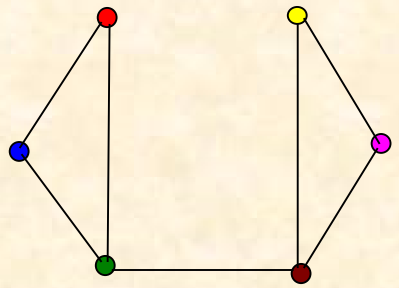
directed graph:

$$Den = \frac{L}{N(N-1)} = \frac{\sum_{i,j=1}^{N} a_{ij}}{N(N-1)}$$

Orientable Graphs

A graph is orientable if its edges can be directed such that the resulting digraph is strongly connected.





← not orientable

→ importance of bridges

→ importance of circuits

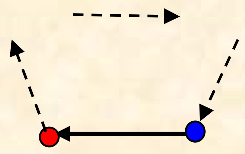
Some More Results

Theorem: A connected graph is orientable if and only if every edge of the graph is contained in a circuit.

Proof.

Necessity - G is orientable → every edge can be directed

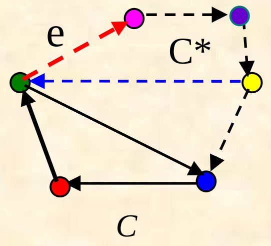
→ connecting from any node to any other node →



Sufficiency - Choose any circuit C and direct it cyclically.

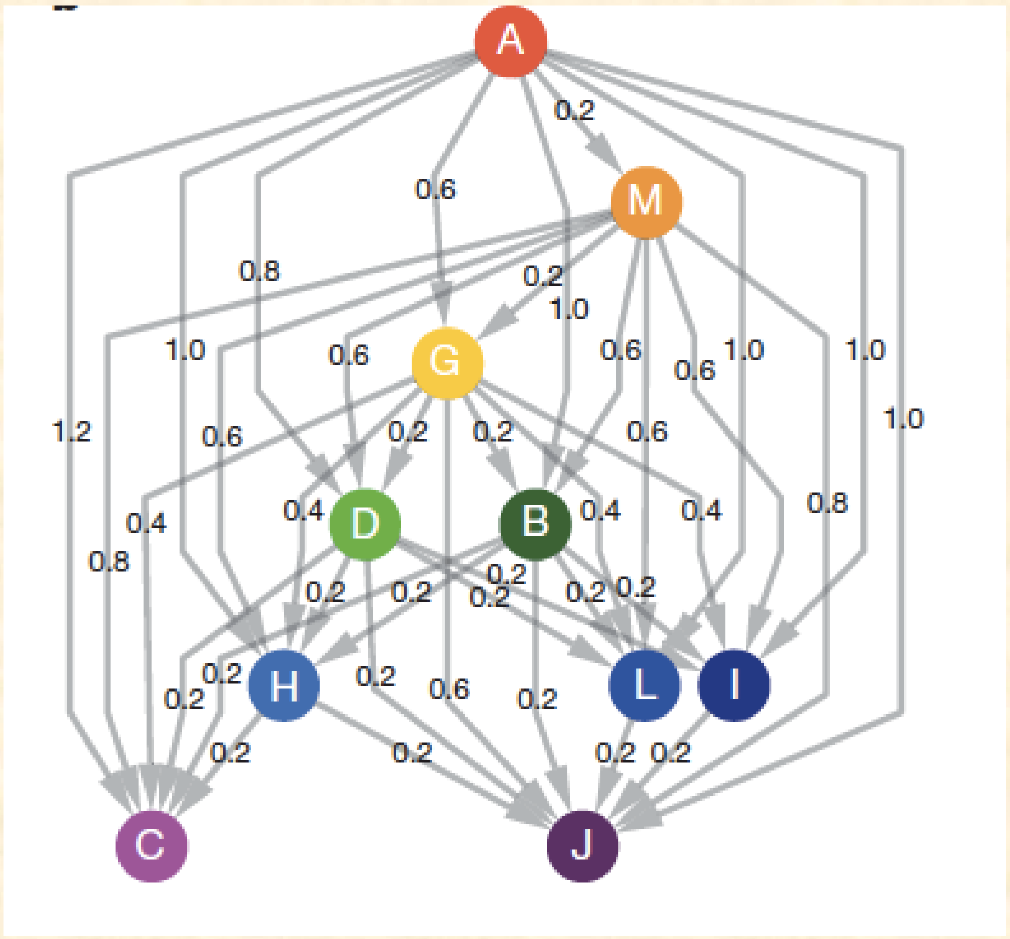
If $C = G$, done.

If $C \subset G$, then consider any adjacent edge e. By assumption, e is in some circuit $e \in C^* \subset G$ Since C^* is a circuit, there is a path to come back to e. One can then direct this path cyclically.



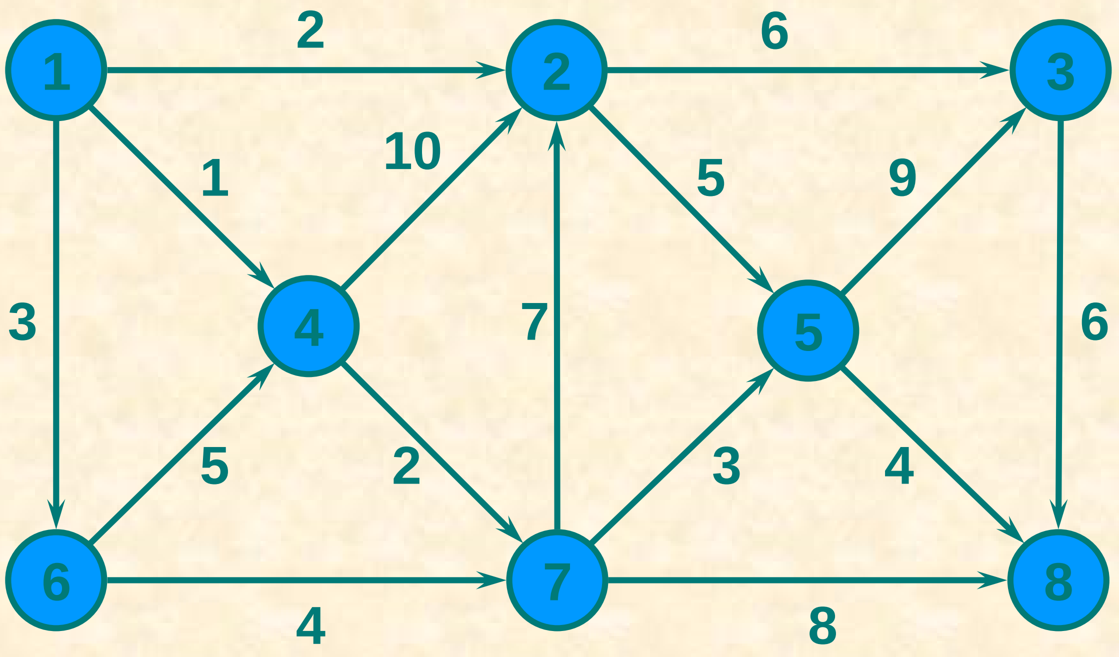
Repeat the above for all edges, then done.

Directed and Weighted Networks

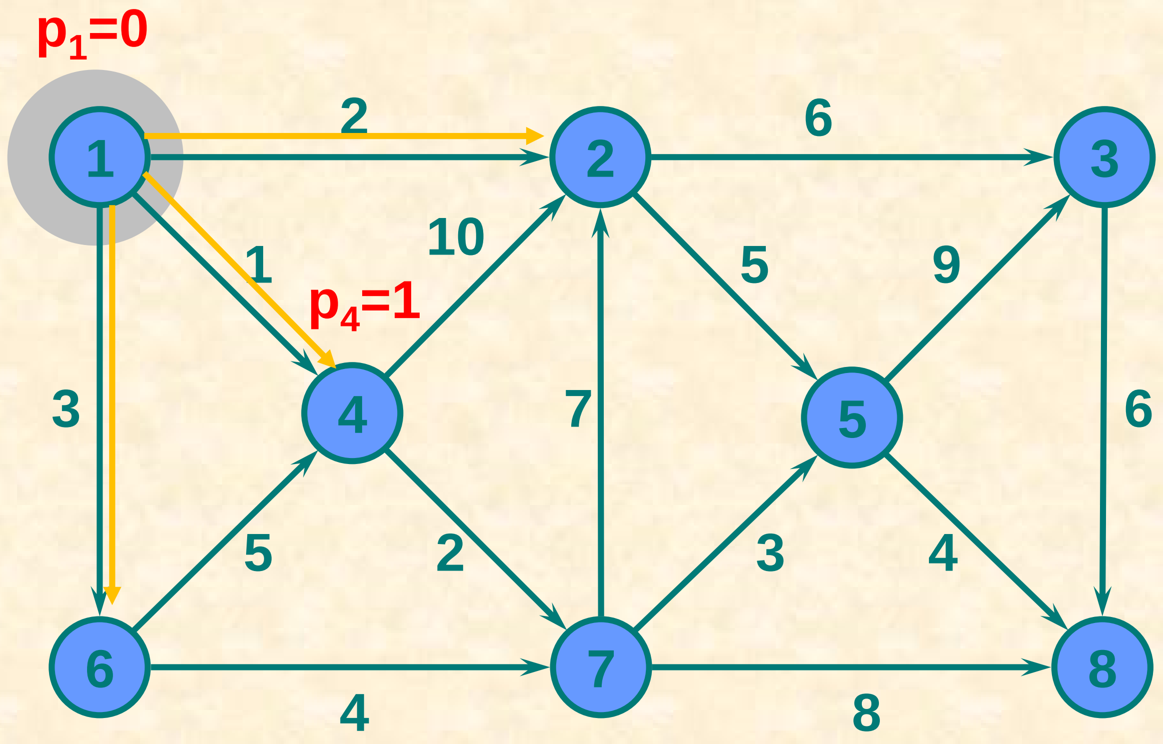


Applications: coding, high-security computer networks (data diode), Industrial control systems, …

