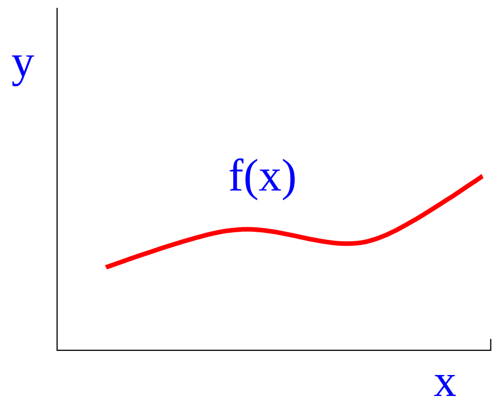
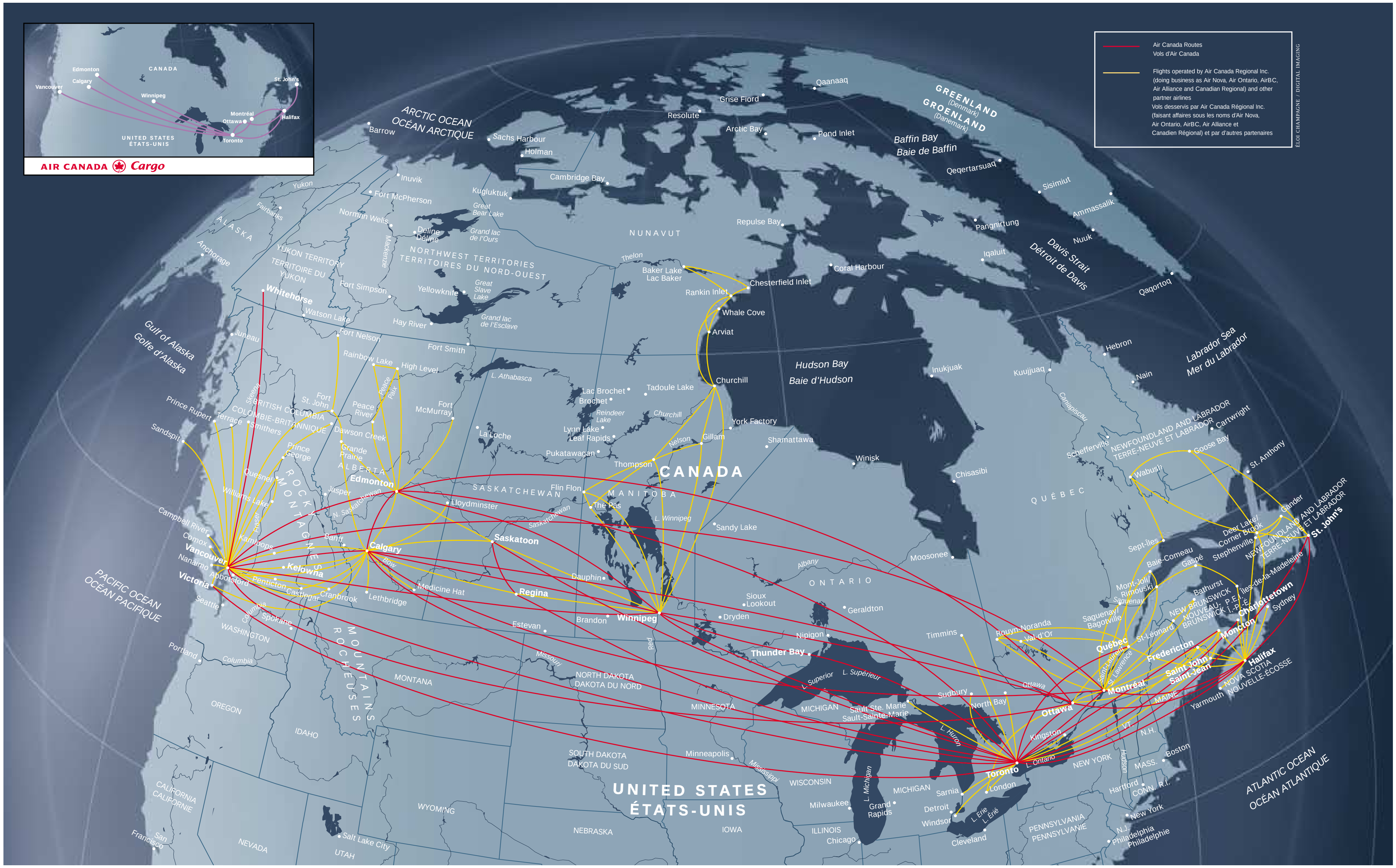
Graphs

The term **graph** is used with several different meanings.

