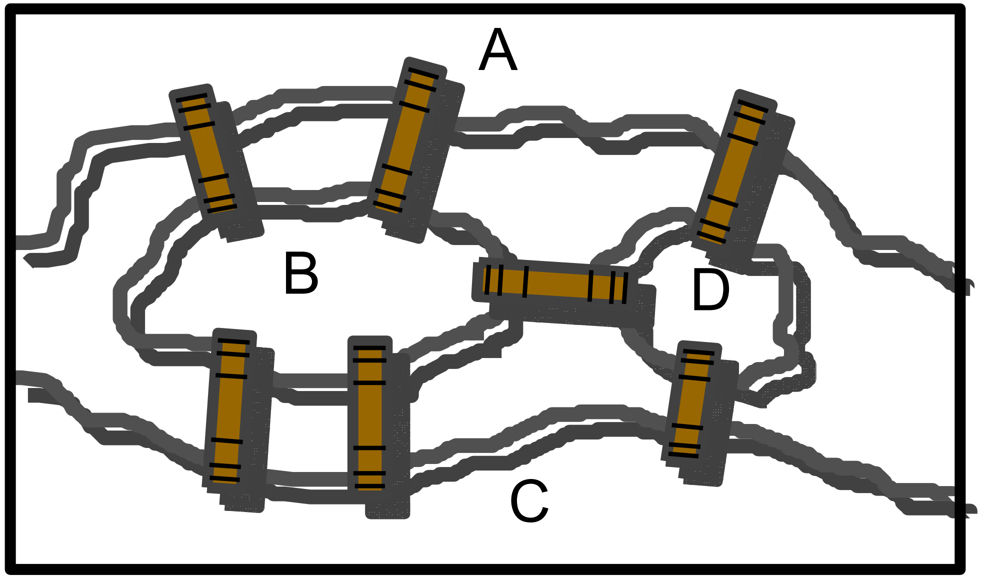
Graphs

สมชาย ประสิทธิ์จูตระกูล
ภาควิชาวิศวกรรมคอมพิวเตอร์
จุฬาลงกรณ์มหาวิทยาลัย
(04/11/48)

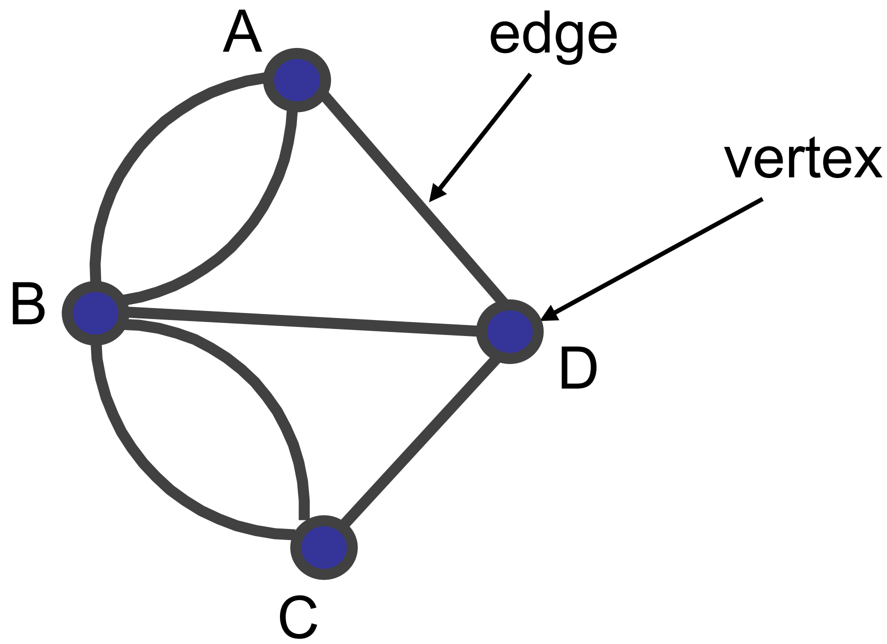
Graphs



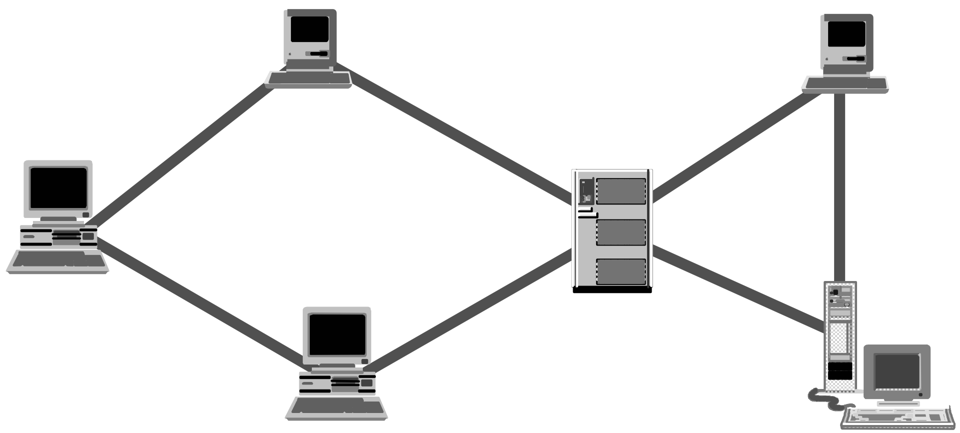
Königsberg Bridge Problem

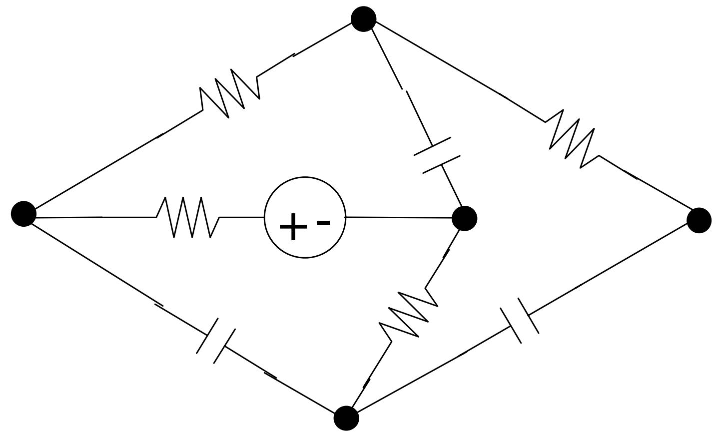
1736:

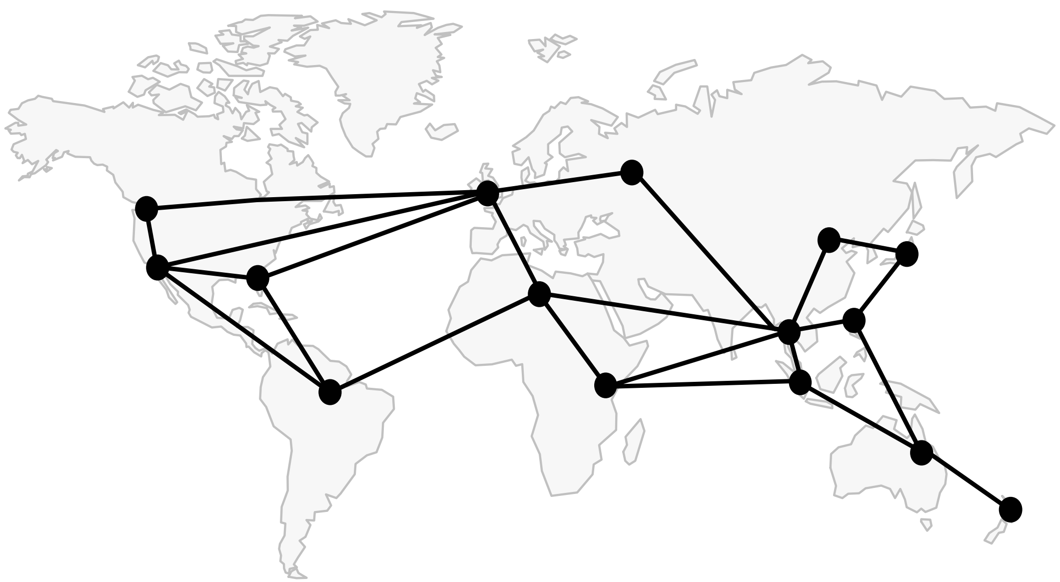
Leonhard Euler

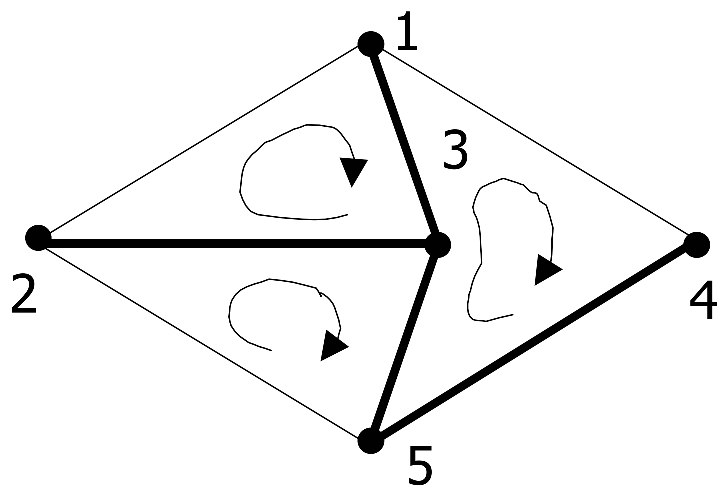


Applications

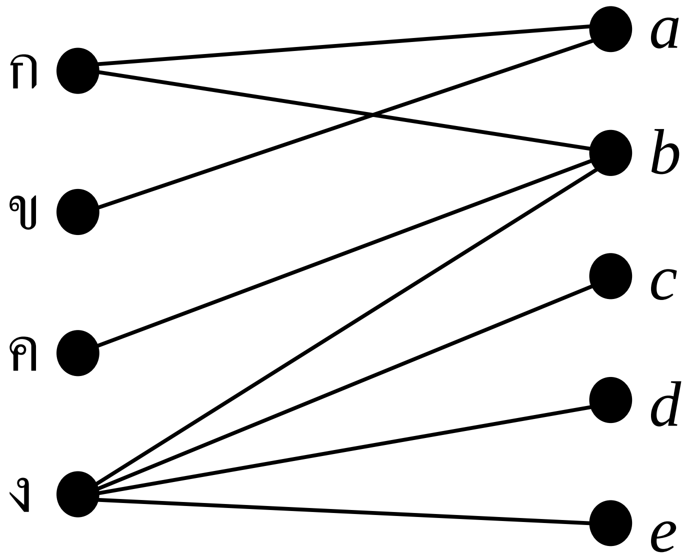


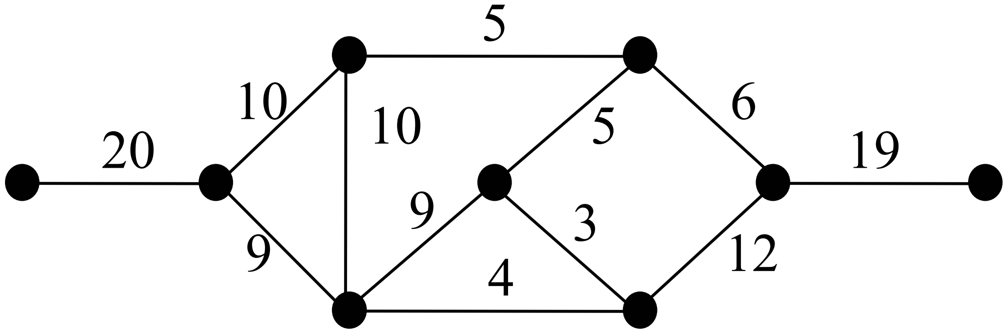


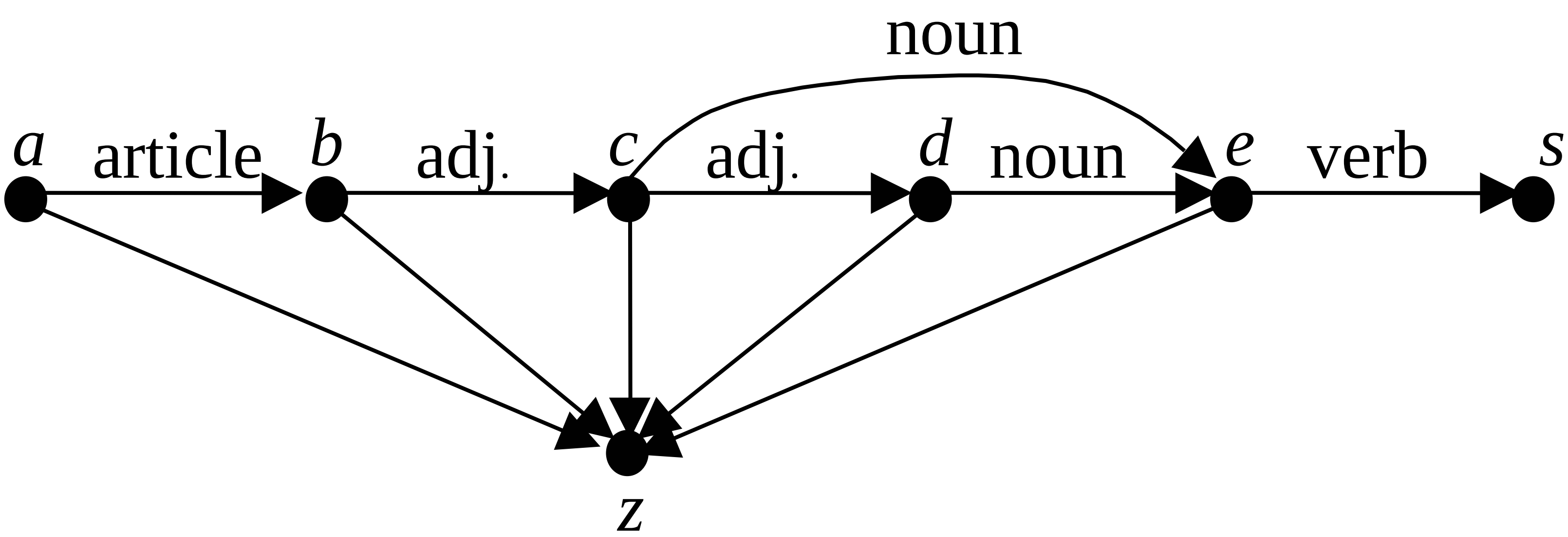




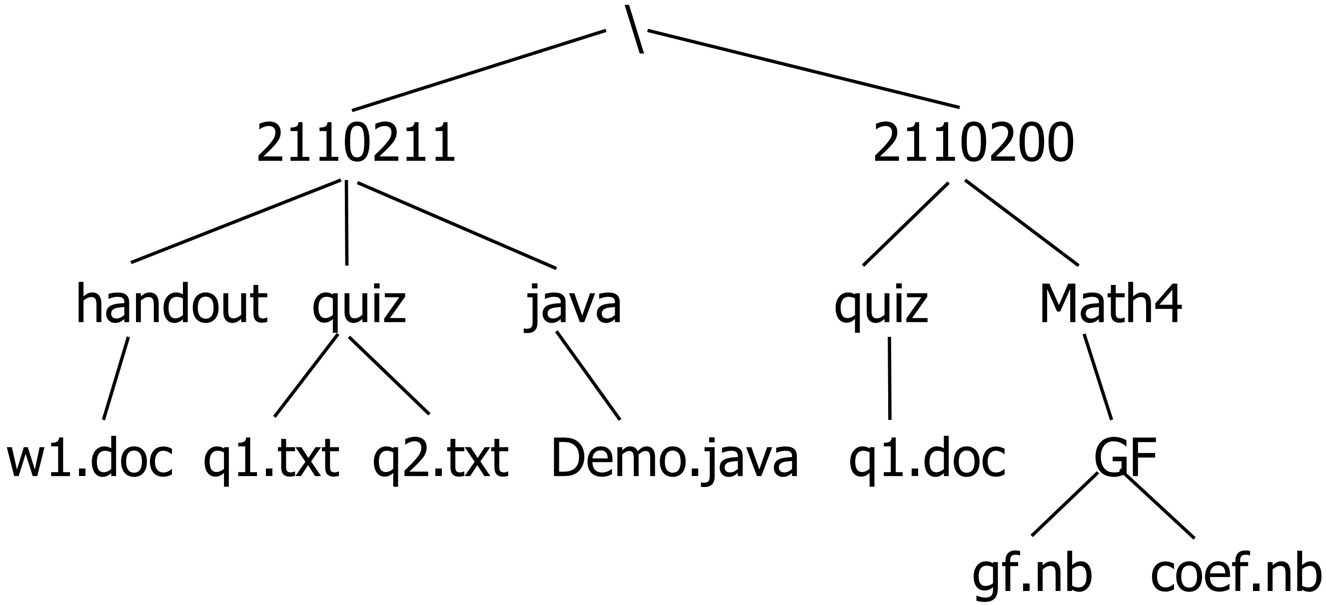
Applications

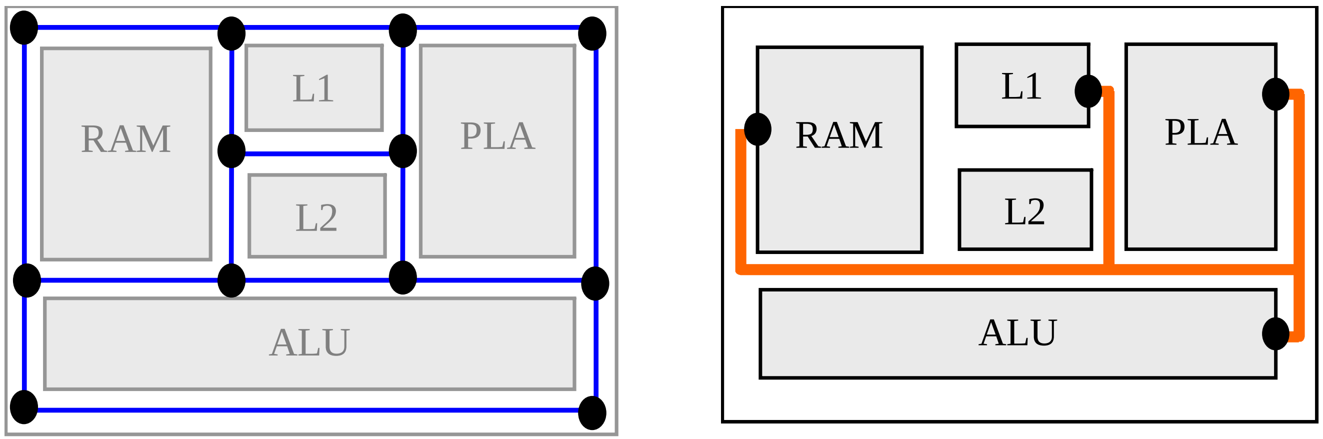




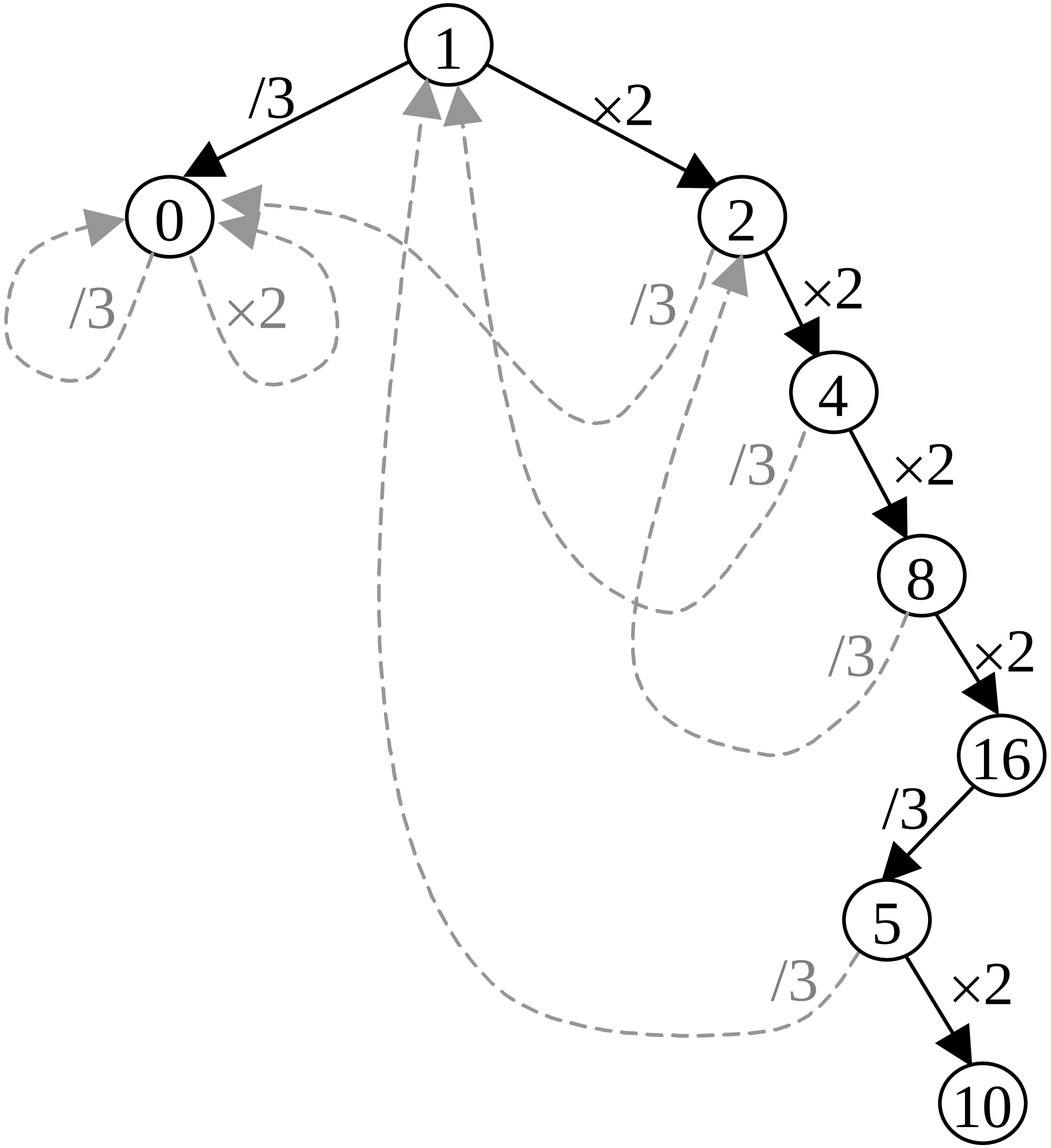


Applications

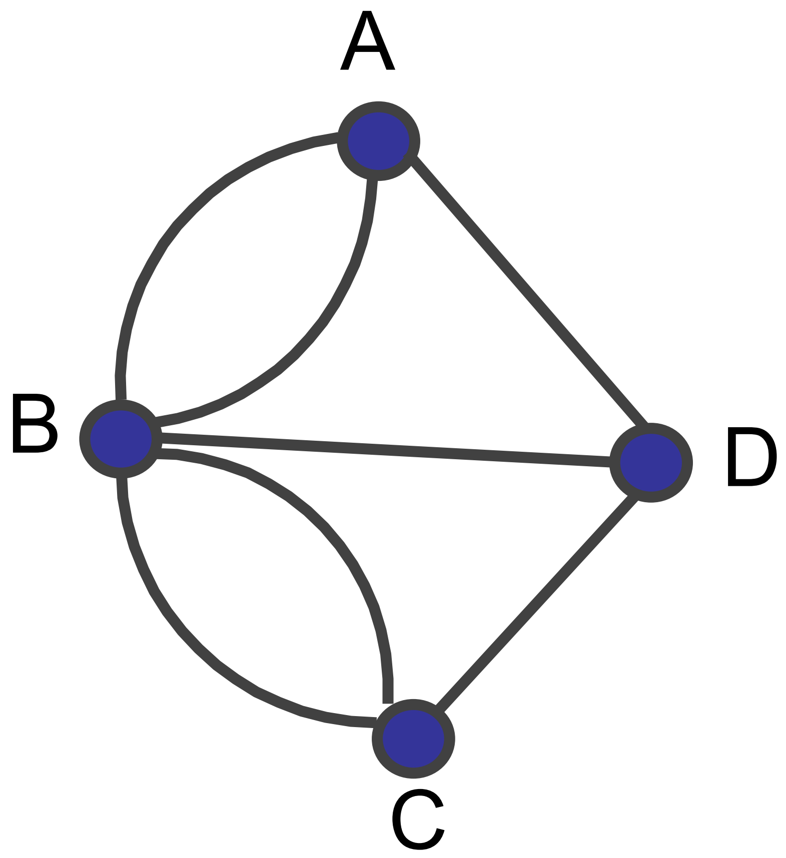


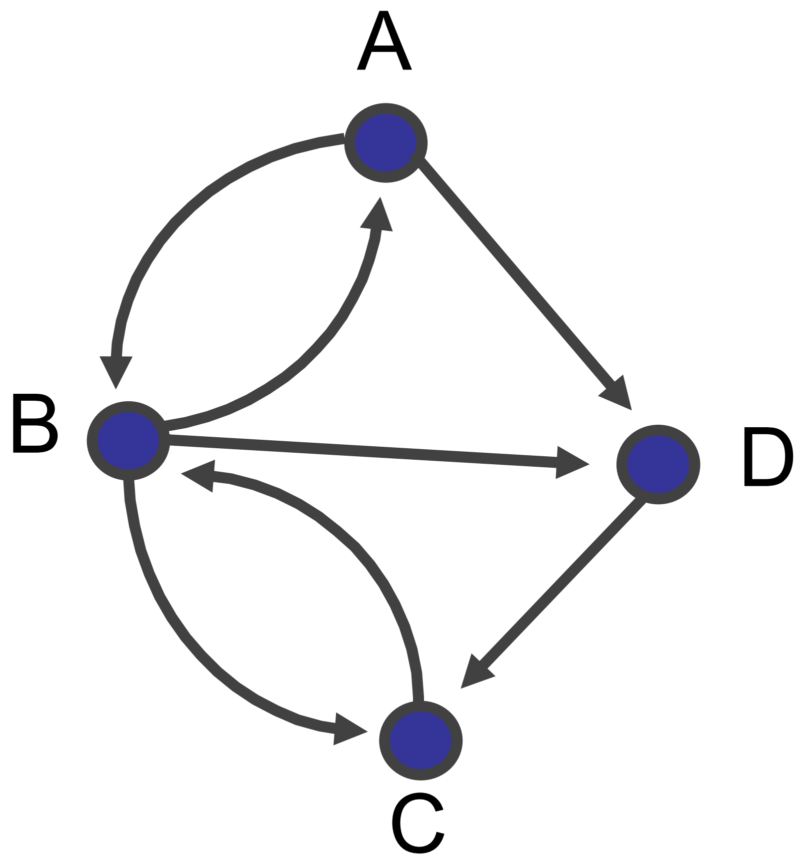


Applications



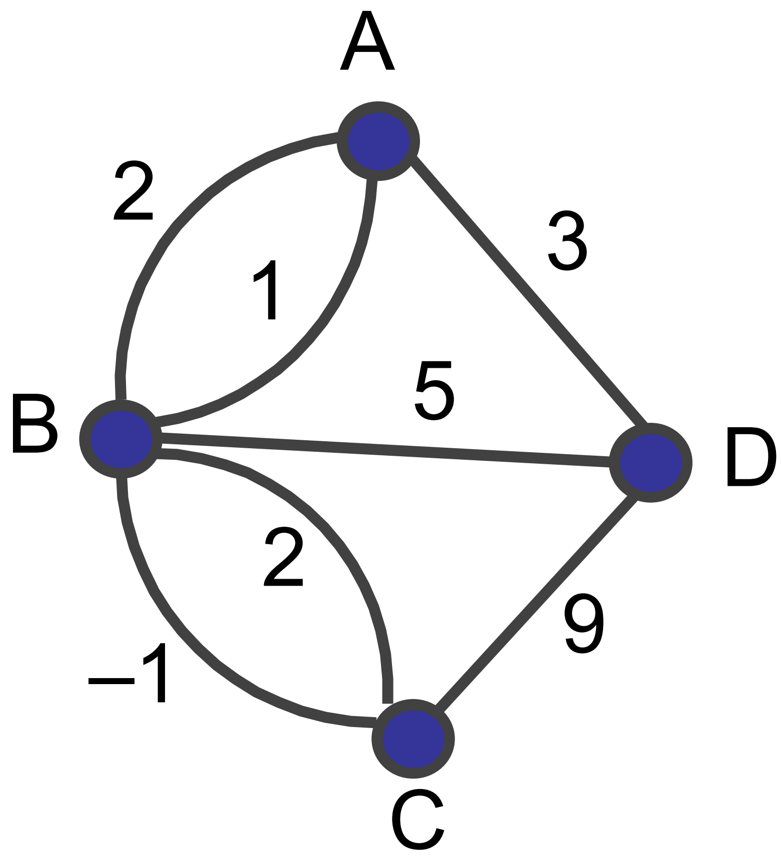
Undirected & Directed Graphs

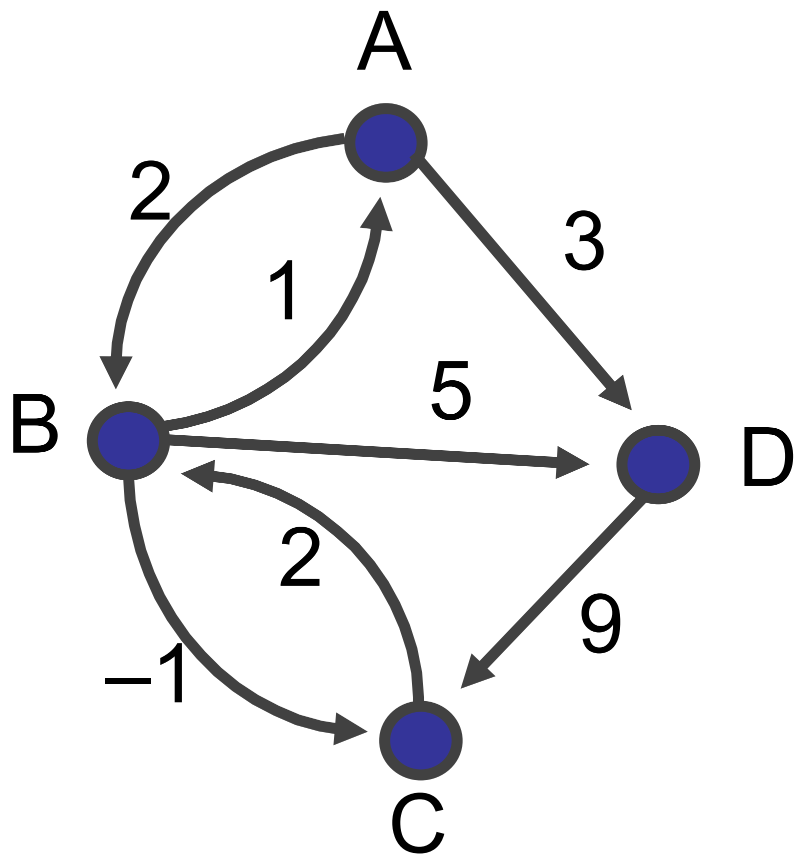




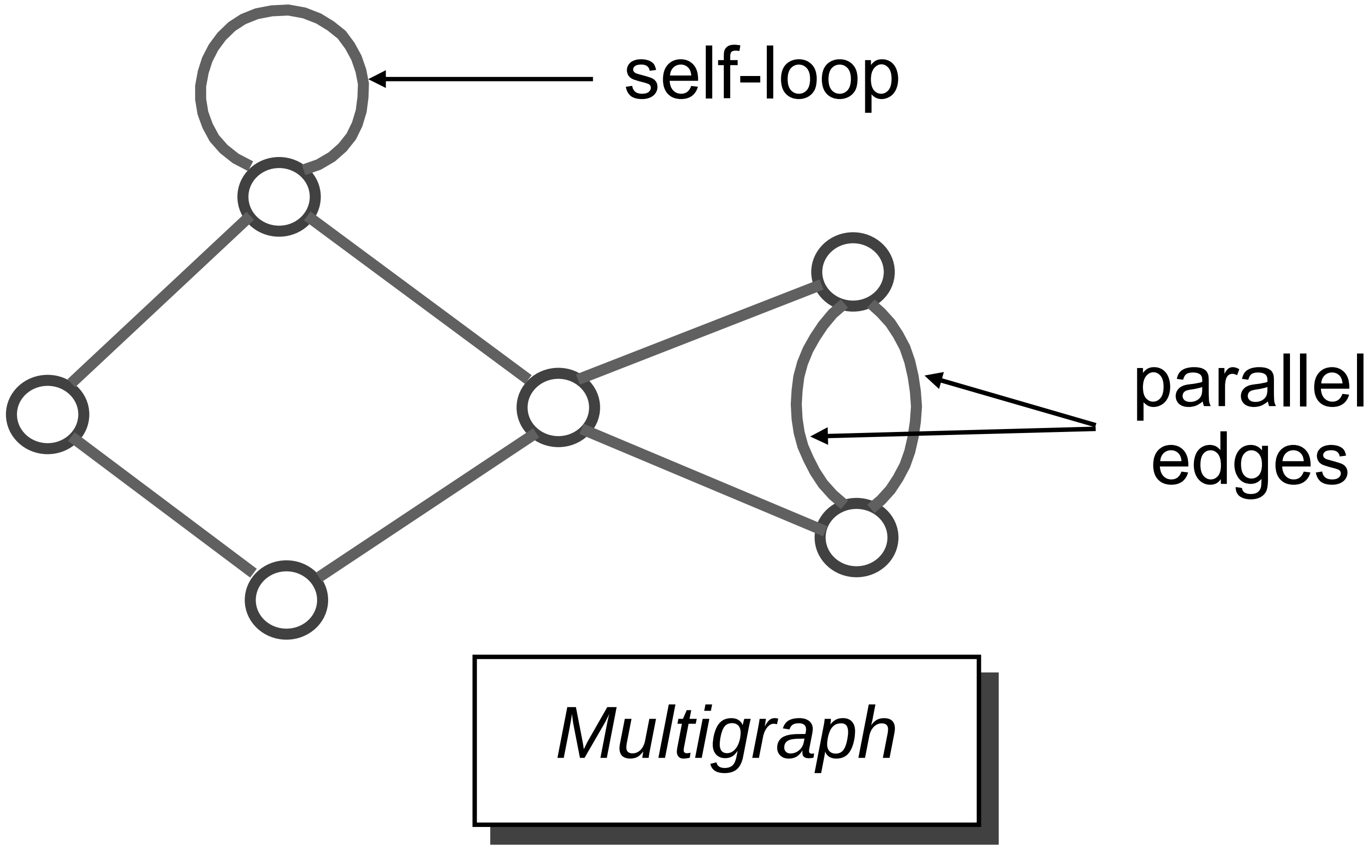
Digraph

Weighted Graphs



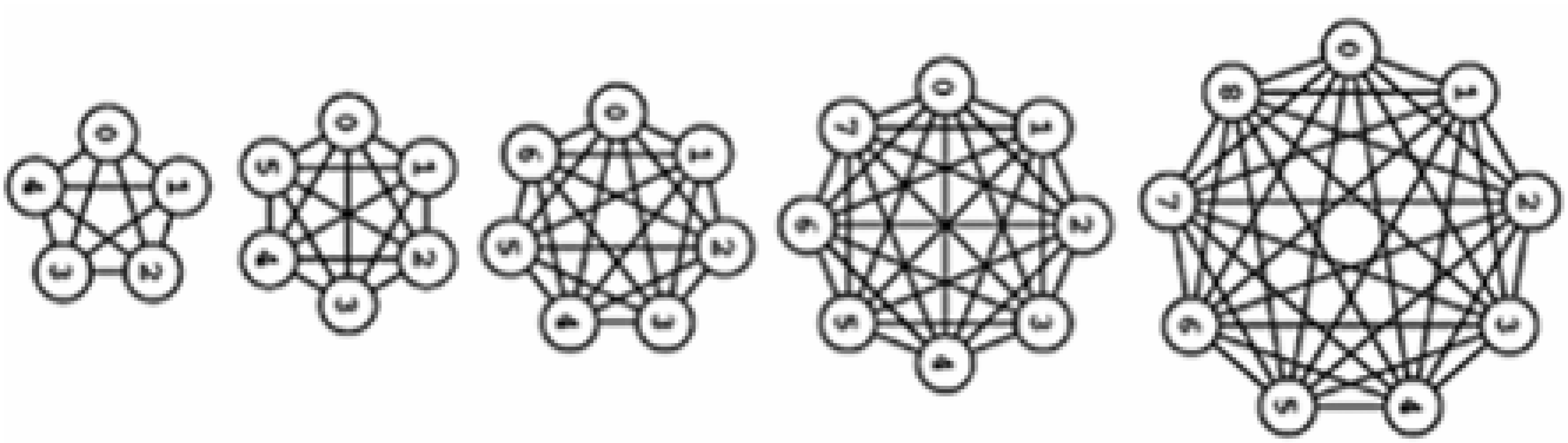


Multigraphs & Simple graphs



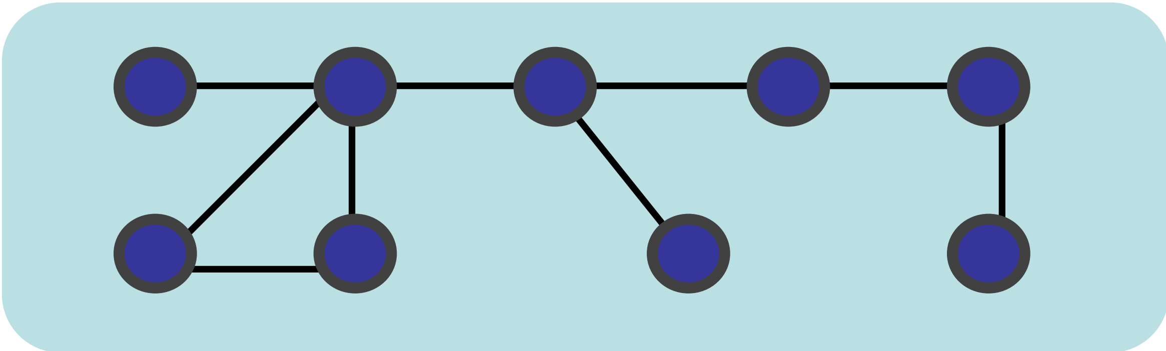
A graph that has neither self-loops nor parallel edges is called a <u>*simple*</u> graph.

Complete Graphs

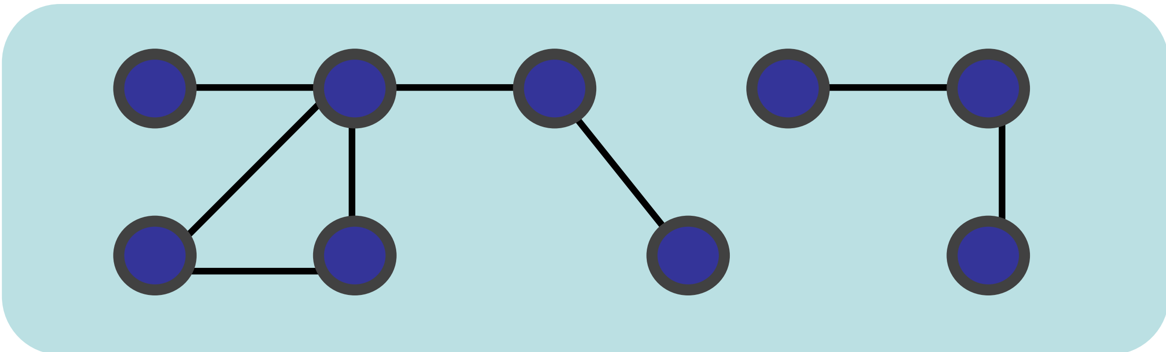


complete graph ที่มี v vertices มี $v(v-1)/2$ edges

Connected Graphs



1 component
(connected graph)