Shortest Path: For the following directed graph, find the shortest path from Node **1** to Node **8**


$$\begin{tikzcd}
1 \arrow[rr, "2"] \arrow[dr, "1"] \arrow[dd, "3"'] & & 2 \arrow[rr, "6"] \arrow[dr, "5"] & & 3 \arrow[dd, "6"] \\
& 4 \arrow[ur, "10"] \arrow[dr, "2"] & & 5 \arrow[ur, "9"] \arrow[dr, "4"] & \\
6 \arrow[ur, "5"] \arrow[rr, "4"'] & & 7 \arrow[uu, "7"] \arrow[ur, "3"] \arrow[rr, "8"'] & & 8
\end{tikzcd}$$

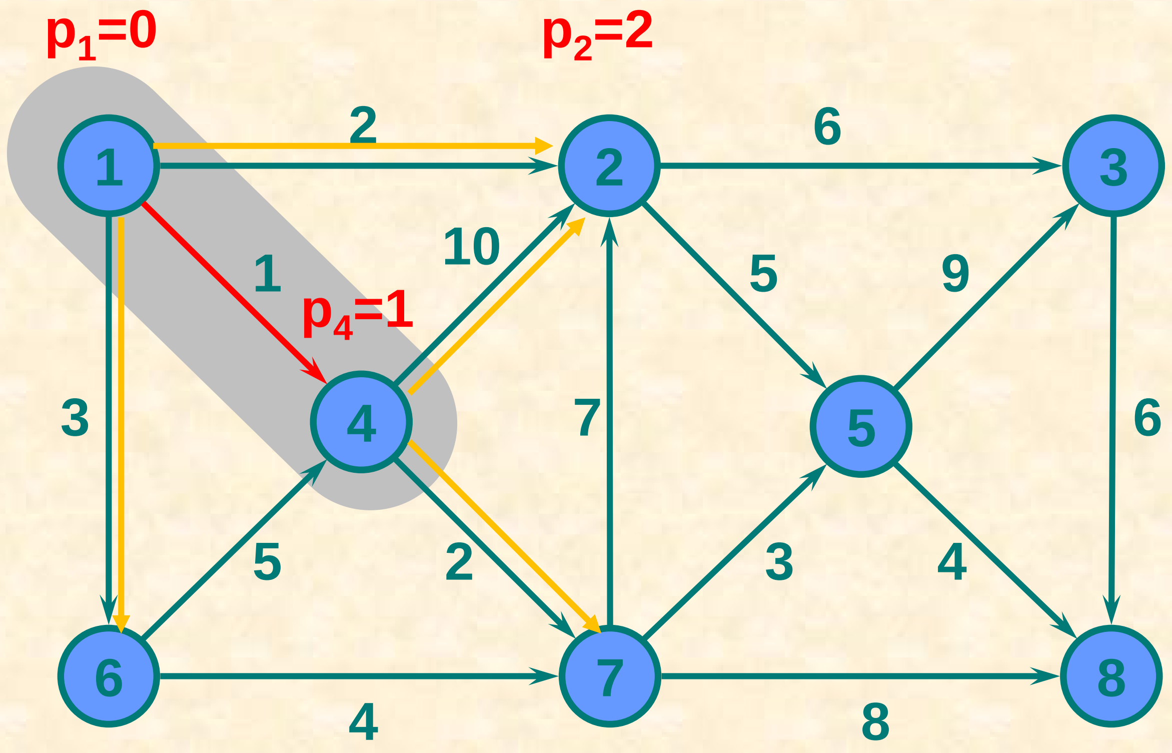
Path: X={1}, p_1=0



min $\{c_{12},c_{14},c_{16}\}$=min {0+2,0+1,0+3}=min {2,1,3}=1

X={1,4}, p_4=1

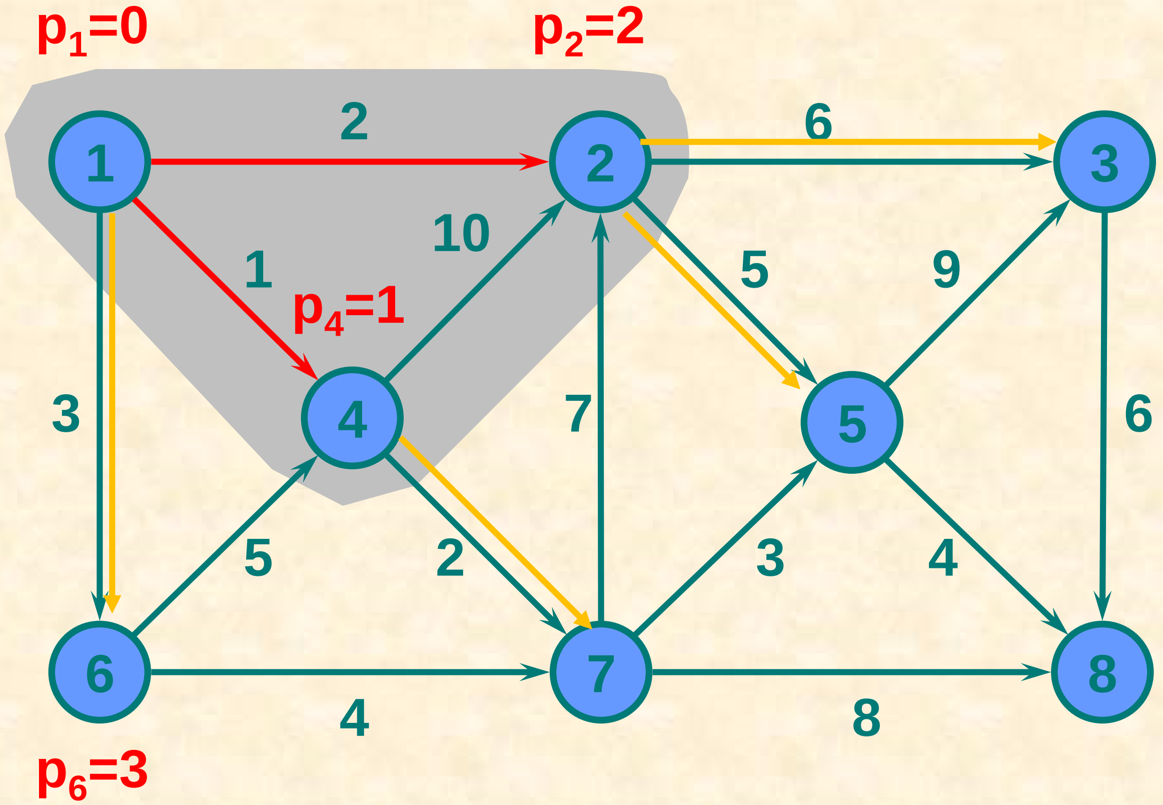
$X=\{1,4\}$



$\min\{c_{12},c_{16},c_{42},c_{47}\}=\min\{0+2,0+3,1+10,1+2\}=\min\{2,3,11,3\}=2$