It often means a graphical representation of functions, e.g. a statistical evaluation of data.



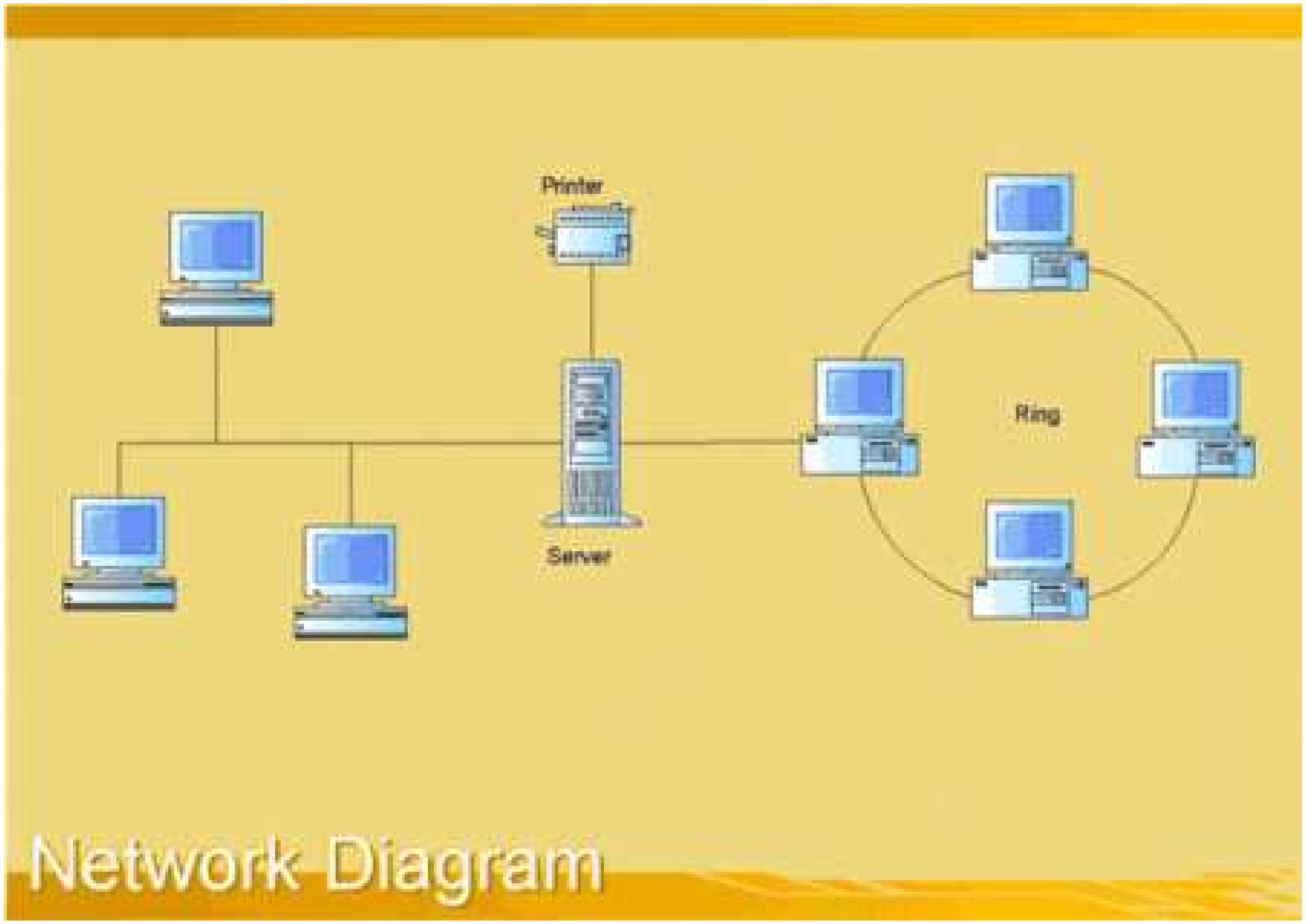
We will study a different type of graphs:
a graphical representation of large structures.