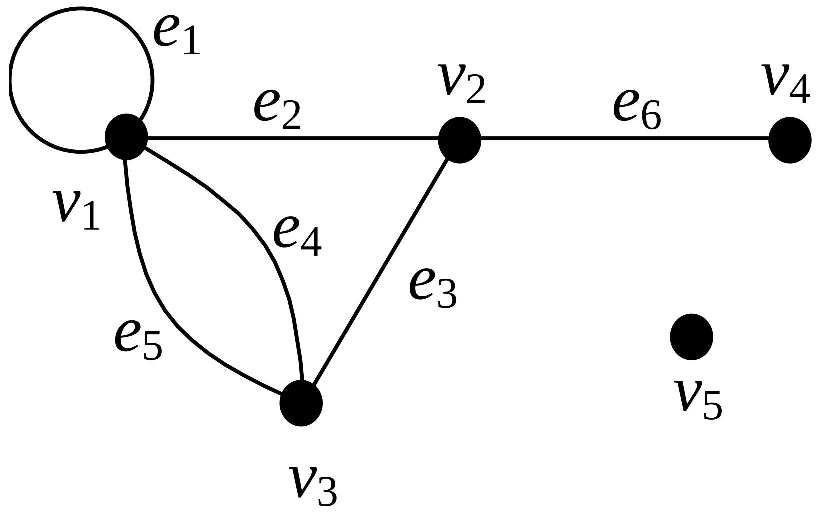


2 components

- A connected (undirected) graph with v vertices has at least $v-1$ edges
- A simple graph with v vertices and $C(v-1,2)$ edges must be connected

Degree

- e_2 is <u>incident</u> on v_2
- v_1 is <u>adjacent</u> to v_2
- <u>degree</u> of v_3 is 3



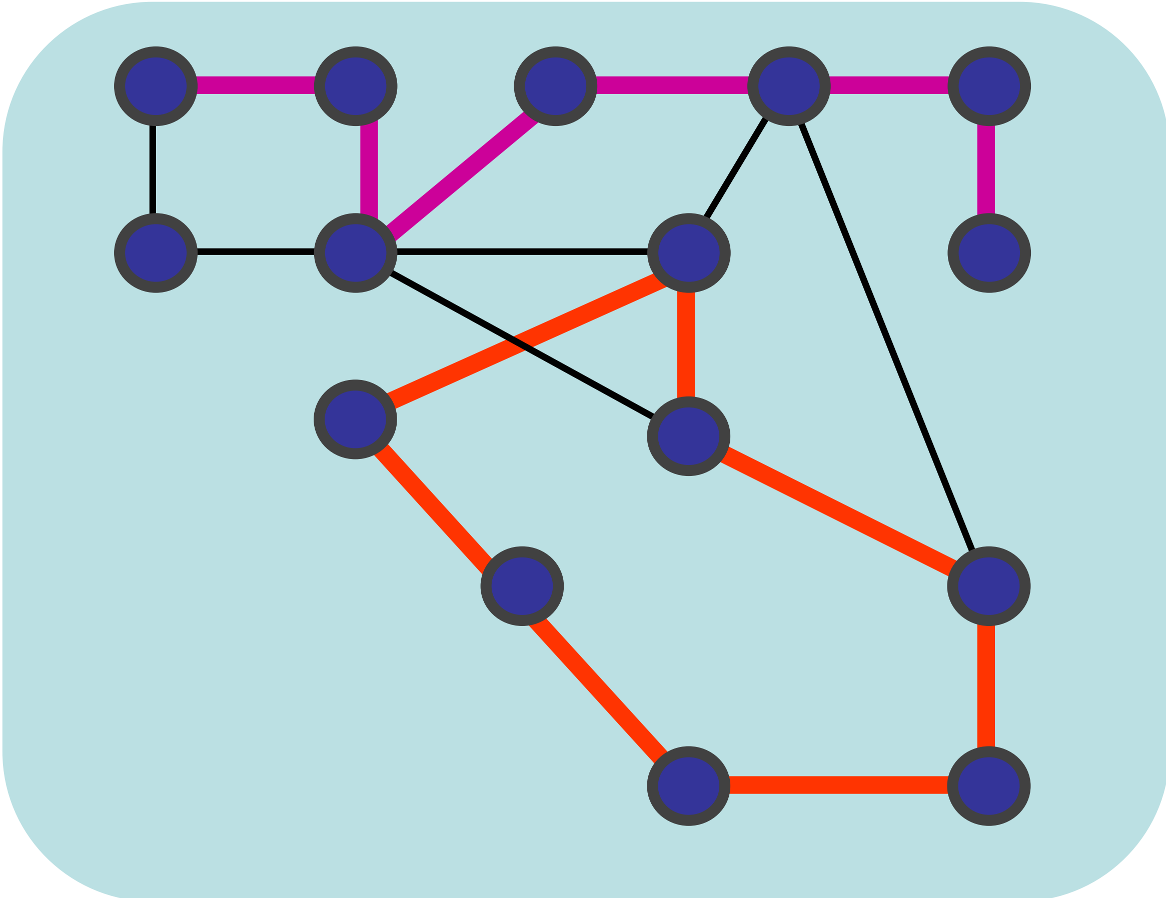
The sum of the degrees of all vertices in an undirected graph is twice the number of edges in the graph.

The number of vertices of odd degree in an undirected graph is always even.

แบบฝึกหัด

- What is the minimum number of cables needed to connect 5 computers so that all of them can exchange information ?
- Can it be concluded that a simple graph with 5 vertices and 6 edges is connected ?
- Must the number of people ever born who had (have) an odd number of brothers and sisters be even ?
- What is the largest possible number of vertices in a graph with 19 edges and all vertices of degree at least 3 ?
- Is it possible that each person at a party know 5 other persons in the party ?