$X=\{1,2,4\}$, $p_2=2$

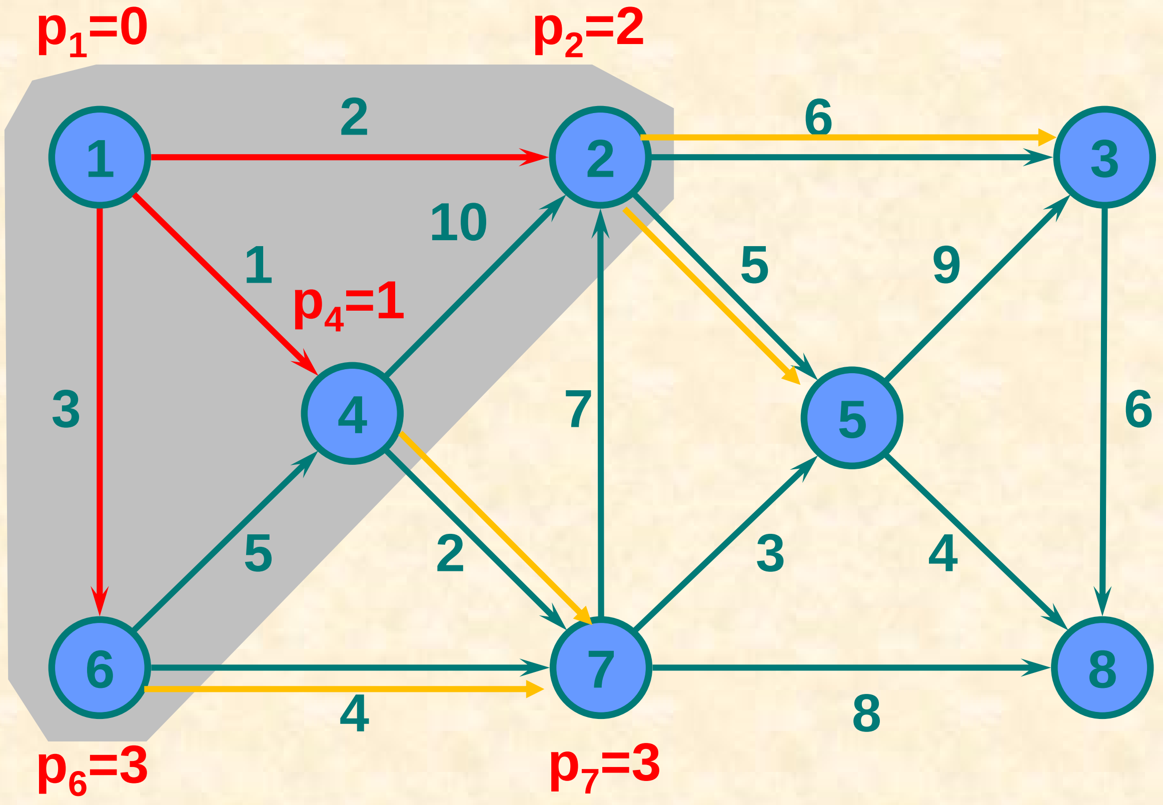
X={1,2,4}



min $\{c_{16},c_{23},c_{25},c_{47}\}$=min {0+3,2+6,2+5,1+2}=min {3,8,7,3}=3

X={1,2,4,6}, $p_6=3$

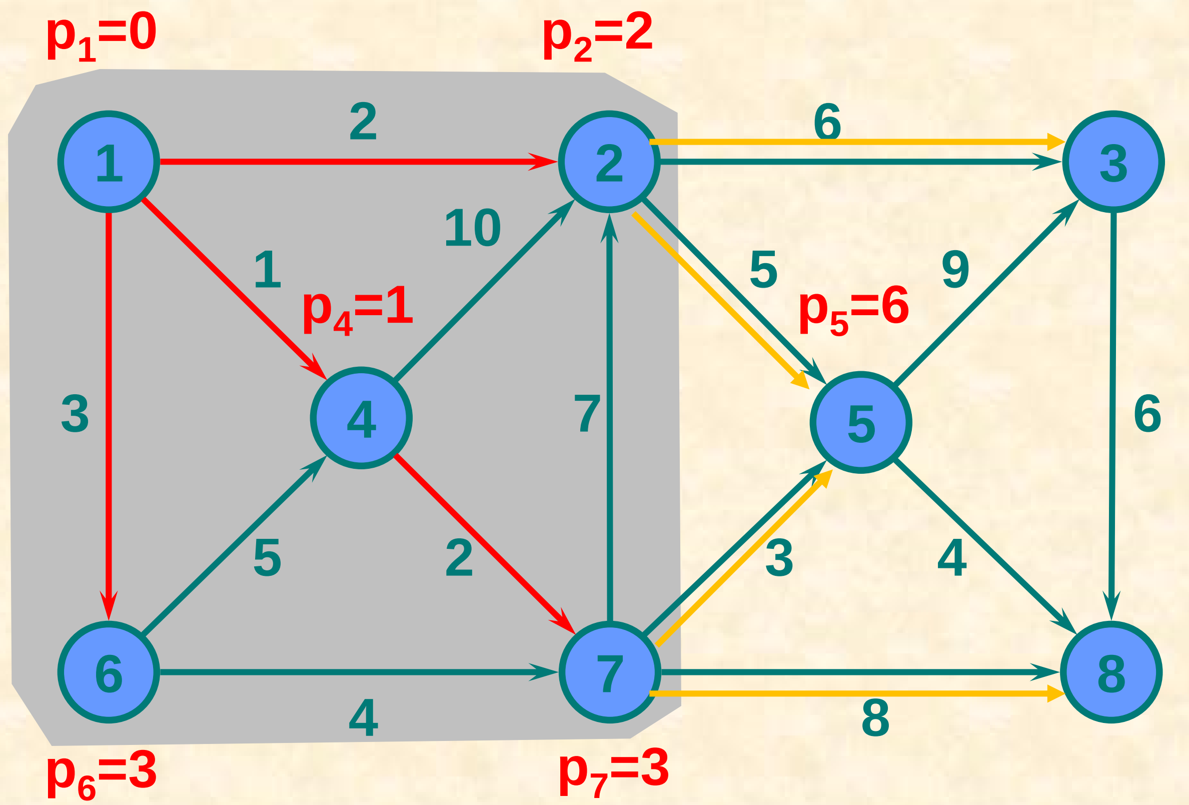
X={1,2,4,6}



min $\{c_{23},c_{25},c_{47},c_{67}\}$=min {2+6,2+5,1+2,3+4}=min {8,7,3,7}=3

X={1,2,4,6,7}, $p_7=3$

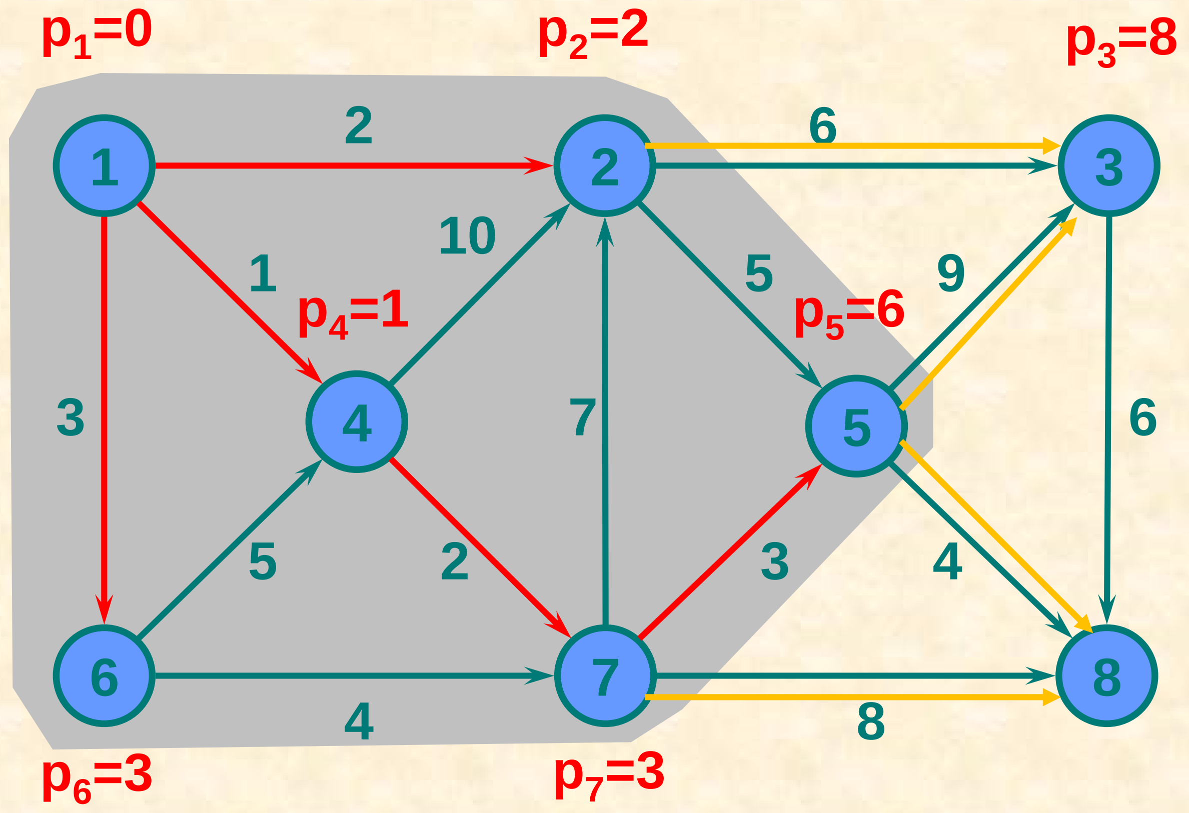
$X=\{1,2,4,6,7\}$