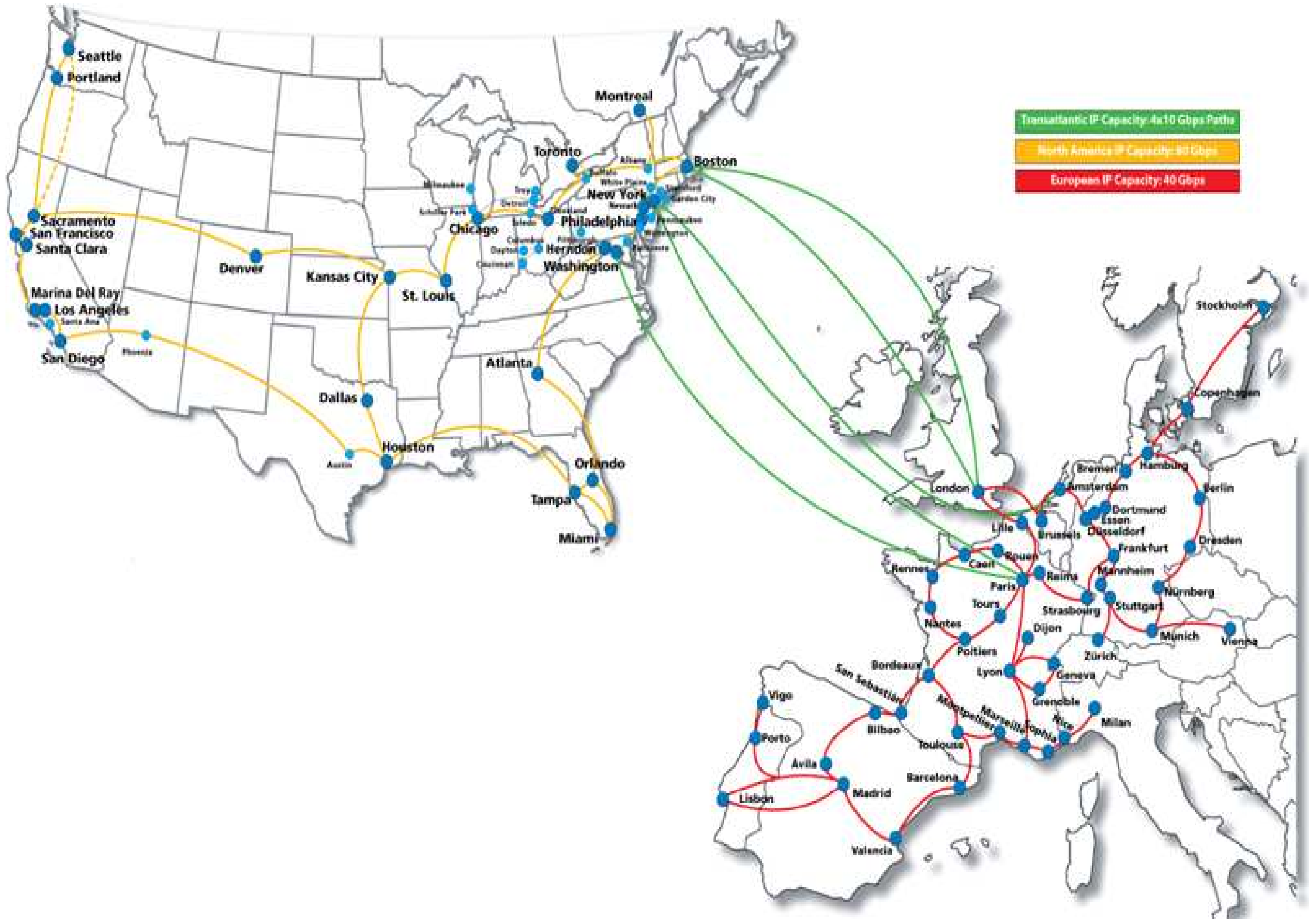
Air Canada Routes
Vols d'Air Canada

Flights operated by Air Canada Regional Inc. (doing business as Air Nova, Air Ontario, AirBC, Air Alliance and Canadian Regional) and other partner airlines
Vols desservis par Air Canada Régional Inc. (faisant affaires sous les noms d'Air Nova, Air Ontario, AirBC, Air Alliance et Canadien Régional) et par d'autres partenaires