Paths & Cycles

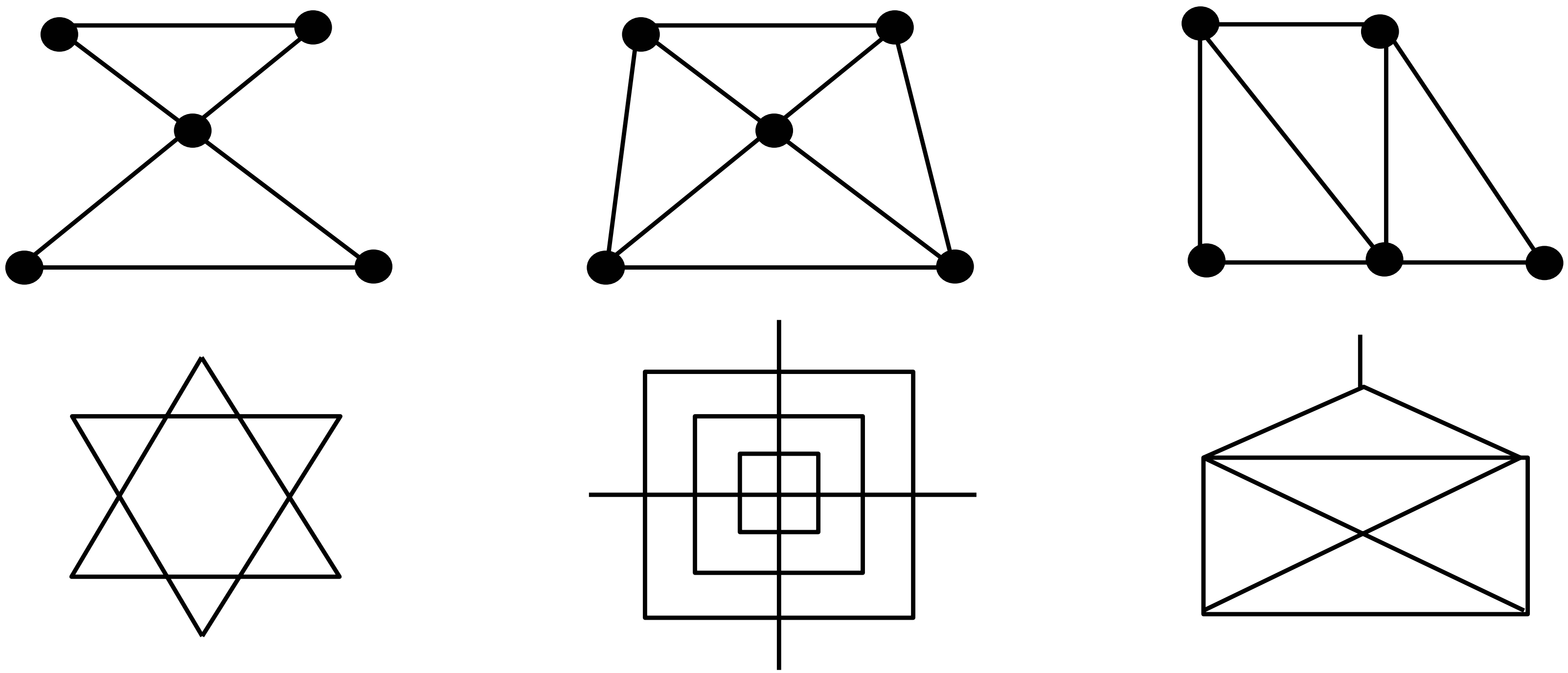


A path or circuit is ***simple*** if it passes through a vertex at most one time.

Euler Paths & Circuits

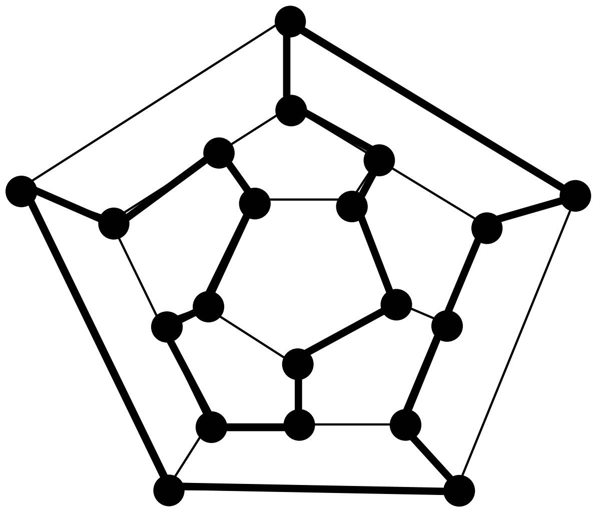
- An *Euler circuit* in a graph is a circuit that traverses all the edges in the graph once.
- An *Euler path* in a graph is a path that traverses all the edges in the graph once.
- An undirected multigraph has an Euler circuit if and only if it is connected and has all vertices of even degree.
- An undirected multigraph has an Euler path, but not Euler circuit, if and only if it is connected and has exactly two vertices of odd degree.

Euler Paths & Circuits



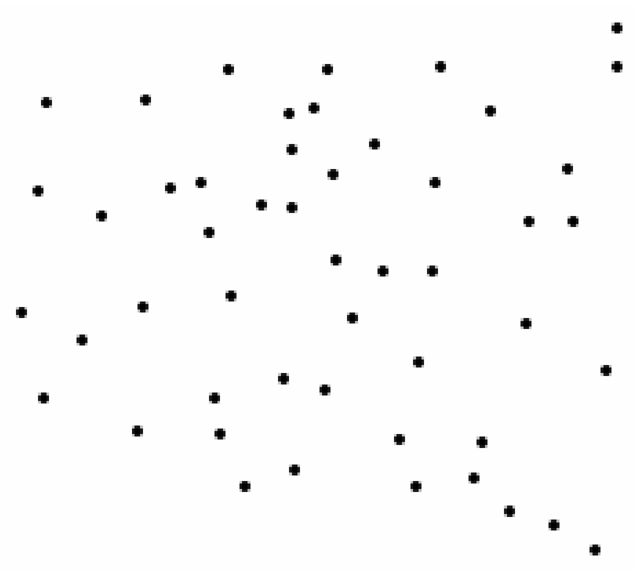
Hamilton Paths & Circuits

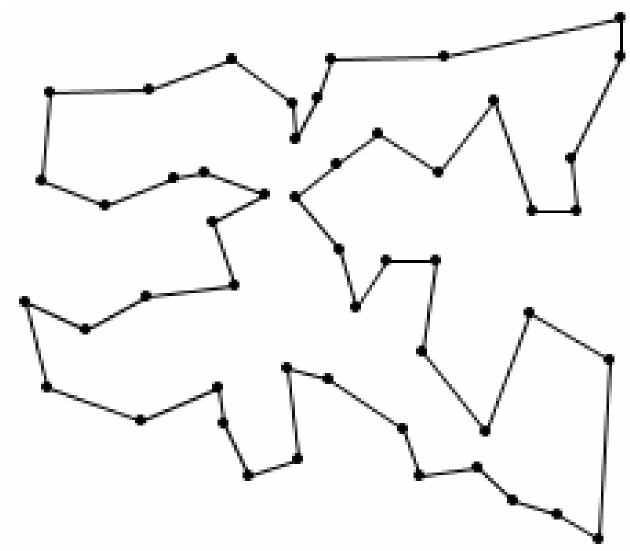
- A *Hamilton circuit* in a graph is a (simple) circuit that visits each vertex in the graph once.
- A *Hamilton path* in a graph is a (simple) path that visits each vertex in the graph once



Traveling Salesperson Problem

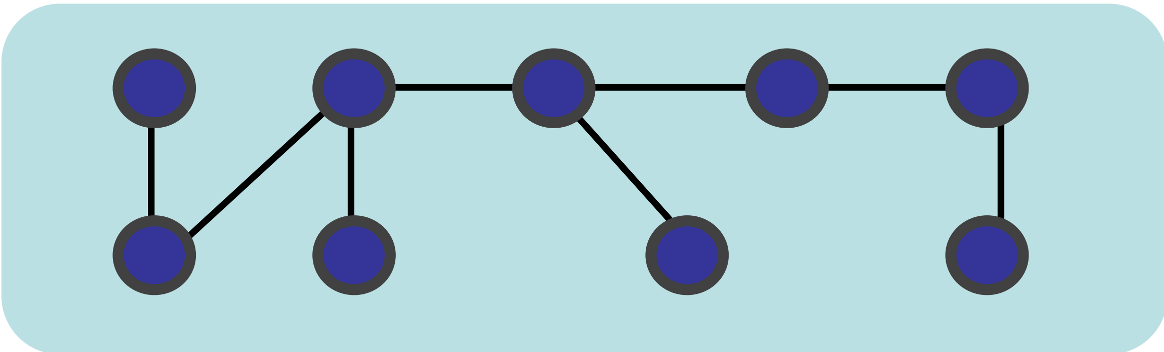
- A salesman is required to visit a number of cities during a trip. Given the distances between cities, in what order should he travel so as to visit every city precisely once and return home, with the *minimum* mileage traveled ?





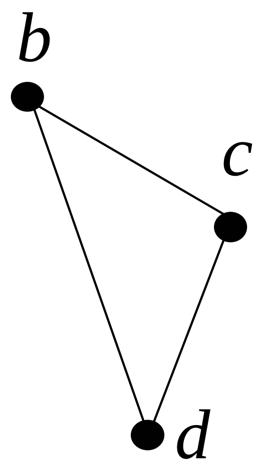
Trees

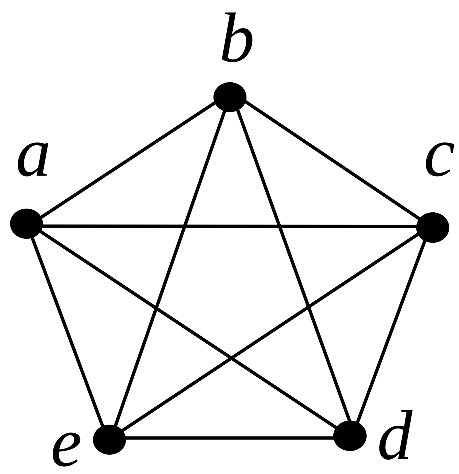
- a acyclic connected graph
- v vertices, v – 1 edges, no cycle
- v vertices, v – 1 edges, connected
- exactly one simple path connects each pair of vertices



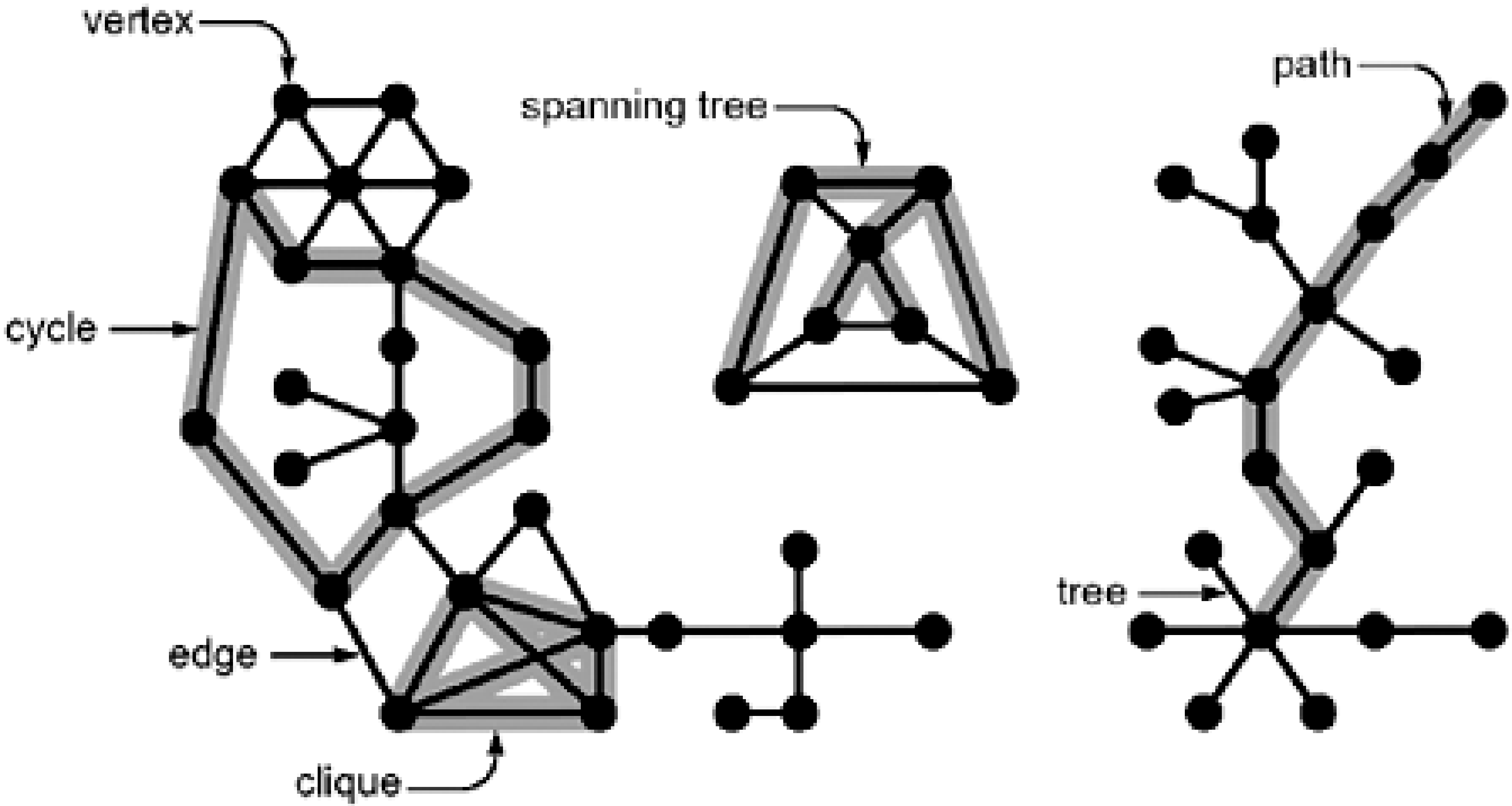
Subgraphs

- A *subgraph* is a subset of a graph's edges (and associated vertices) that constitutes a graph