$\min\{c_{23},c_{25},c_{75},c_{78}\}=\min\{2+6,2+5,3+3,3+8\}=\min\{8,7,6,11\}=6$

$X=\{1,2,4,5,6,7\}$, $p_5=6$

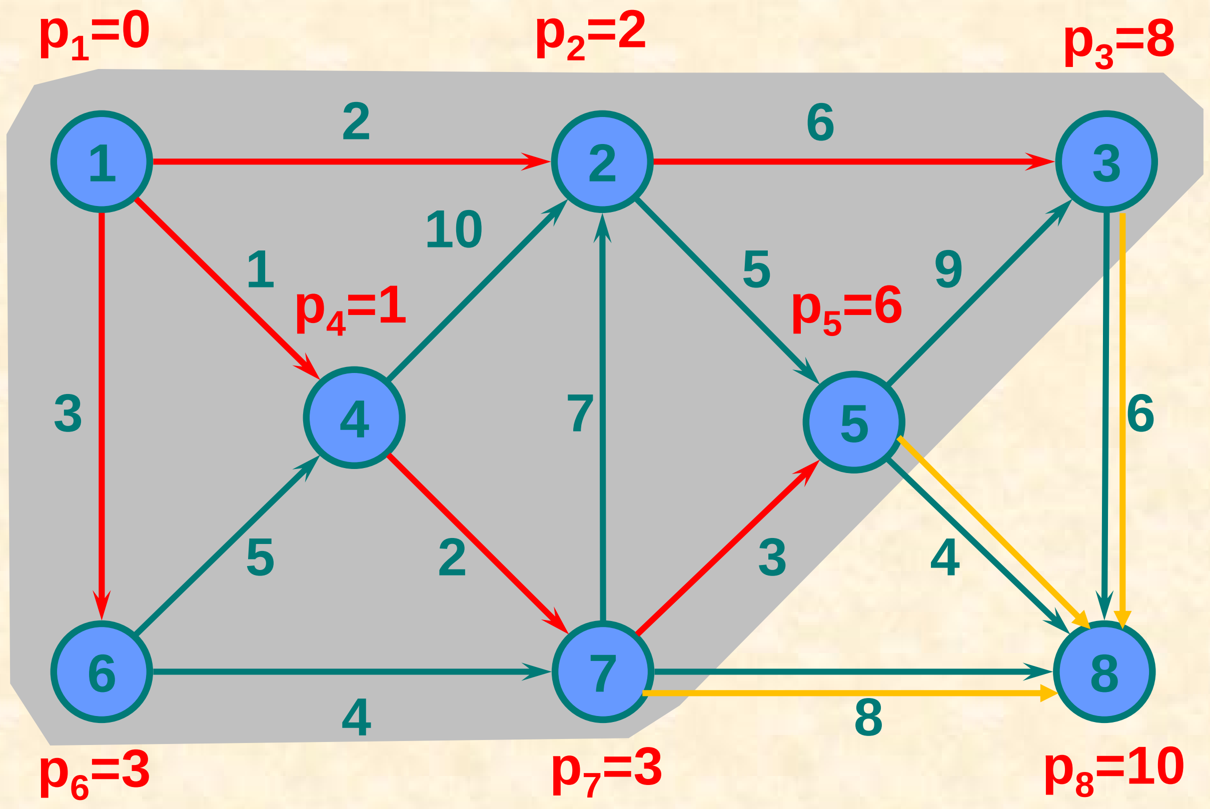
$X=\{1,2,4,6,7\}$



$\min\{c_{23},c_{53},c_{58},c_{78}\}=\min\{2+6,6+9,6+4,3+8\}=\min\{8,15,10,11\}=8$

$X=\{1,2,3,4,5,6,7\}$, $p_3=8$

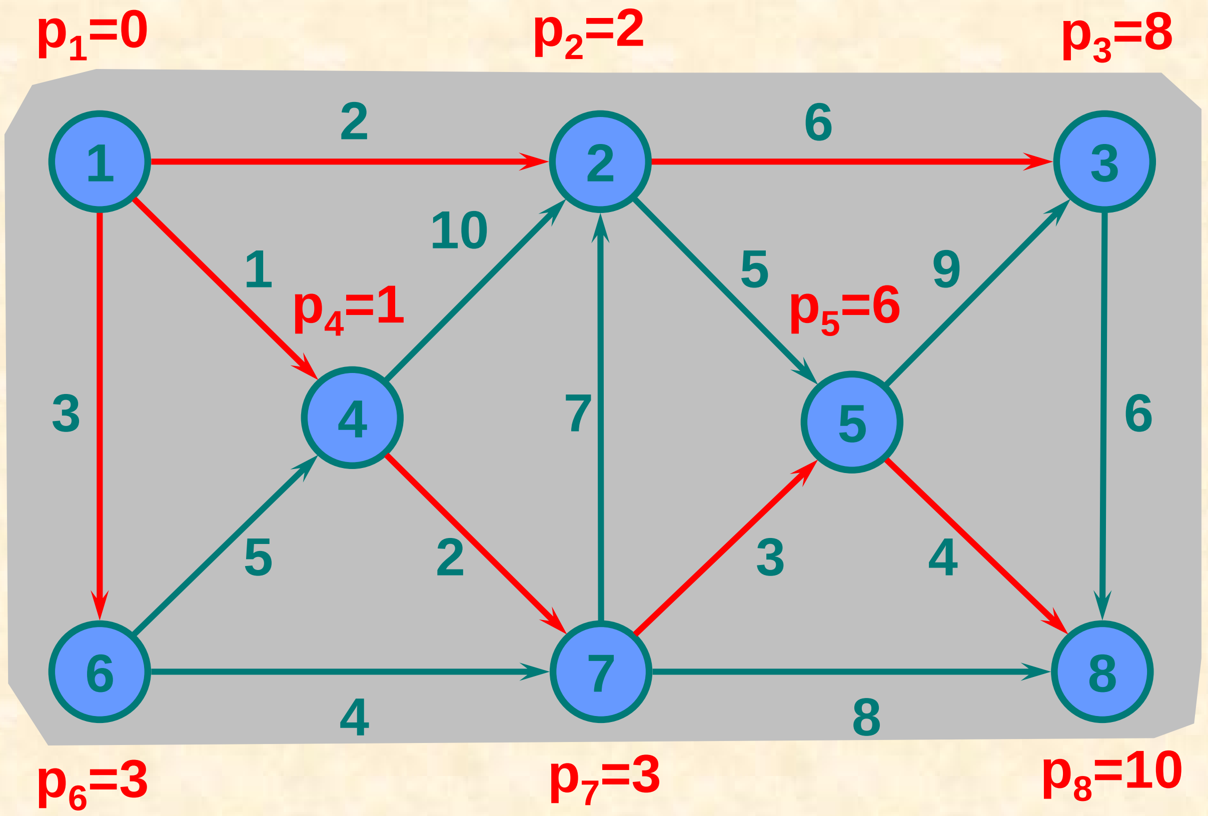
X={1,2,3,4,6,7}



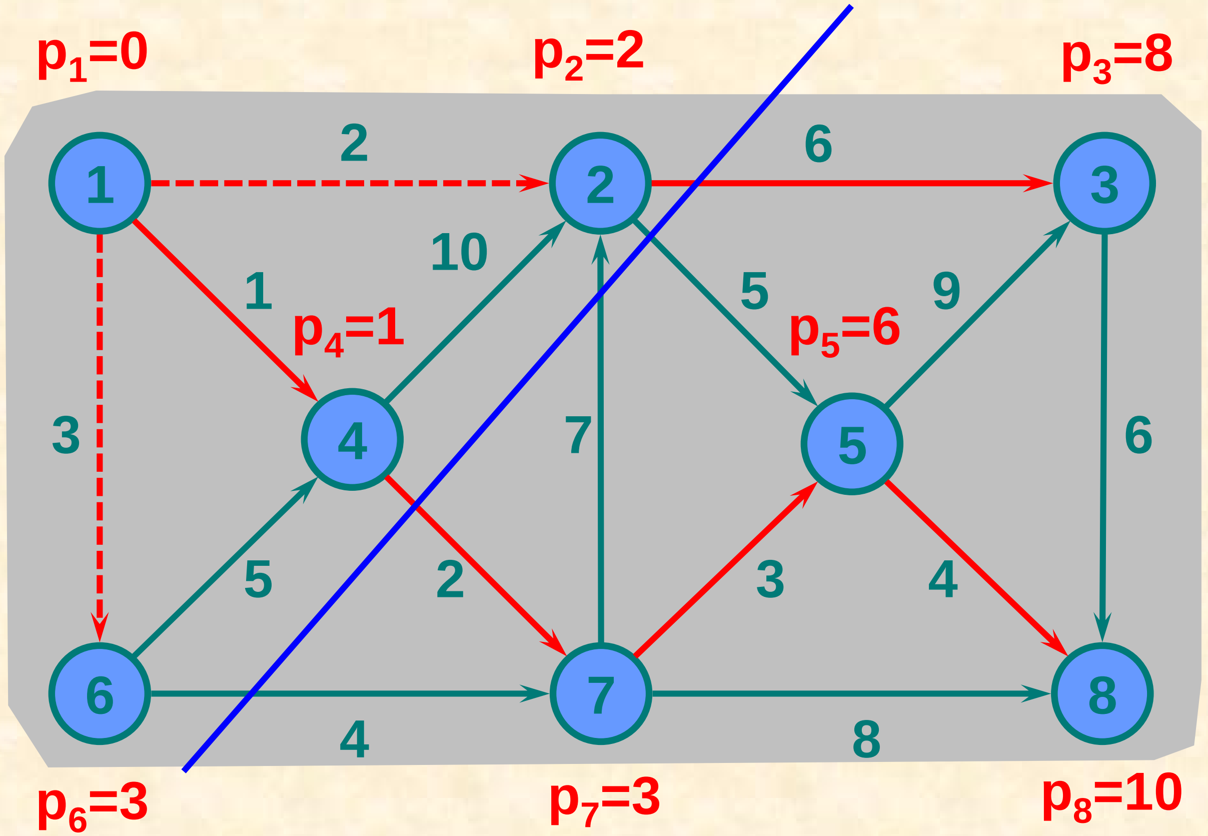
min $\{c_{38},c_{58},c_{78}\}$=min {8+6,6+4,3+7}=min {14,10,11}=10