ÉLOI CHAMPAGNE / DIGITAL IMAGING

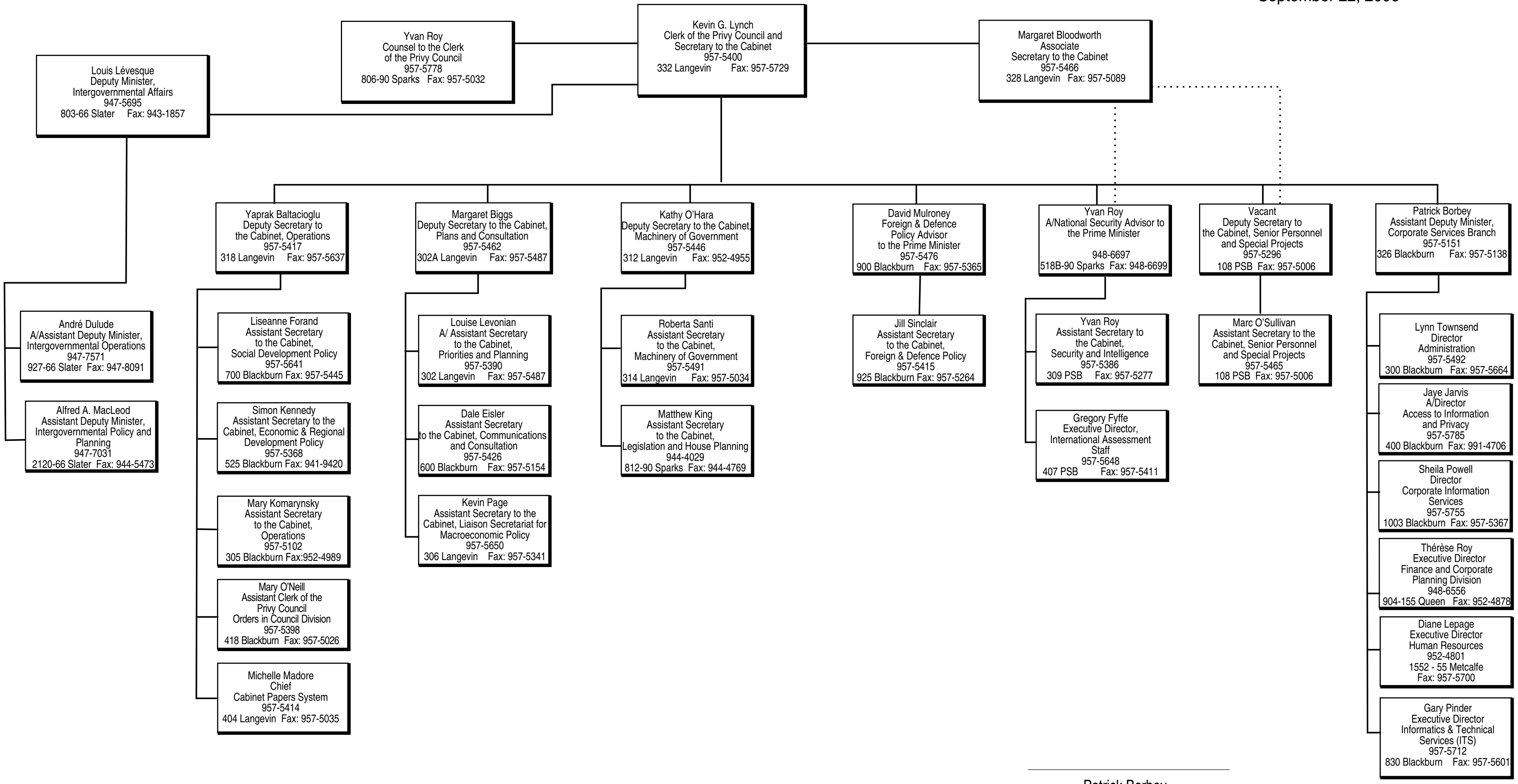


Network Diagram


Transatlantic IP Capacity: 4x10 Gbps Paths
North America IP Capacity: 80 Gbps
European IP Capacity: 40 Gbps
Seattle
Portland
Sacramento
San Francisco
Santa Clara
Denver
Kansas City
St. Louis
Marina Del Ray
Los Angeles
Santa Ana
San Diego
Phoenix
Dallas
Austin
Houston
Atlanta
Orlando
Tampa
Miami
Chicago
Milwaukee
Schiller Park
Troy
Detroit
Toledo
Cleveland
Columbus
Dayton
Cincinnati
Pittsburgh
Herndon
Washington
Baltimore
Philadelphia
Wilmington
New York
Newark
White Plains
Garden City
Stamford
Albany
Buffalo
Toronto
Montreal
Boston
Stockholm
Copenhagen
Hamburg
Bremen
Berlin
Amsterdam
London
Dortmund
Essen
Düsseldorf
Brussels
Lille
Dresden
Frankfurt
Mannheim
Nürnberg
Stuttgart
Munich
Vienna
Rennes
Caen
Rouen
Reims
Paris
Tours
Strasbourg
Dijon
Nantes
Poitiers
Zürich
Lyon
Geneva
Grenoble
Bordeaux
San Sebastian
Montpellier
Marseille
Sophia
Nice
Milan
Toulouse
Vigo
Porto
Bilbao
Avila
Madrid
Lisbon
Barcelona
Valencia