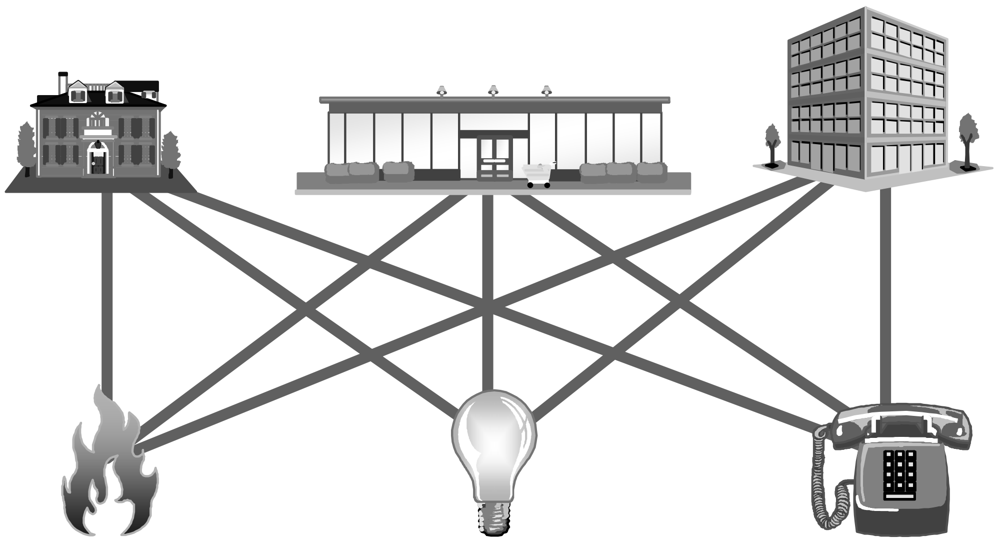


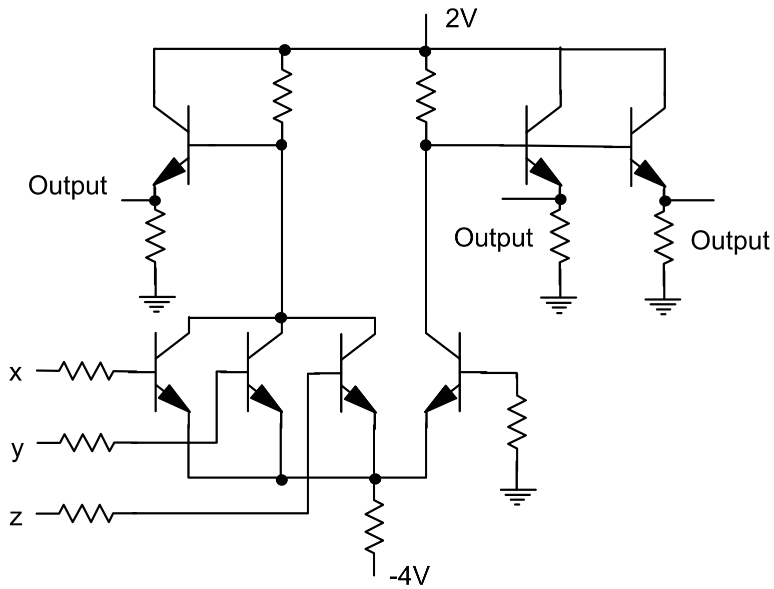
Terminology



Planar Graphs

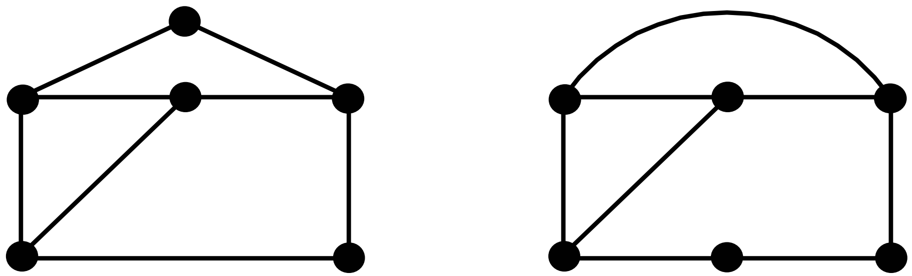
A graph is called *planar* if it can be drawn in the plane without any edges crossing.





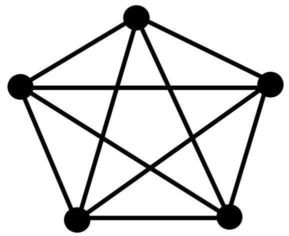
Homeomorphic Gaphs

Two graphs are called <u>*homeomorphic*</u> if one graph can be obtained from the other by the creation of edges in series or by the merger of edges in series.

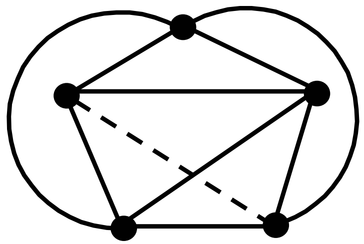


A graph *G* is planar if and only if every graph that is homeographic to *G* is planar.

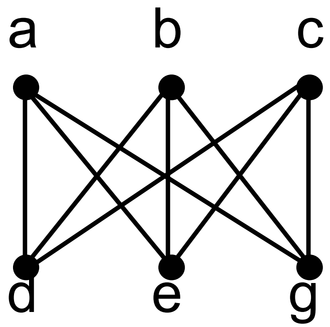
Kuratowski Graphs



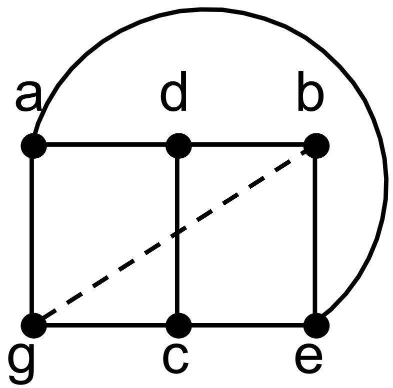
K_5



$|V| = 5, \ |E| = 10$



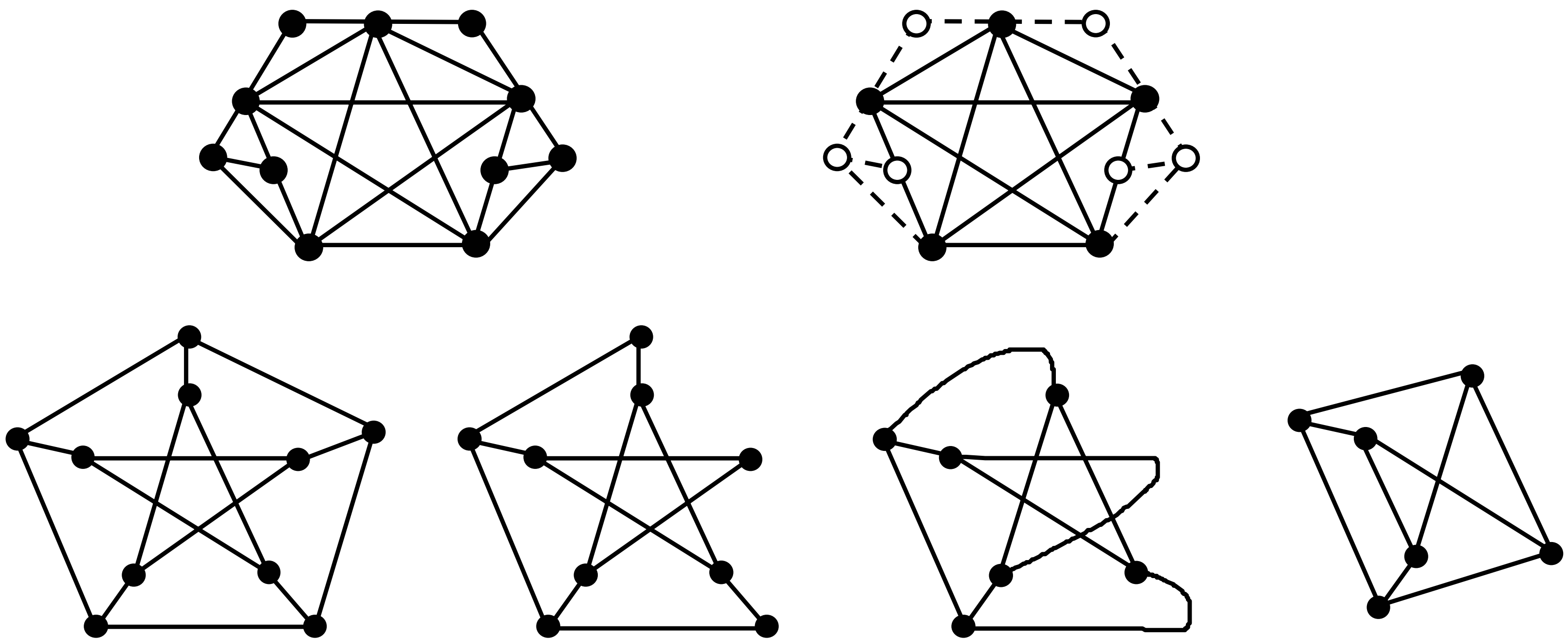
$K_{3,3}$



$|V| = 6, \ |E| = 9$

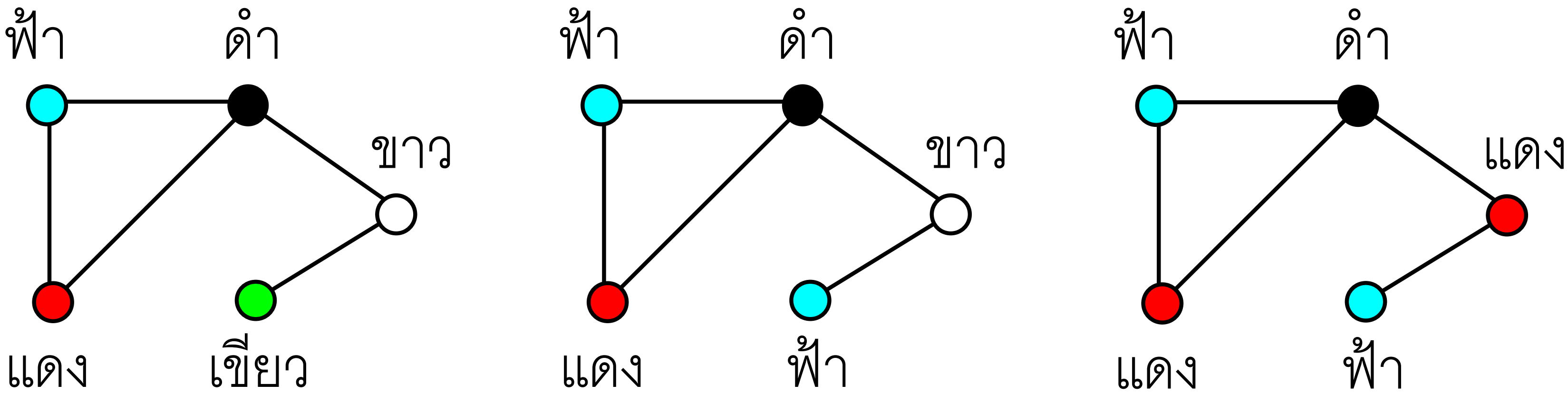
Kuratowski's Theorem

A graph is nonplanar if and only if it contains a subgraph homeomorphic to K_5 or $K_{3,3}$



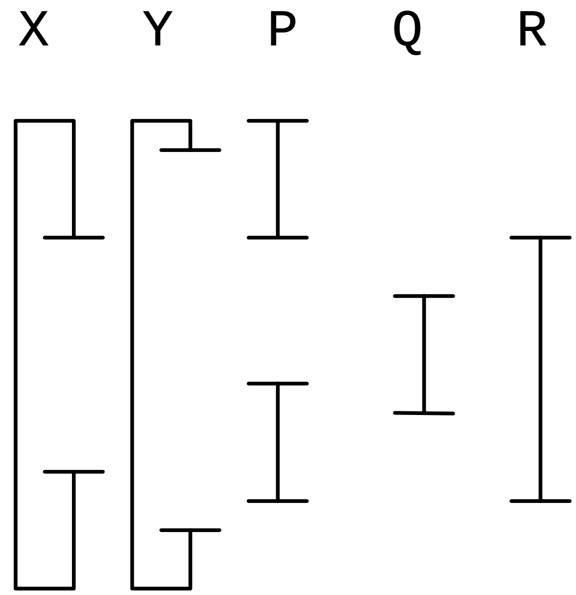
Graph Coloring

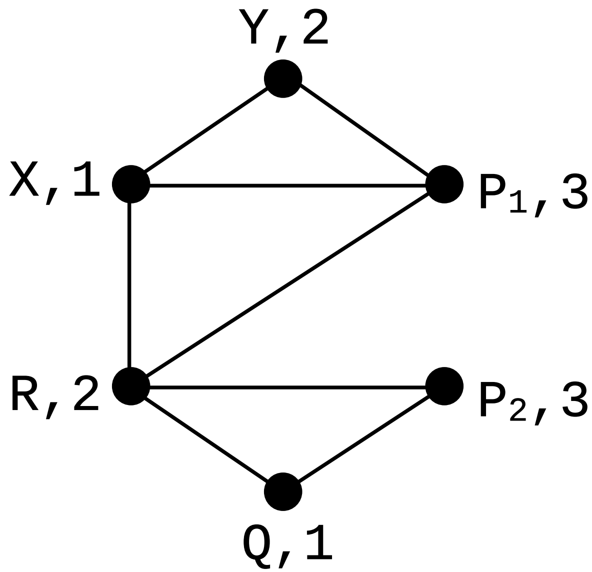
A *coloring* of a simple graph is the assignment of a color to each vertex of the graph so that no two adjacent vertices are assigned the same color.



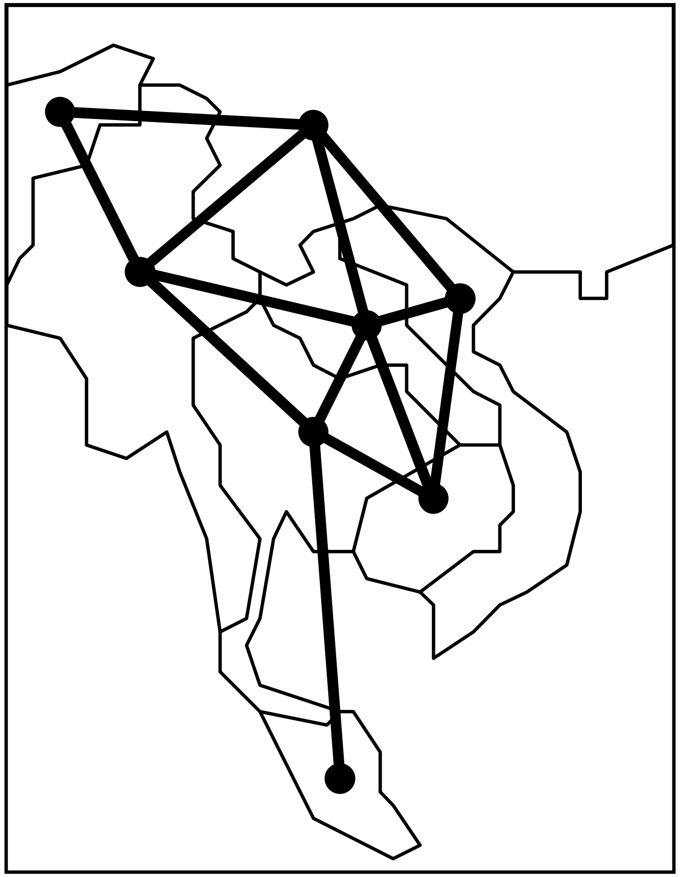
Graph Coloring

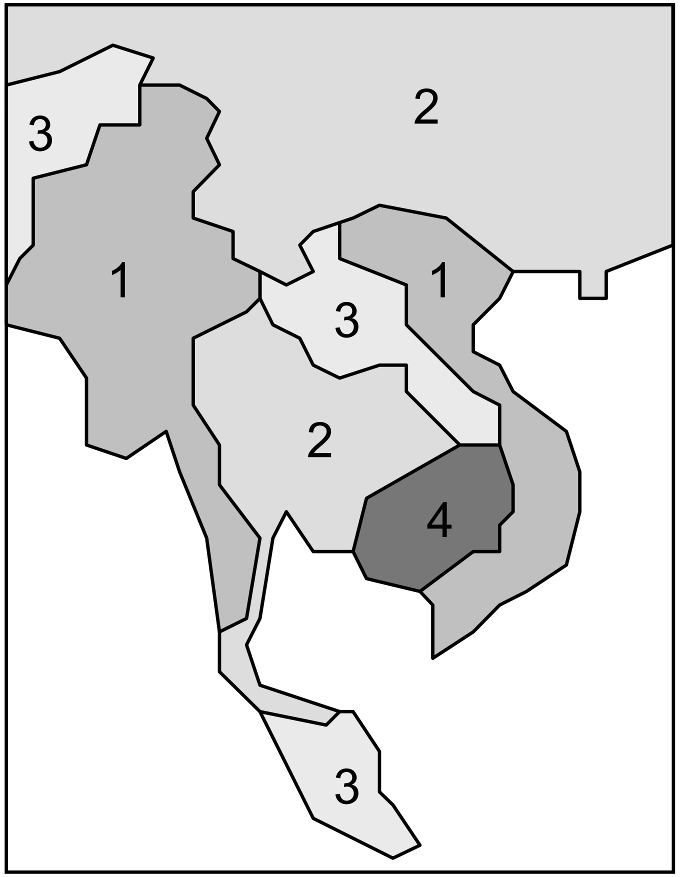
```
1: P := X + Y
2: X := X * P
3: Q := 1/R
4: P := R - Q
5: X := R/P
6: Y := 0.5
```