X={1,2,3,4,5,6,7,8}, $p_8=10$

X={1,2,3,4,6,7,8}



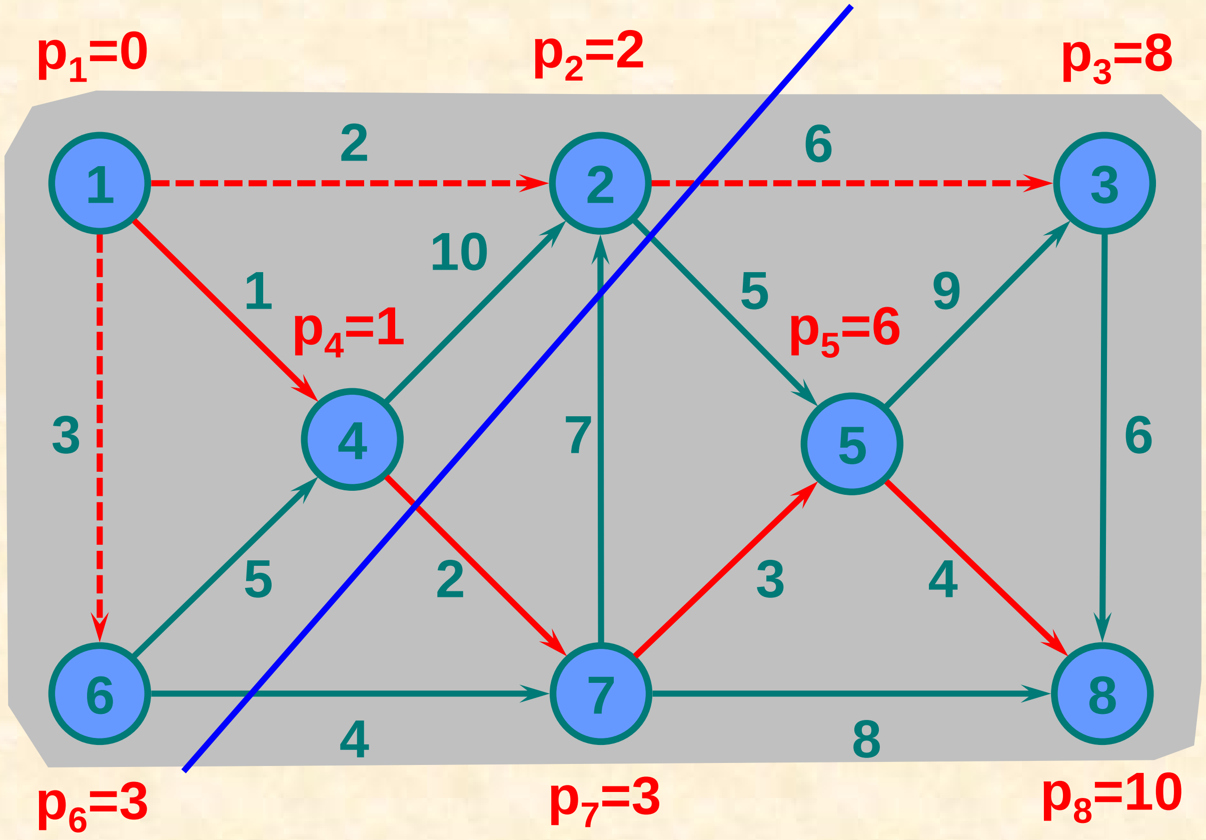
X={1,2,3,4,6,7,8}



Upper-left corner: Compare c_{12} = 2, c_{14} = 1, c_{16} = 3

→ Keep c_{14} = 1

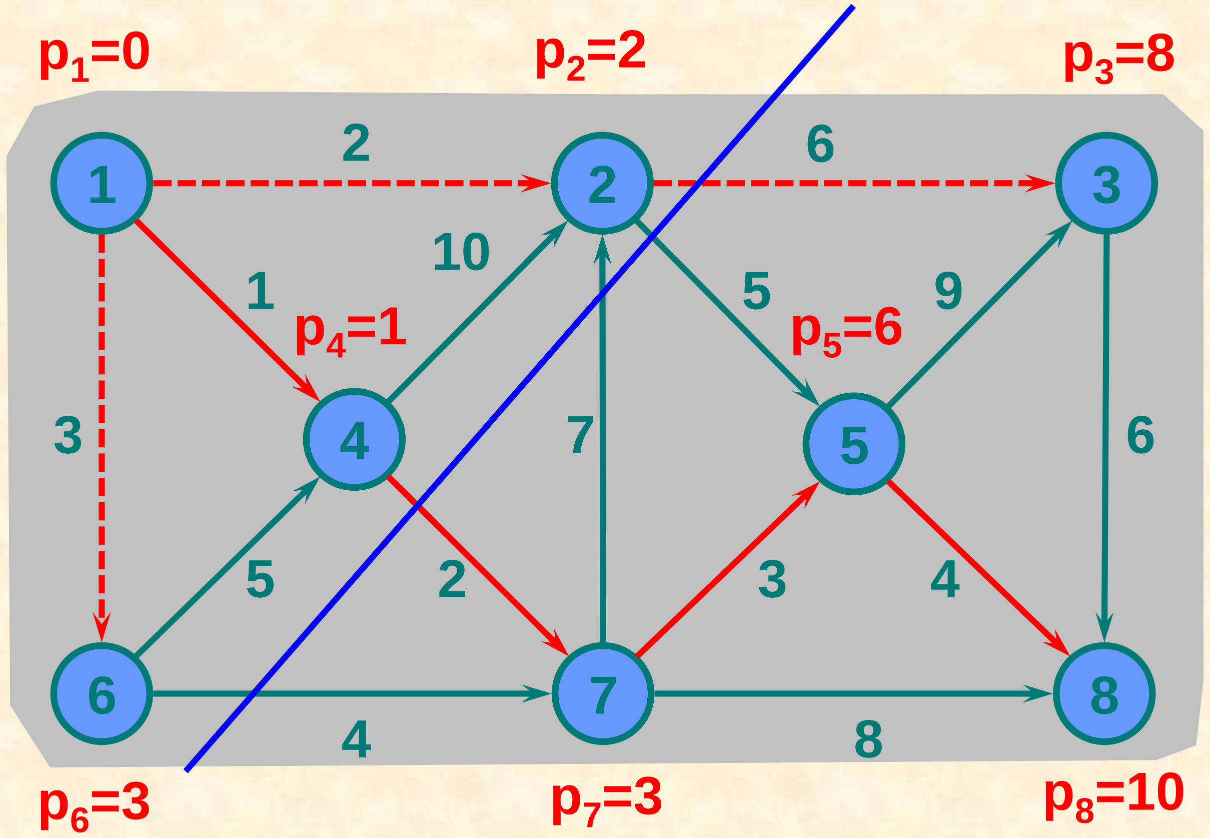
X={1,2,3,4,6,7,8}



Middle part: Compare c_{23} = 6, c_{47} = 2

→ Keep c_{47} = 2

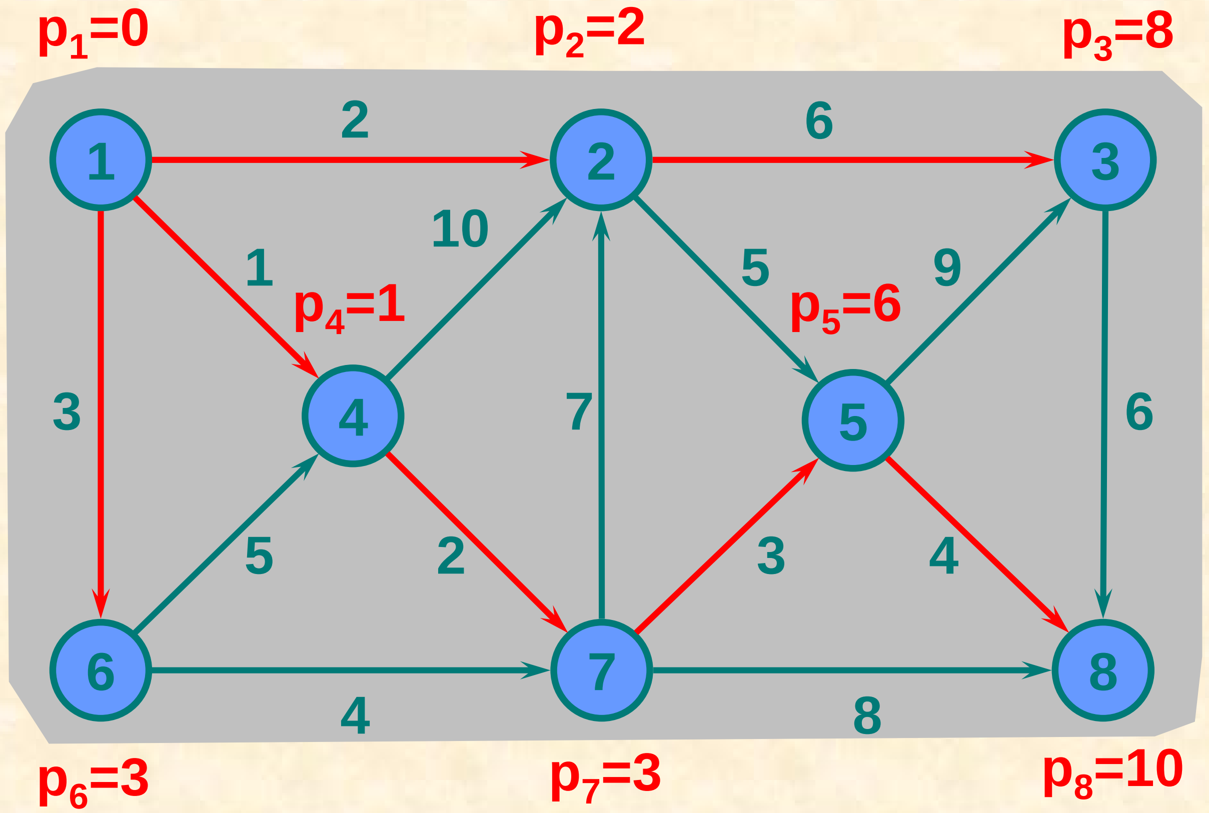
X={1,2,3,4,6,7,8}



Middle part: Compare $c_{23} = 6$, $c_{47} = 2$

→ Keep $c_{47} = 2$ → The rest: $c_{75} = 3$, $c_{58} = 4$

X={1,2,3,4,6,7,8}



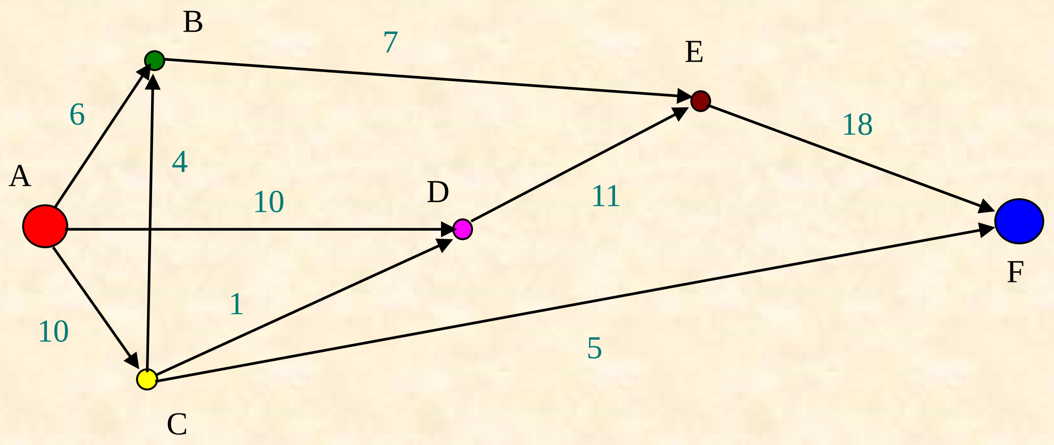
Solution:

The shortest path from Node 1 to Node 8 is 10, via {1，4，7，5，8}

Longest Path or Maximum Flows

Consider the following roadmap, where weights are profit values.

Starting from A, follow the directed road to traverse to F, not allowed to repeat any road, finally arrive at F while making the maximum profits.