Privy Council Office

September 22, 2006



- Kevin G. Lynch, Clerk of the Privy Council and Secretary to the Cabinet, 957-5400, 332 Langevin Fax: 957-5729
  - Yvan Roy, Counsel to the Clerk of the Privy Council, 957-5778, 806-90 Sparks Fax: 957-5032
  - Margaret Bloodworth, Associate Secretary to the Cabinet, 957-5466, 328 Langevin Fax: 957-5089
  - Louis Lévesque, Deputy Minister, Intergovernmental Affairs, 947-5695, 803-66 Slater Fax: 943-1857
    - André Dulude, A/Assistant Deputy Minister, Intergovernmental Operations, 947-7571, 927-66 Slater Fax: 947-8091
    - Alfred A. MacLeod, Assistant Deputy Minister, Intergovernmental Policy and Planning, 947-7031, 2120-66 Slater Fax: 944-5473
  - Yaprak Baltacioglu, Deputy Secretary to the Cabinet, Operations, 957-5417, 318 Langevin Fax: 957-5637
    - Liseanne Forand, Assistant Secretary to the Cabinet, Social Development Policy, 957-5641, 700 Blackburn Fax: 957-5445
    - Simon Kennedy, Assistant Secretary to the Cabinet, Economic & Regional Development Policy, 957-5368, 525 Blackburn Fax: 941-9420
    - Mary Komarynsky, Assistant Secretary to the Cabinet, Operations, 957-5102, 305 Blackburn Fax:952-4989
    - Mary O'Neill, Assistant Clerk of the Privy Council, Orders in Council Division, 957-5398, 418 Blackburn Fax: 957-5026
    - Michelle Madore, Chief, Cabinet Papers System, 957-5414, 404 Langevin Fax: 957-5035
  - Margaret Biggs, Deputy Secretary to the Cabinet, Plans and Consultation, 957-5462, 302A Langevin Fax: 957-5487
    - Louise Levonian, A/ Assistant Secretary to the Cabinet, Priorities and Planning, 957-5390, 302 Langevin Fax: 957-5487
    - Dale Eisler, Assistant Secretary to the Cabinet, Communications and Consultation, 957-5426, 600 Blackburn Fax: 957-5154
    - Kevin Page, Assistant Secretary to the Cabinet, Liaison Secretariat for Macroeconomic Policy, 957-5650, 306 Langevin Fax: 957-5341
  - Kathy O'Hara, Deputy Secretary to the Cabinet, Machinery of Government, 957-5446, 312 Langevin Fax: 952-4955
    - Roberta Santi, Assistant Secretary to the Cabinet, Machinery of Government, 957-5491, 314 Langevin Fax: 957-5034
    - Matthew King, Assistant Secretary to the Cabinet, Legislation and House Planning, 944-4029, 812-90 Sparks Fax: 944-4769