Four-Color Theorem

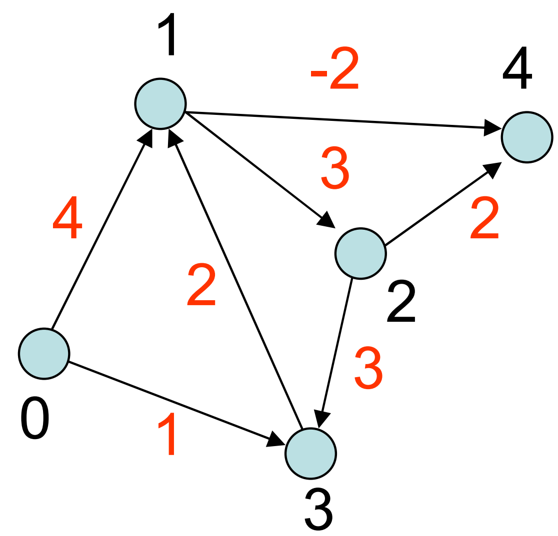




Graph Representations

- adjacency matrix

	0	1	2	3	4
0	-	4	-	1	-
1	-	-	3	-	-2
2	-	-	-	3	2
3	-	2	-	-	-
4	-	-	-	-	-

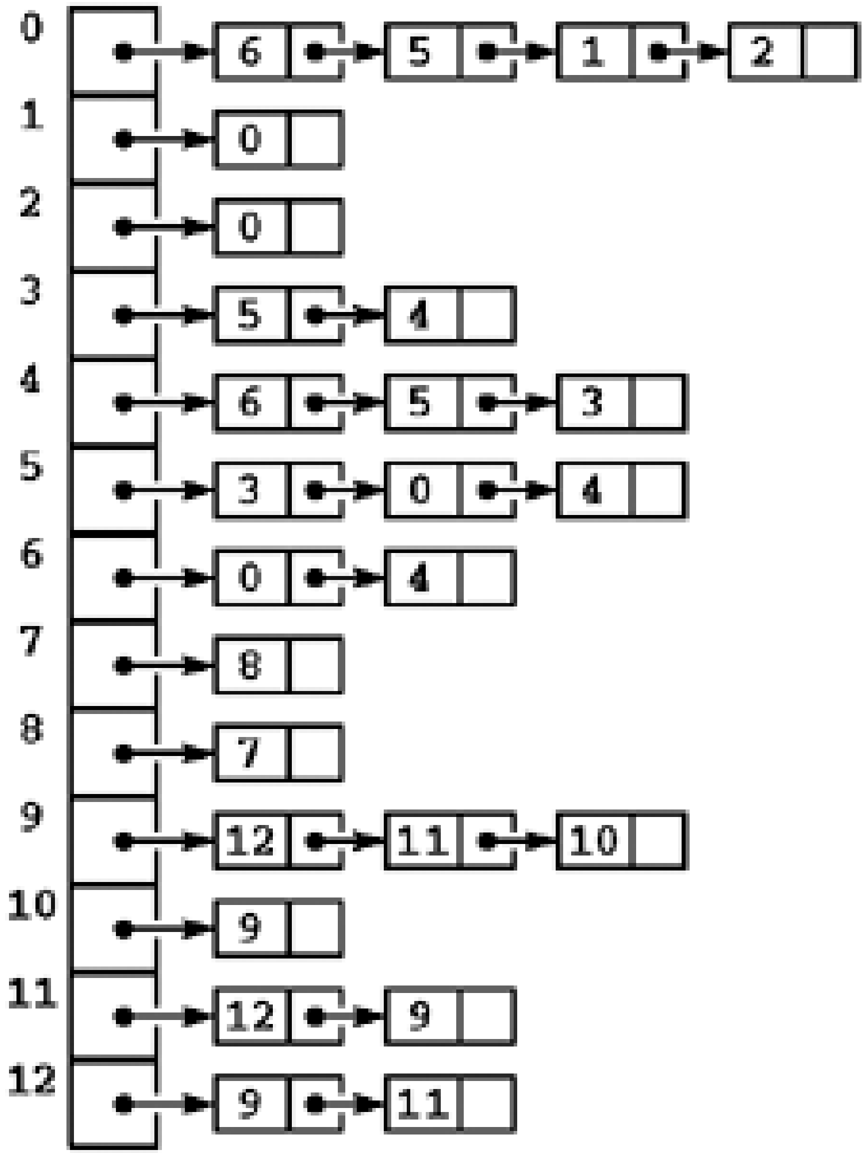


- adjacency list

```
0 → < (1,4), (3,1) >
1 → < (2,3), (4,-2) >
2 → < (3,3), (4,2) >
3 → < (1,2) >
4 → < >
```

Graph Representations

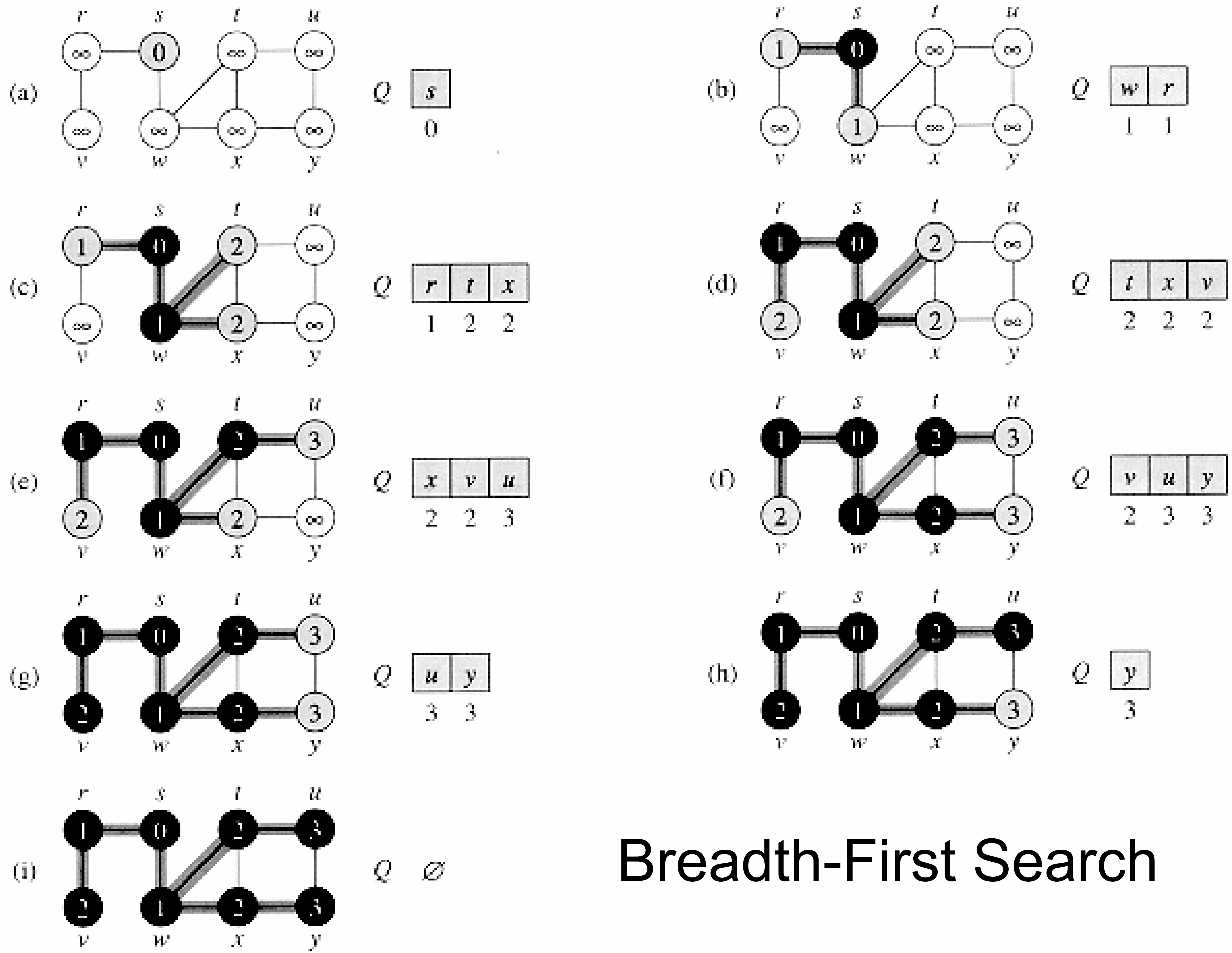
	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	1	0	0	1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	1	0	0	0	0	0	0	0
4	0	0	0	1	0	1	1	0	0	0	0	0	0
5	1	0	0	1	1	0	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	1
10	0	0	0	0	0	0	0	0	0	1	0	0	0
11	0	0	0	0	0	0	0	0	0	1	0	0	1
12	0	0	0	0	0	0	0	0	0	1	0	1	0



Basic Graph Algorithms

- Breadth-First Search
- Depth-First Search
- Topological Sort
- Strongly Connected Components

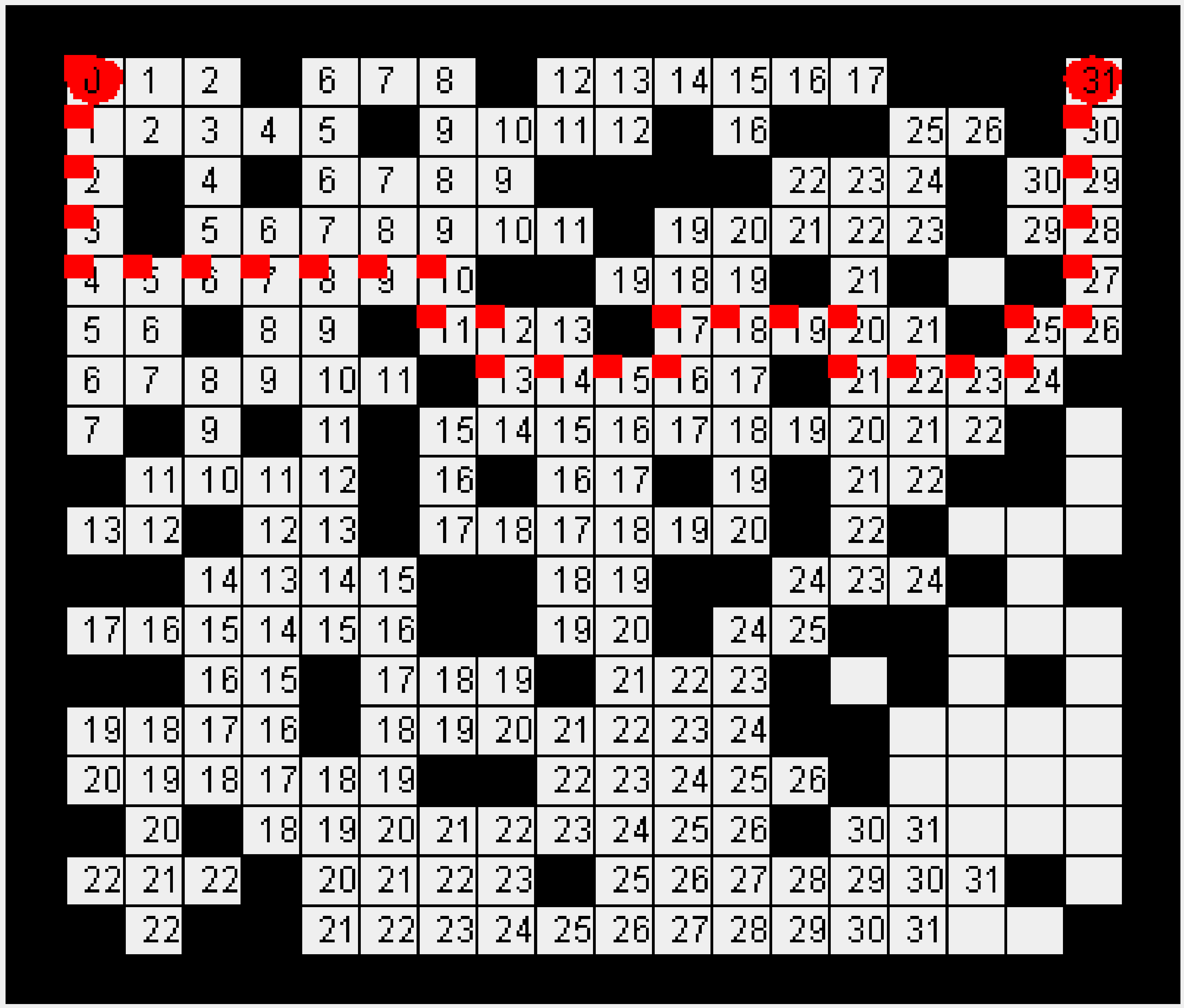
Breadth-First Search



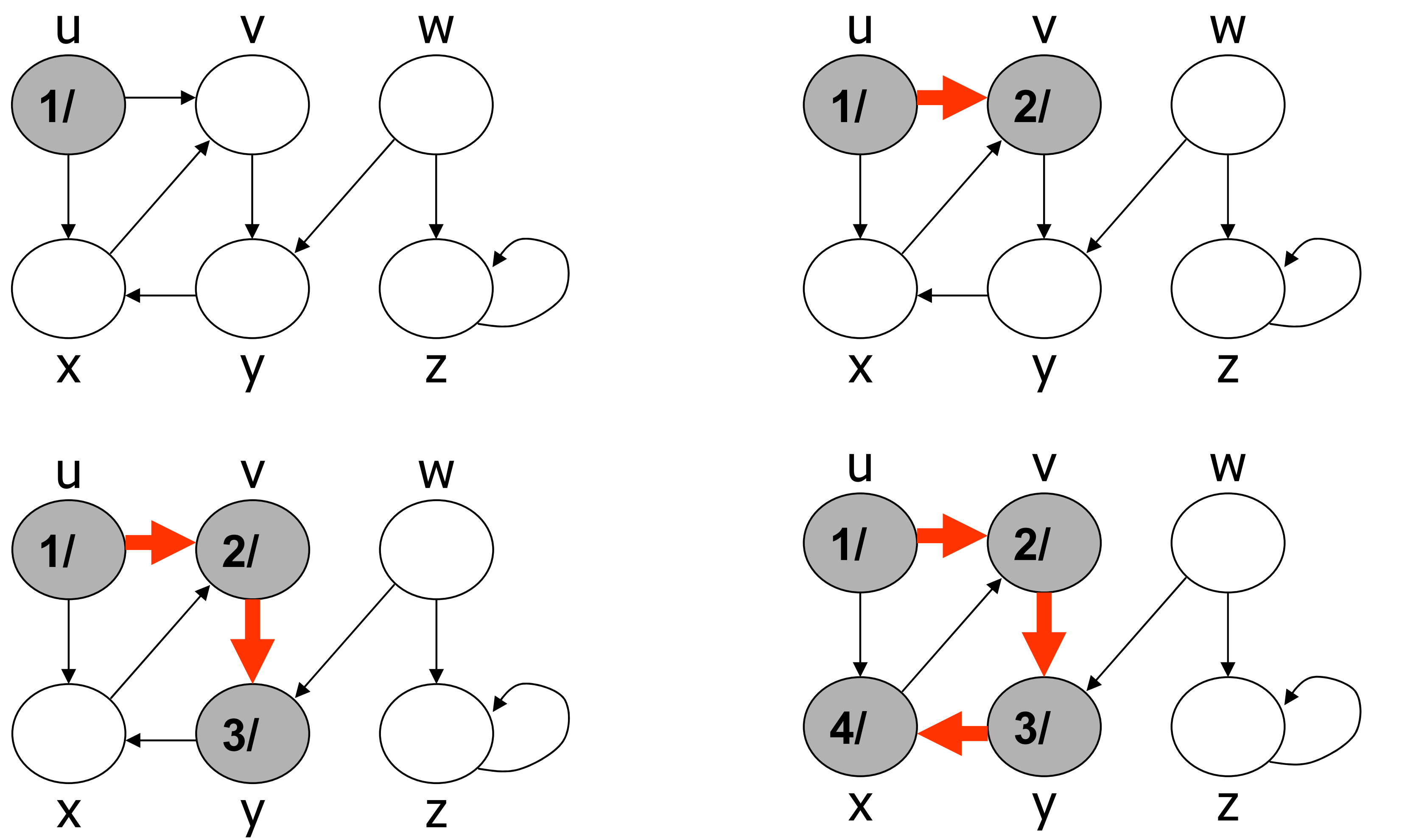
```
BFS(G, s)
  for each vertex u ∈  V[G] - {s}
    do color[u] ← WHITE
       d[u] ← ∞
       p[u] ← NIL
   color[s] ← GRAY
   d[s] ← 0
   p[s] ← NIL
   Q ← Ø
   ENQUEUE(Q, s)
   while Q ≠ Ø
     do u ← DEQUEUE(Q)
       for each v ∈  Adj[u]
         do if color[v] = WHITE
              then color[v] ← GRAY
                   d[v] ← d[u] + 1
                   p[v] ← u
                   ENQUEUE(Q, v)
       color[u] ← BLACK
```