Ford–Fulkerson Algorithm

$l(A) = 0$

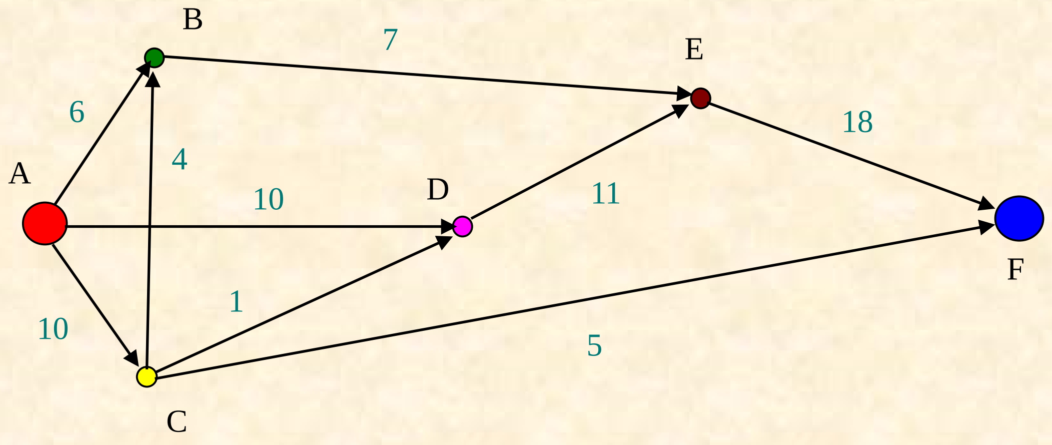
$l(B) = l(A) + 6 = 6$

$l(C) = l(A) + 10 = 10 \rightarrow l(B) = l(C) + 4 = 14 \rightarrow l(E) = l(B) + 7 = 21 \rightarrow l(F) = l(E) + 18 = 39$

$l(D) = l(C) + 1 = 11 \rightarrow l(E) = l(D) + 11 = 22 \rightarrow l(F) = l(E) + 18 = 40$

$l(F) = l(C) + 5 = 15$

$l(D) = l(A) + 10 = 10 \rightarrow l(E) = l(D) +11 = 21 \rightarrow l(F) = l(E) + 18 = 39$



Optimal Solution

$l(A) = 0$

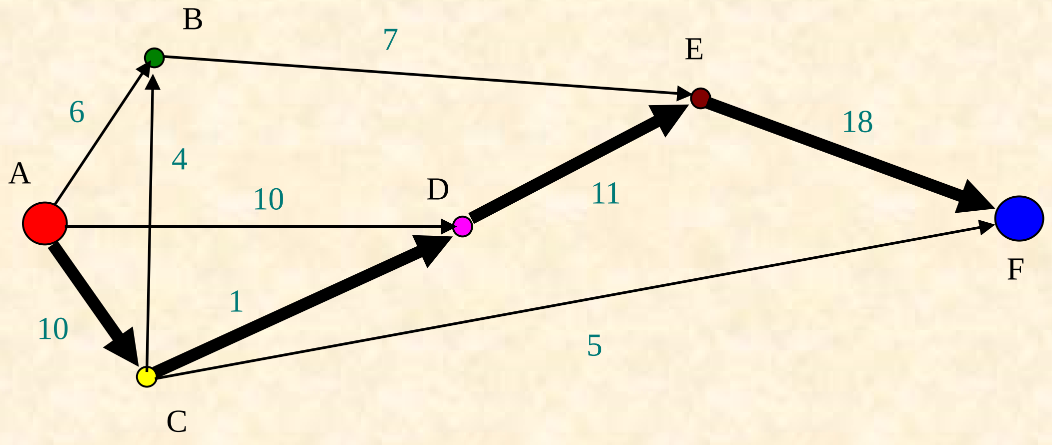
$l(B) = l(A) + 6 = 6$

$l(C) = l(A) + 10 = 10 \rightarrow l(B) = l(C) + 4 = 14 \rightarrow l(E) = l(B) + 7 = 21 \rightarrow l(F) = l(E) + 18 = 39$

$l(D) = l(C) + 1 = 11 \rightarrow l(E) = l(D) + 11 = 22 \rightarrow l(F) = l(E) + 18 = 40$