  - David Mulroney, Foreign & Defence Policy Advisor to the Prime Minister, 957-5476, 900 Blackburn Fax: 957-5365
    - Jill Sinclair, Assistant Secretary to the Cabinet, Foreign & Defence Policy, 957-5415, 925 Blackburn Fax: 957-5264
  - Yvan Roy, A/National Security Advisor to the Prime Minister, 948-6697, 518B-90 Sparks Fax: 948-6699
    - Yvan Roy, Assistant Secretary to the Cabinet, Security and Intelligence, 957-5386, 309 PSB Fax: 957-5277
    - Gregory Fyffe, Executive Director, International Assessment Staff, 957-5648, 407 PSB Fax: 957-5411
  - Vacant, Deputy Secretary to the Cabinet, Senior Personnel and Special Projects, 957-5296, 108 PSB Fax: 957-5006
    - Marc O'Sullivan, Assistant Secretary to the Cabinet, Senior Personnel and Special Projects, 957-5465, 108 PSB Fax: 957-5006
  - Patrick Borbey, Assistant Deputy Minister, Corporate Services Branch, 957-5151, 326 Blackburn Fax: 957-5138
    - Lynn Townsend, Director, Administration, 957-5492, 300 Blackburn Fax: 957-5664
    - Jaye Jarvis, A/Director, Access to Information and Privacy, 957-5785, 400 Blackburn Fax: 991-4706
    - Sheila Powell, Director, Corporate Information Services, 957-5755, 1003 Blackburn Fax: 957-5367
    - Thérèse Roy, Executive Director, Finance and Corporate Planning Division, 948-6556, 904-155 Queen Fax: 952-4878
    - Diane Lepage, Executive Director, Human Resources, 952-4801, 1552 - 55 Metcalfe, Fax: 957-5700
    - Gary Pinder, Executive Director, Informatics & Technical Services (ITS), 957-5712, 830 Blackburn Fax: 957-5601

Patrick Borbey
Assistant Deputy Minister
Corporate Services Branch

Graphs are used to represent large discrete systems:

Transportation networks,

Computer Networks,

Communication systems,

Structure of Organizations

Software Structure,

....

Thus we consider a graph as an **abstraction** of the **structure of discrete systems**.

This will allow us to find **general properties** of discrete systems and **algorithms** that are applicable in all discrete systems.

Several types of graphs are also used as basic data structures and we need to know their properties.

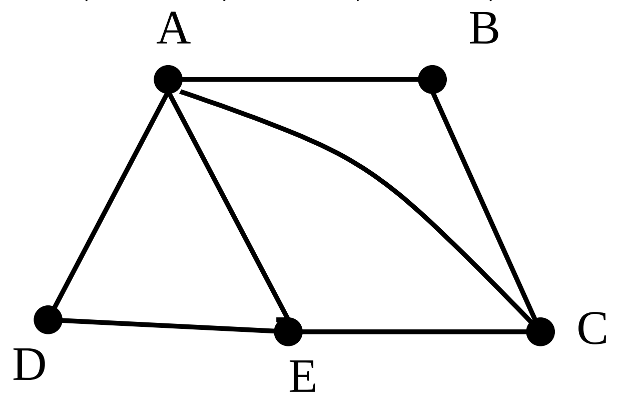
A graph consists of a set of **nodes** (some call them points, vertices), and a set **edges** connecting some pairs of nodes.