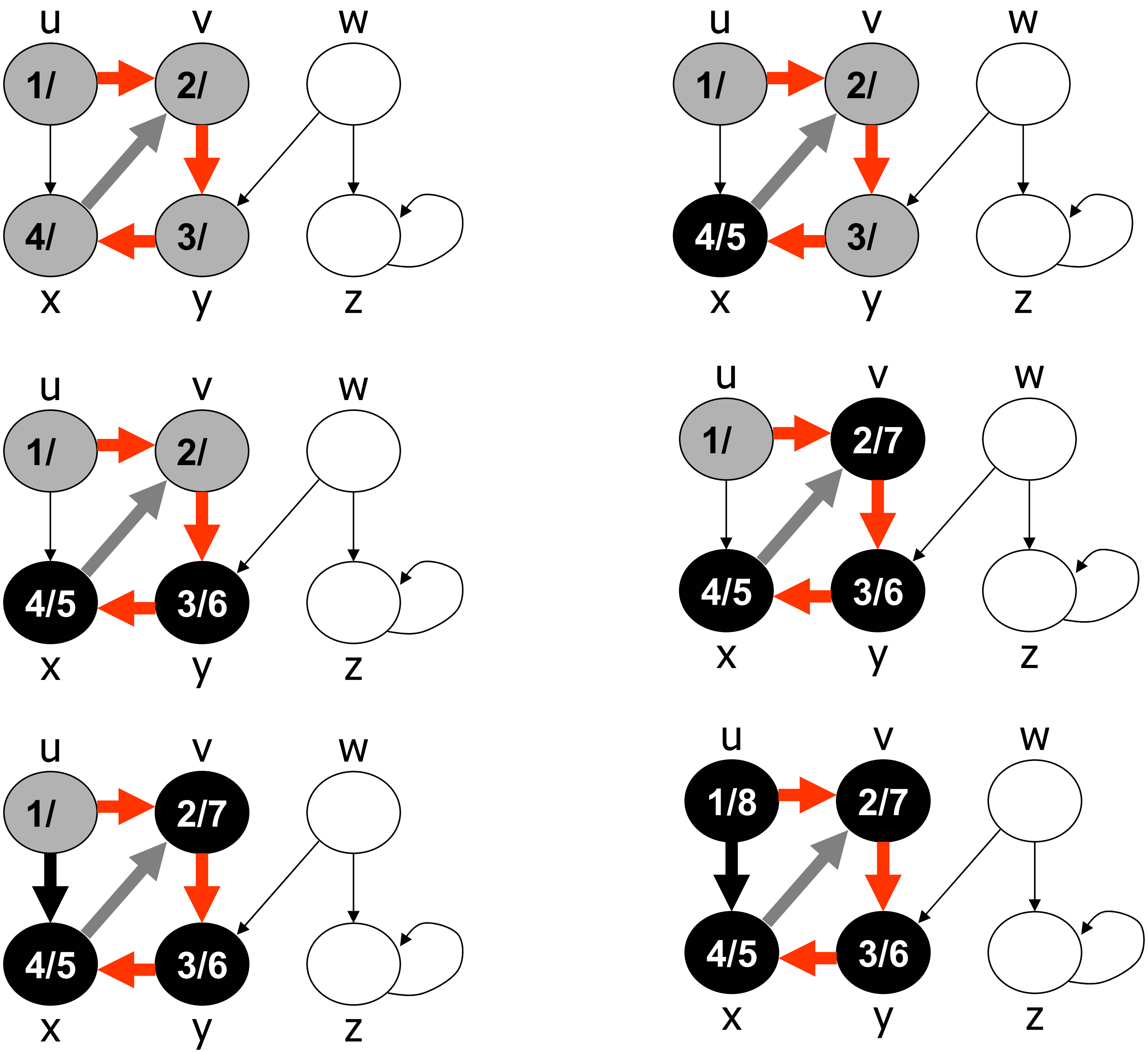
```
PRINT-PATH(G, s, v)
  if v = s then print s
  else if p[v] = NIL then "no path"
  else PRINT-PATH(G, s, p[v])
       print v
```

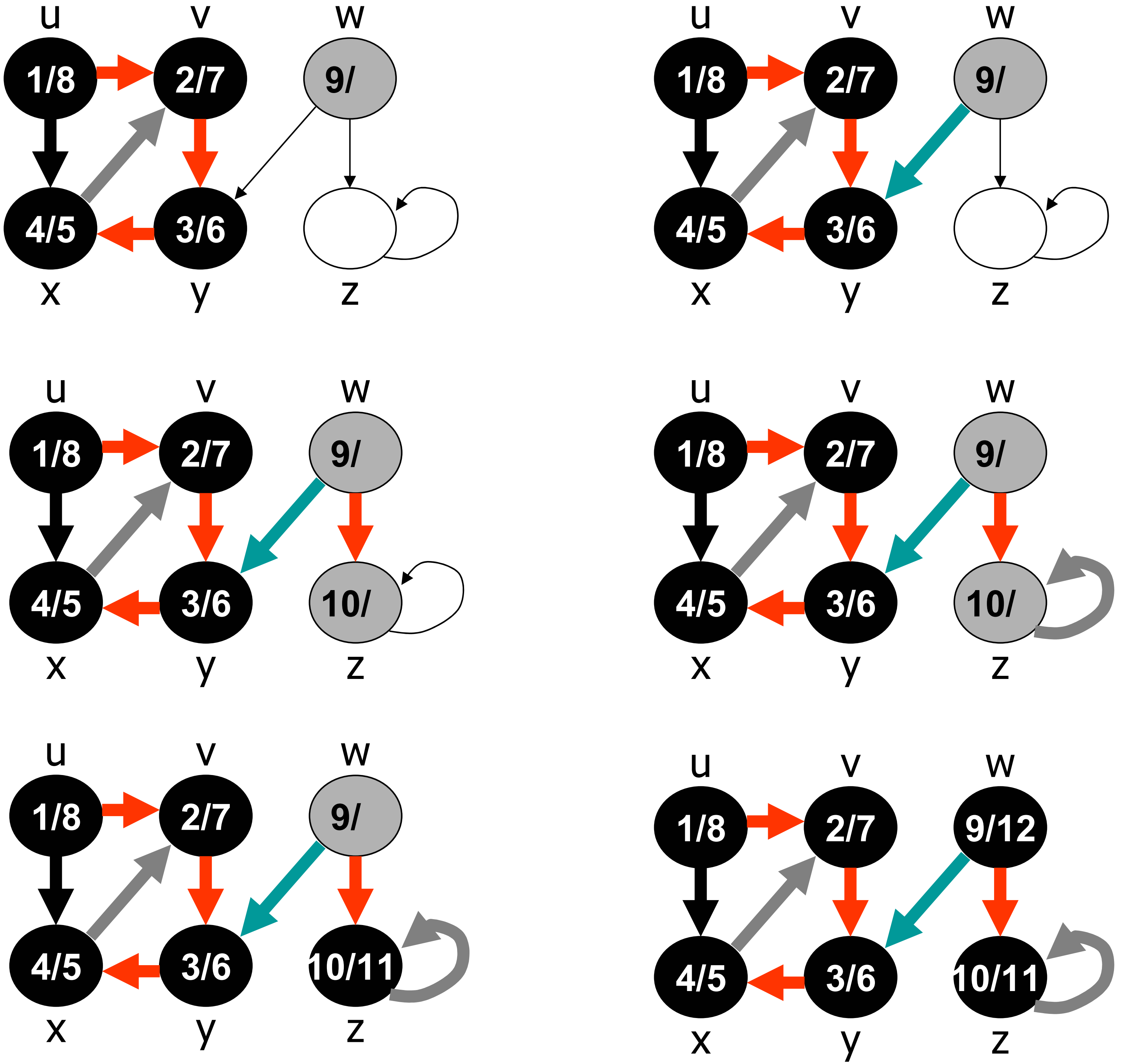
Shortest Path



Depth-First Search




u v w
1/ 2/
4/ 3/
x y z
u v w
1/ 2/
4/5 3/
x y z
u v w
1/ 2/
4/5 3/6
x y z
u v w
1/ 2/7
4/5 3/6
x y z
u v w
1/ 2/7
4/5 3/6
x y z
u v w
1/8 2/7
4/5 3/6
x y z


u
v
w
1/8
2/7
9/
4/5
3/6
x
y
z
u
v
w
1/8
2/7
9/
4/5
3/6
x
y
z
u
v
w
1/8
2/7
9/
4/5
3/6
10/
x
y
z
u
v
w
1/8
2/7
9/
4/5
3/6
10/
x
y
z
u
v
w
1/8
2/7
9/
4/5
3/6
10/11
x
y
z
u
v
w
1/8
2/7
9/12
4/5
3/6
10/11
x
y
z

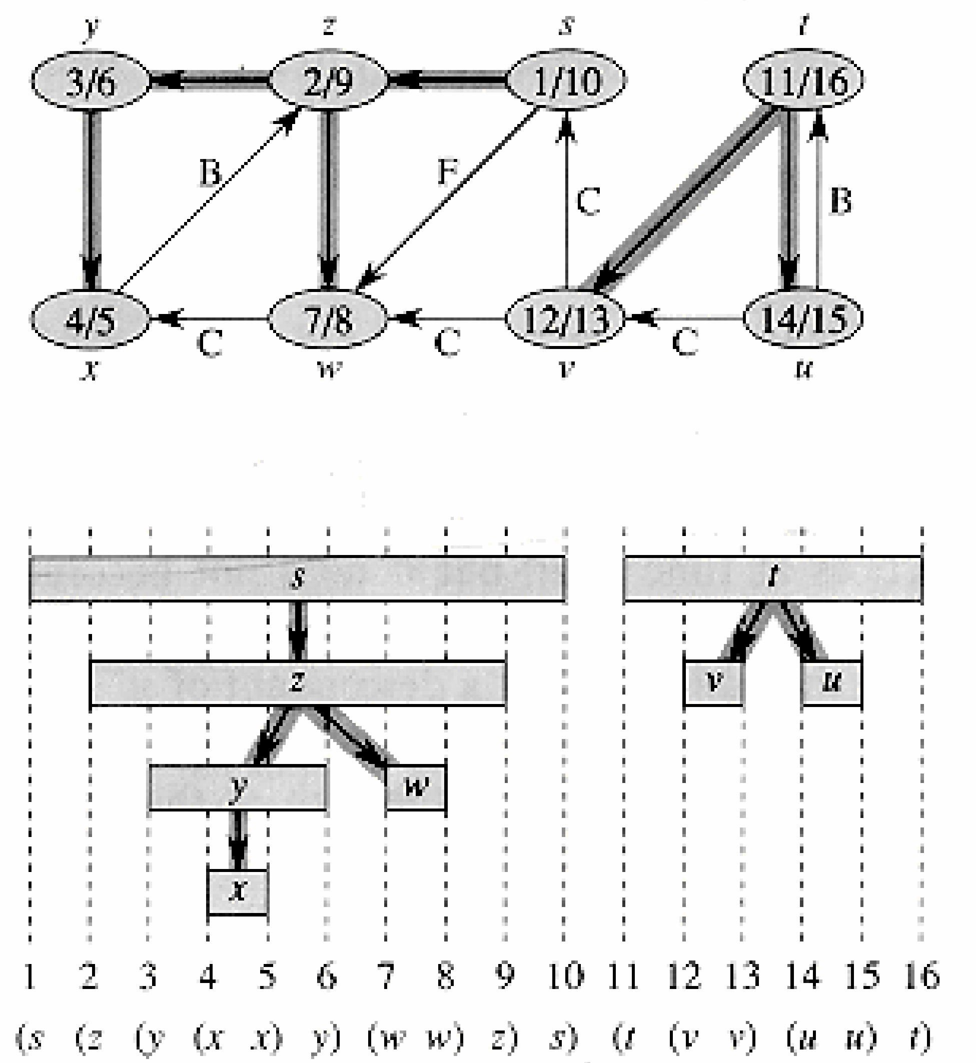
```
DFS(G)
  for each vertex u ∈ V[G]
    do color[u] ← WHITE
       p[u] ← NIL
  time ← 0
  for each vertex u ∈ V[G]
    do if color[u] = WHITE
       then DFS-VISIT(u)

DFS-VISIT(u)
  color[u] ← GRAY
  d[u] time ← time + 1
  for each v ∈ Adj[u]
     do if color[v] = WHITE
        then p[v] ← u
             DFS-VISIT(v)
 color[u] ← BLACK
 f [u] ← time ← time +1
```

tree edge (เจอขาว)