$l(F) = l(C) + 5 = 15$

$l(D) = l(A) + 10 = 10 \rightarrow l(E) = l(D) +11 = 21 \rightarrow l(F) = l(E) + 18 = 39$

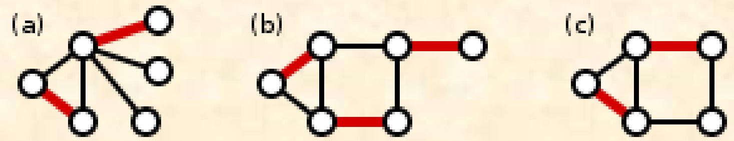


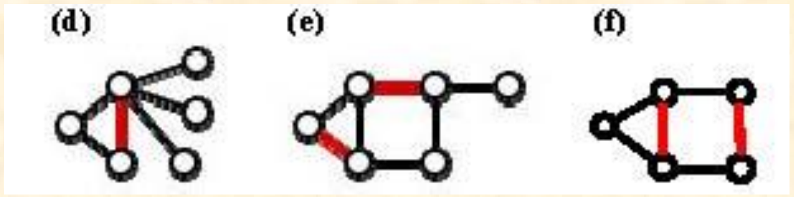
Matching

Undirected Graphs

A matching in a graph G is a set of non-adjacent edges

Examples: all sets of red-colored edges in the following figures are matchings (in fact, they are also maximal matchings):

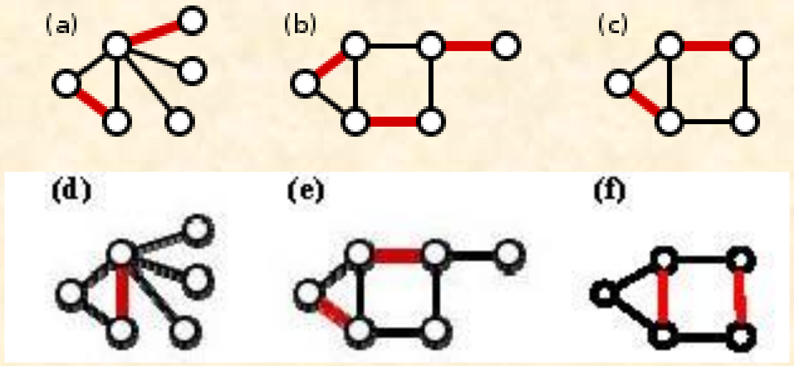


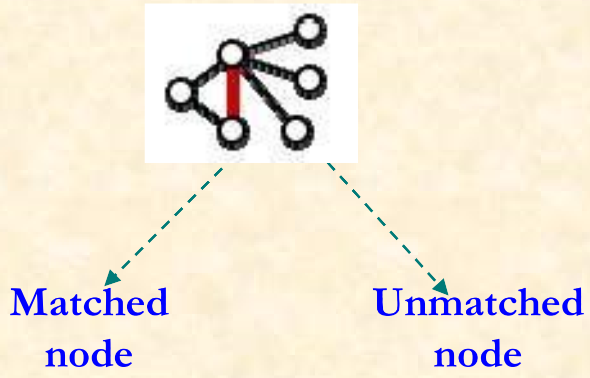


A maximal matching is one on which no more edge can be added.

A maximum matching, furthermore, is a maximal matching that contains the largest possible number of edges.

For example: Matchings in figures (a), (b), (c) and (f), are maximum, but those in (d) and (e) are only maximal but not maximum.





Both maximal and maximum matchings in a graph may not be unique. This is clear by inspecting the above figures.

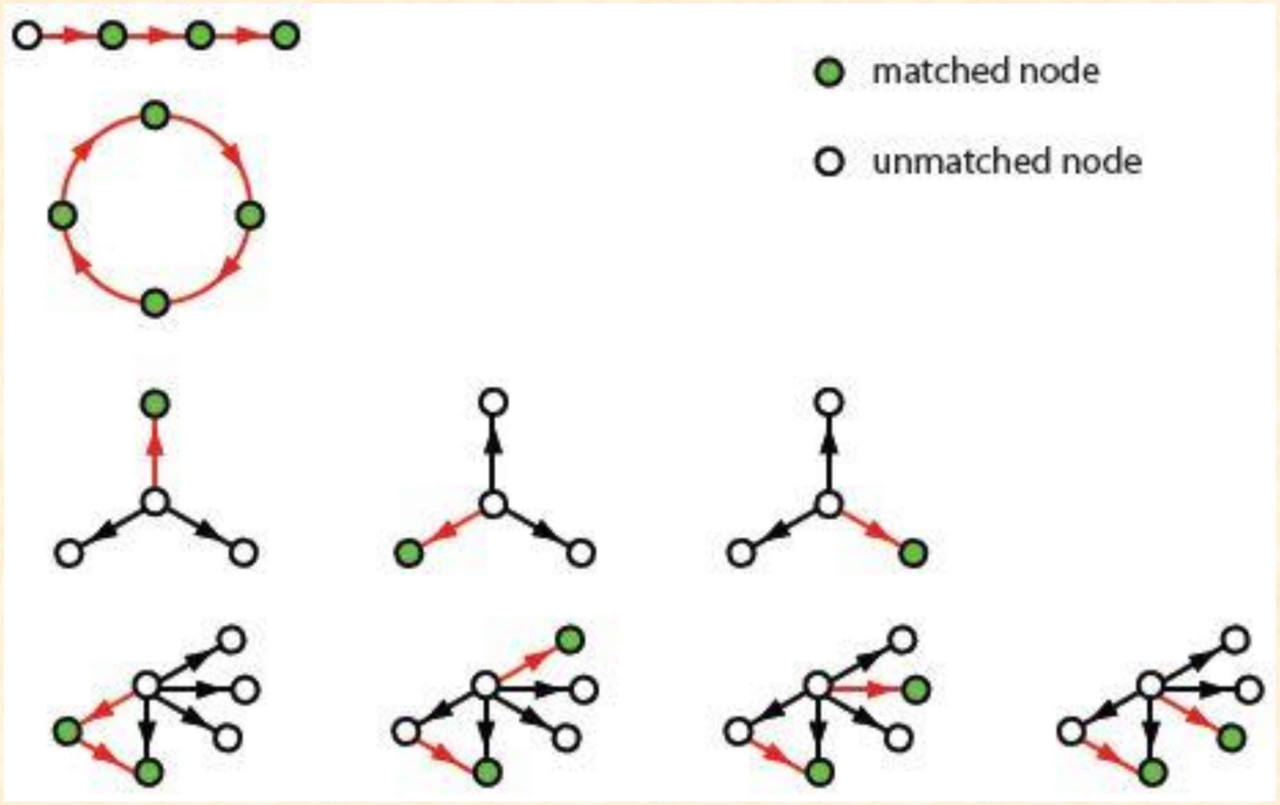
A **node** is **matched** if it is an end-point of an edge in a matching; otherwise, the node is **unmatched**.

A **perfect matching** is a matching which matches all nodes of the graph. Figure (b) is a perfect matching, while the others are not.

Every perfect matching is always maximum (hence, also maximal)

Matching in Directed Graphs

- **Matching**: a set of directed edges without common heads and tails
- **Matched node**: the head node of a matching edge

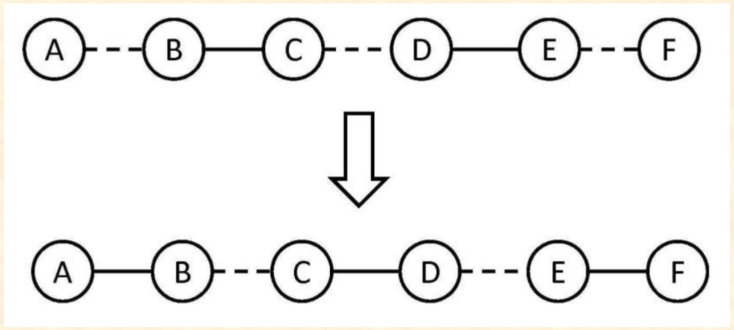


A perfect matching is a maximum matching where all nodes are matched nodes

Matching Paths

Alternating path is a path whose edges are alternately in and out of the matching (for directed graphs, also change its direction to opposite)

Example:



At top, there is a matching {BC, DE}
At bottom, it is an **alternating path**

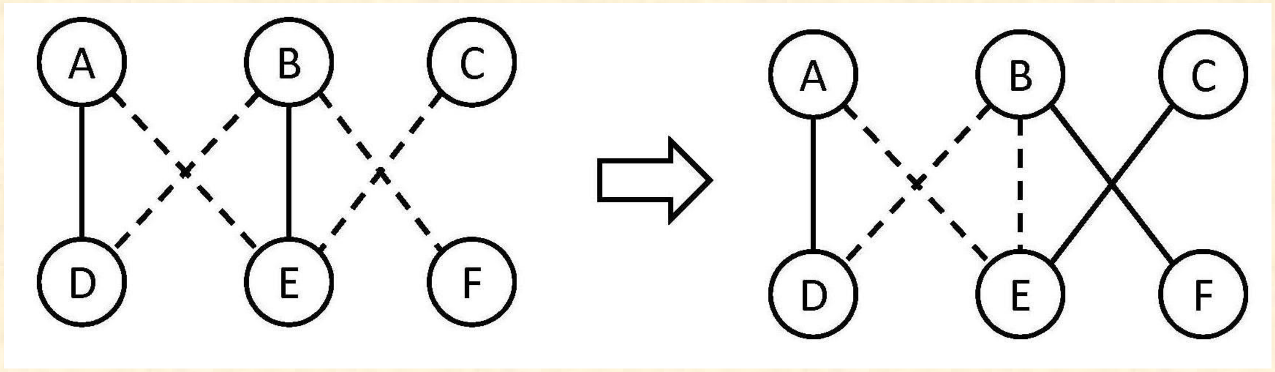
Note: at top, nodes A, F are unmatched; at bottom, nodes A, F become matched. In such a case, the bottom alternating path is called an augmented path of the above path (next page)

Augmenting path is an alternating path between two unmatched nodes

Example:

Consider the graph on the left-hand side, which has a matching {AD, BE}. Here, C and F are unmatched nodes.

For path C-E-B-F, which goes from unmatched node C to unmatched node F, its **augmenting path** is the one on the right-hand side, "C-E-B-F", in which dashed lines became solid lines and solid lines became dashed lines.



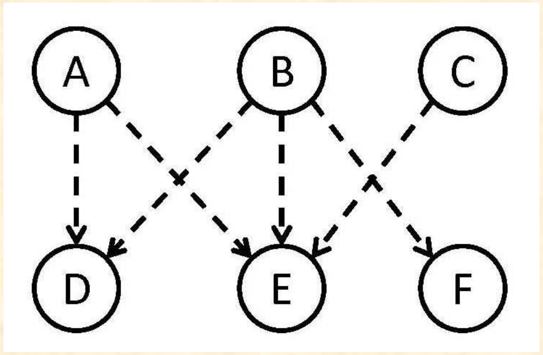
Augmenting path algorithm (to find maximum matching)

This algorithm works only for bipartite graphs

Start with an empty matching
Rule: If there is an augmenting path, do an augmentation

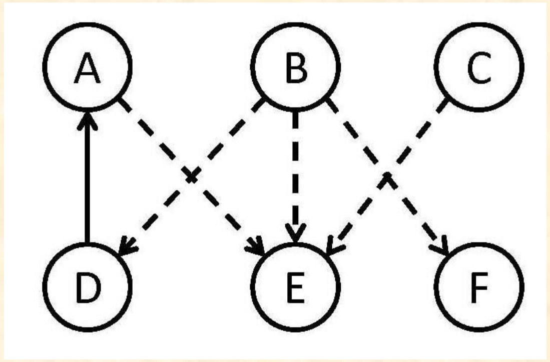
Procedure: Consider the directed bipartite graph (a)

Step 1) Start from node A, search for a path (here, it can go to either D [or E, which is not chosen here], because of the given directions); then augment it, which yields graph (b) [since it is directed, remember to change its direction after augmentation]



(a)

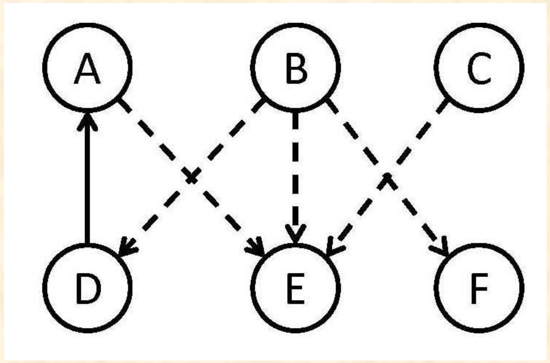
0 solid edge

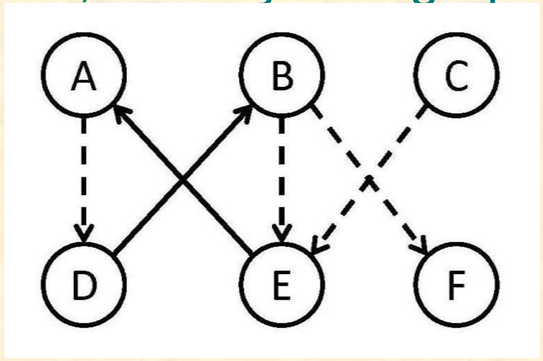


(b)

1 solid edge

Step 2) Start from node B, search for a path, which can go through B-D-A-E [or stop only at E or F]; augment it, which yields graph (c)





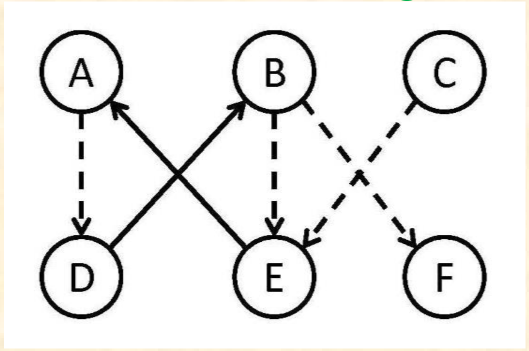
(b)

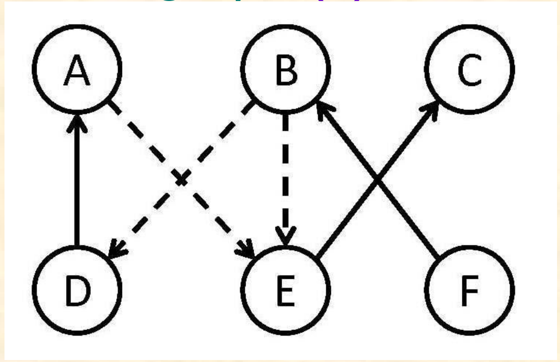
1 solid edge

(c)

2 solid edges

Step 3) Start from node C, search for a path, which can only go to F: C-E-A-D-B-F; then augment it, so as to obtain graph (d).



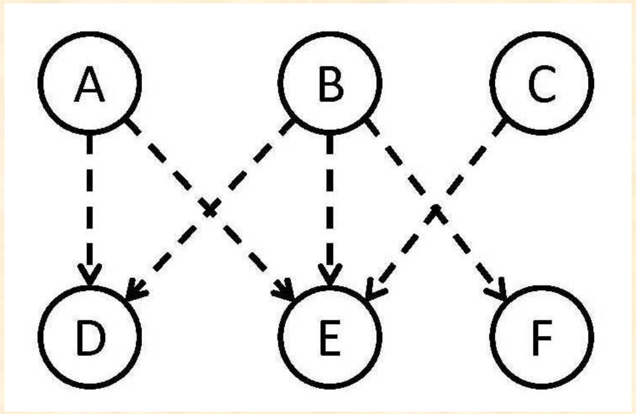


(c)

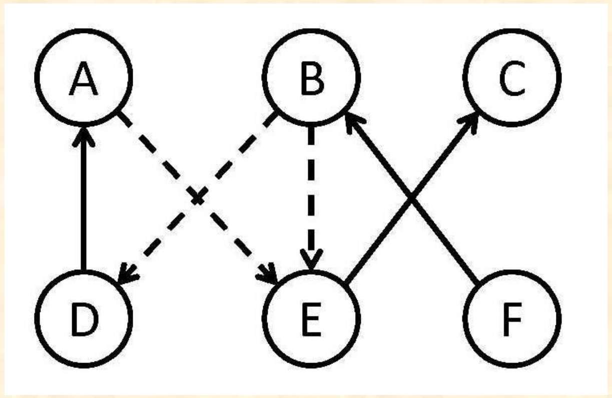
2 solid edges

(d)

3 solid edges



Summary



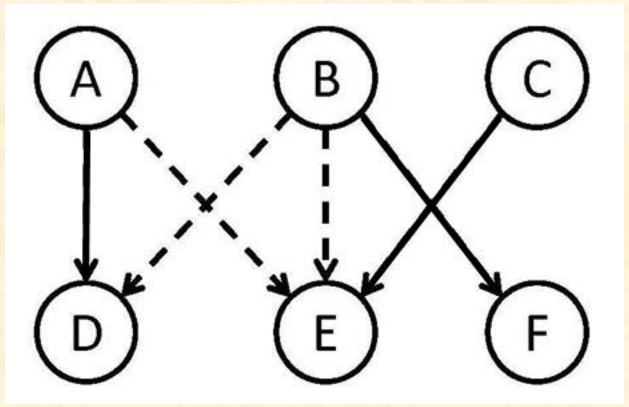
(a) → (b) → (c) → (d)

The search has completed on one side (top) of the bipartile graph, stop

Finally, change all the edge directions to the opposite

Solution:

(maximum matching)



Solution may not be unique

Theorem:

A matching is maximum if and only if there is no augmenting path

Maximum matching is useful for network control

(To be studied in Lecture-9)

End