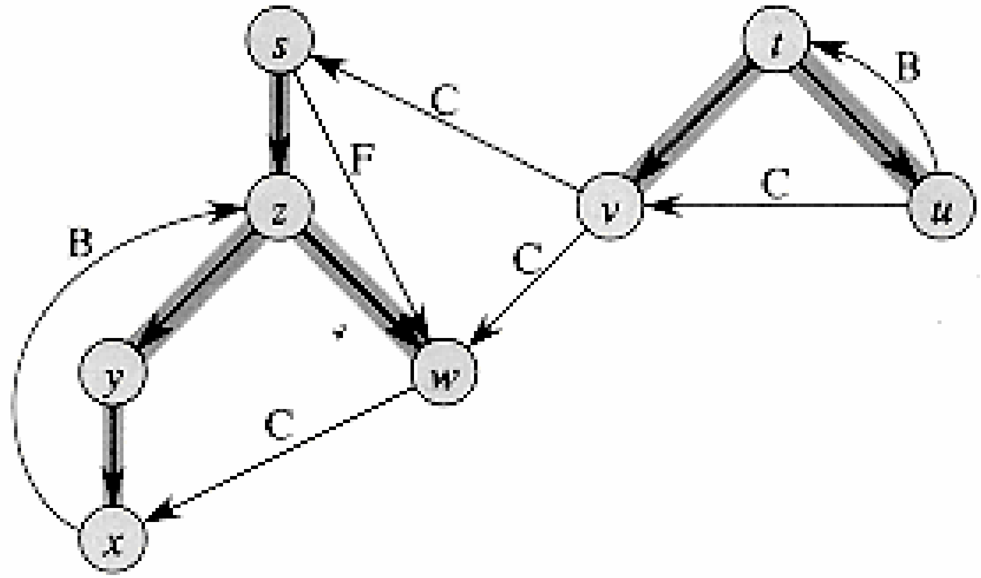
back edge (เจอเทา)

forward edge (เจอดำ d เราน้อย)

cross edge (เจอดำ d เรามาก)

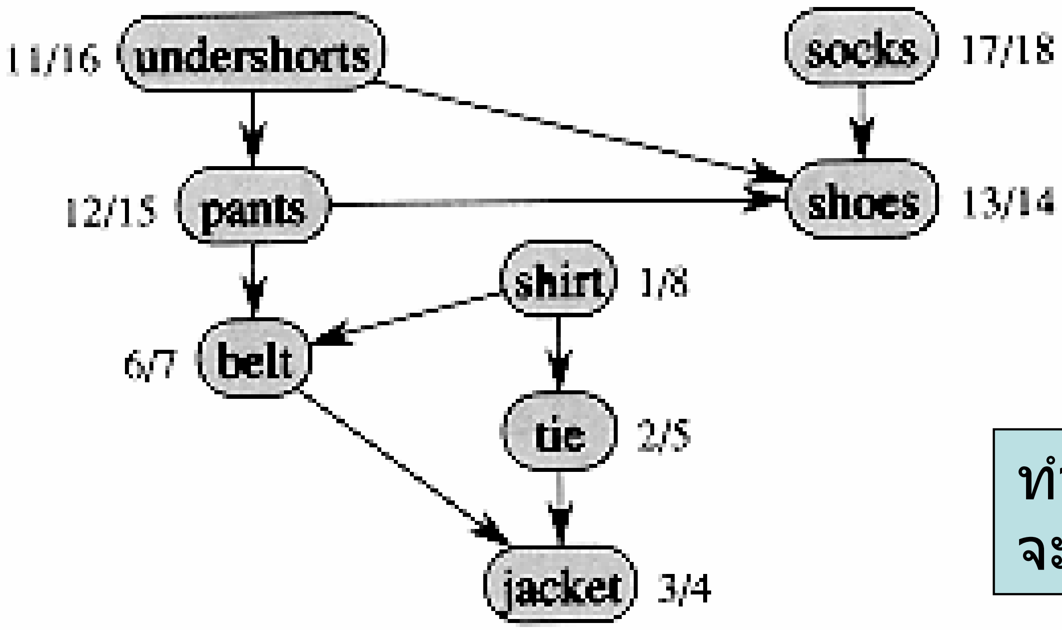


y
z
s
t
3/6
2/9
1/10
11/16
B
F
C
B
4/5
7/8
12/13
14/15
C
C
C
x
w
v
u
s
t
z
v
u
y
w
x
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
(s (z (y (x x) y) (w w) z) s) (t (v v) (u u) t)

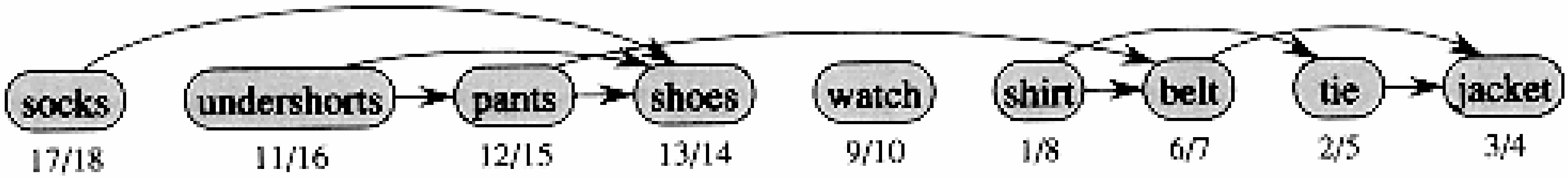


s
t
C
B
F
z
C
v
u
B
C
y
w
C
x

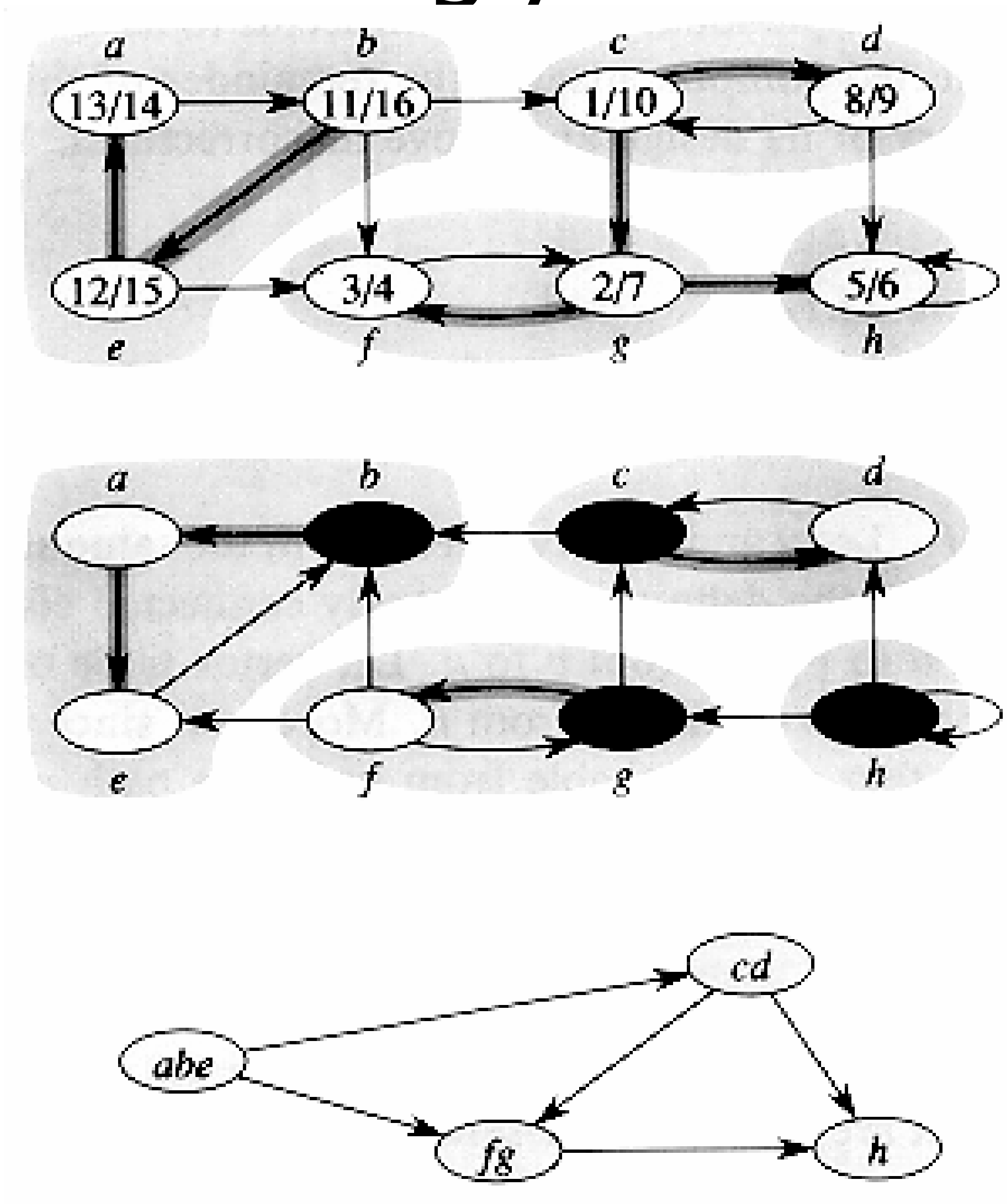
Topological Sort



ทำ DFS ลำดับจากซ้ายไปขวา
จะเรียงตาม f จากมากไปน้อย



Strongly Connected Components



ทำ DFS(G)

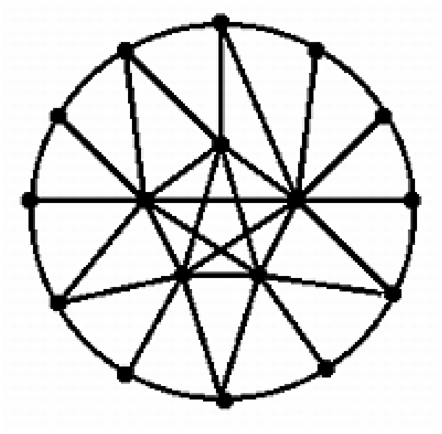
ทำ DFS(G^T) โดยให้พิจารณา u เรียงตาม f จากมากไปน้อย ที่หาได้จาก DFS ครั้งแรก

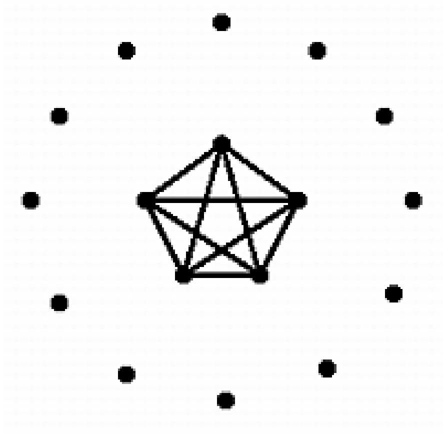
แต่ละต้นในป่าไม้ที่ได้คือ SCCs

```
DFS(G)
  for each vertex u ∈ V[G]
    do color[u] ← WHITE
       p[u] ← NIL
  time ← 0
  for each vertex u ∈ V[G]
    do if color[u] = WHITE
       then DFS-VISIT(u)
```

Hard Graph Problems

clique

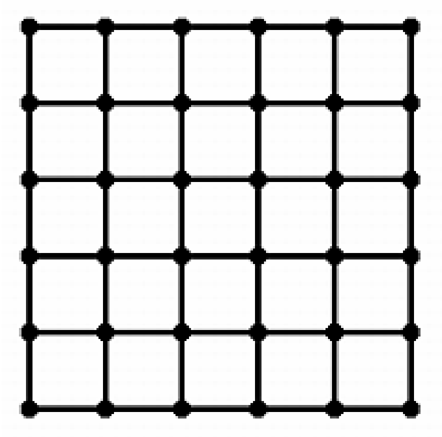


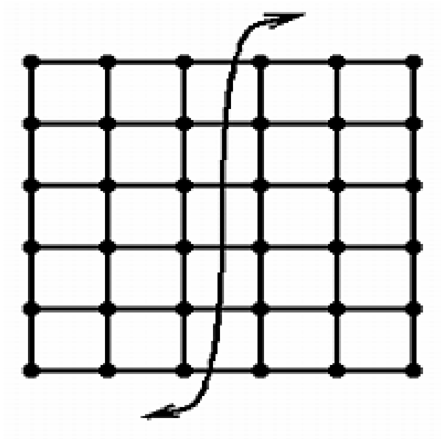


input

output

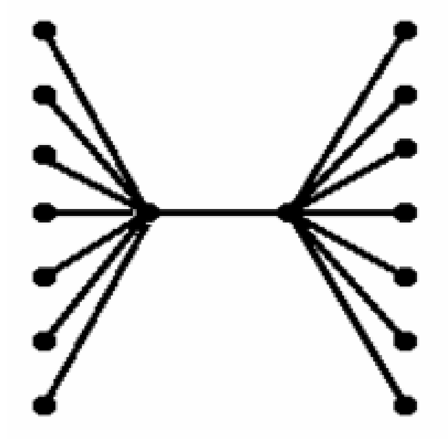
partitioning



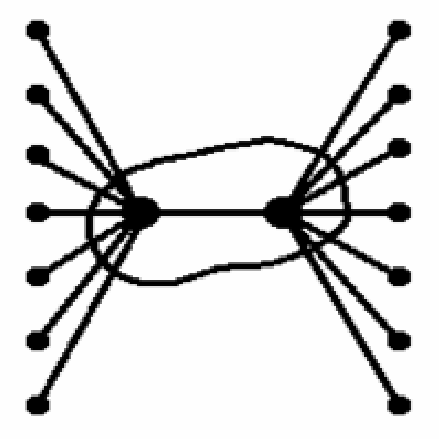


Hard Graph Problems

vertex cover

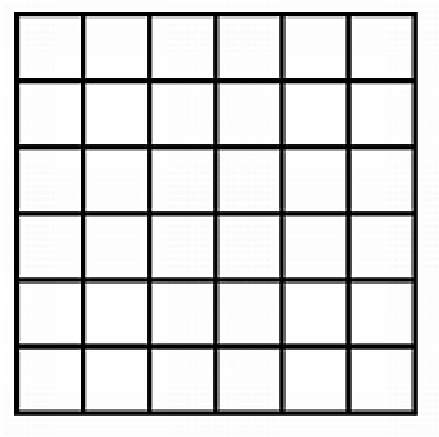


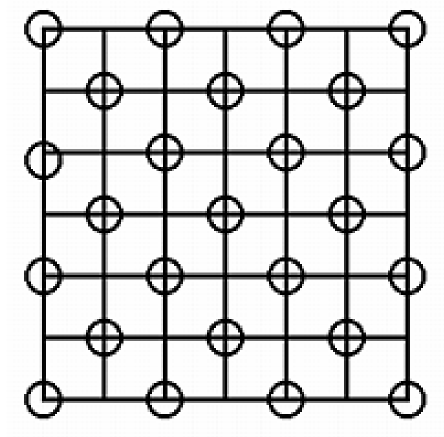
input



output

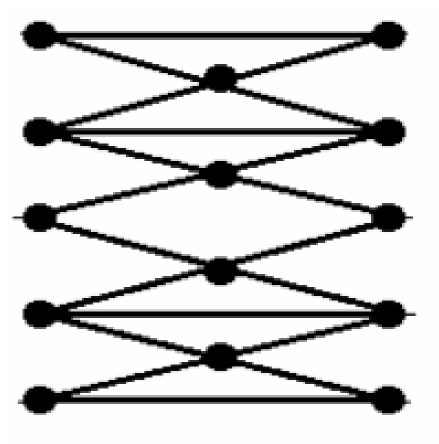
independent set



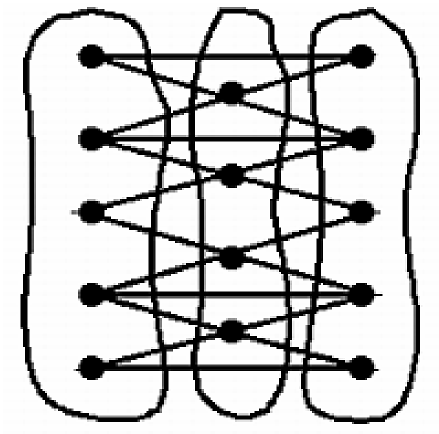


Hard Graph Problems

vertex coloring



input



output

edge coloring