The set of nodes of a graph G is often denoted by V and the set of edges by E.

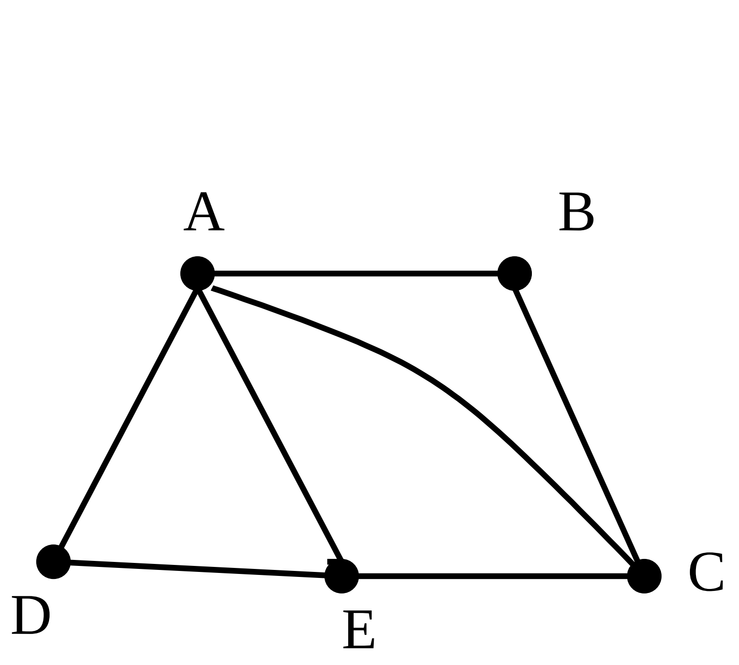
So we write $G = (V, E)$

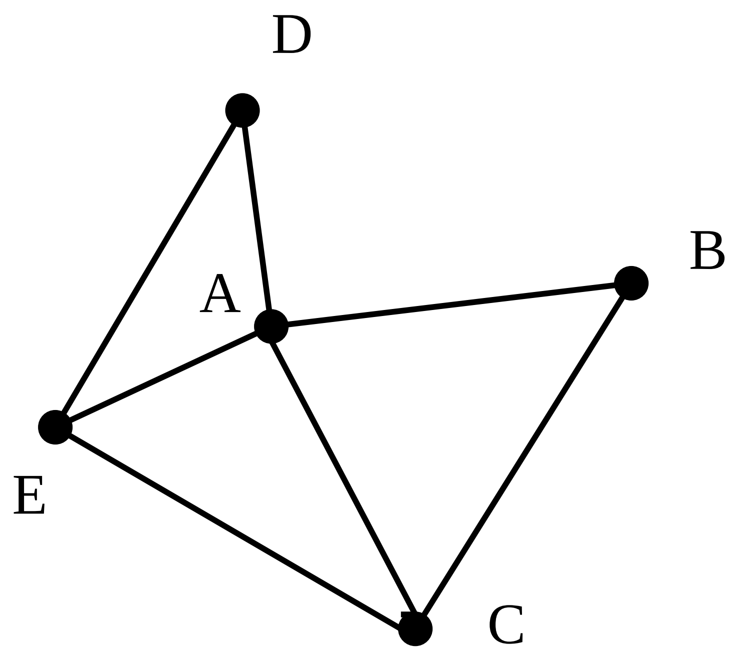
If $V = \{A, B, C, D, E\}$, $E = \{AB, AC, AD, AE, BC, CE, ED\}$

then a picture of the graph is



The two graphs below are the **same**:





they have the same nodes $\{A, B, C, D, E\}$ and

the same edges $\{AB, AC, AD, AE, BC, CE, ED\}$.

Only their pictures are **drawn differently**.

Notice that two graphs are different if they have different vertex set or different edge set.

In some applications we are only interested in the general type of the graph and we don't necessarily specify the vertex set in detail.

In that case we don't put names of the node in the picture representation of the graph.

We shall start by considering **simple graphs:**

- there is no edge connecting a node to itself (edge of type AA is not allowed),

- there is at most one edge between two nodes.

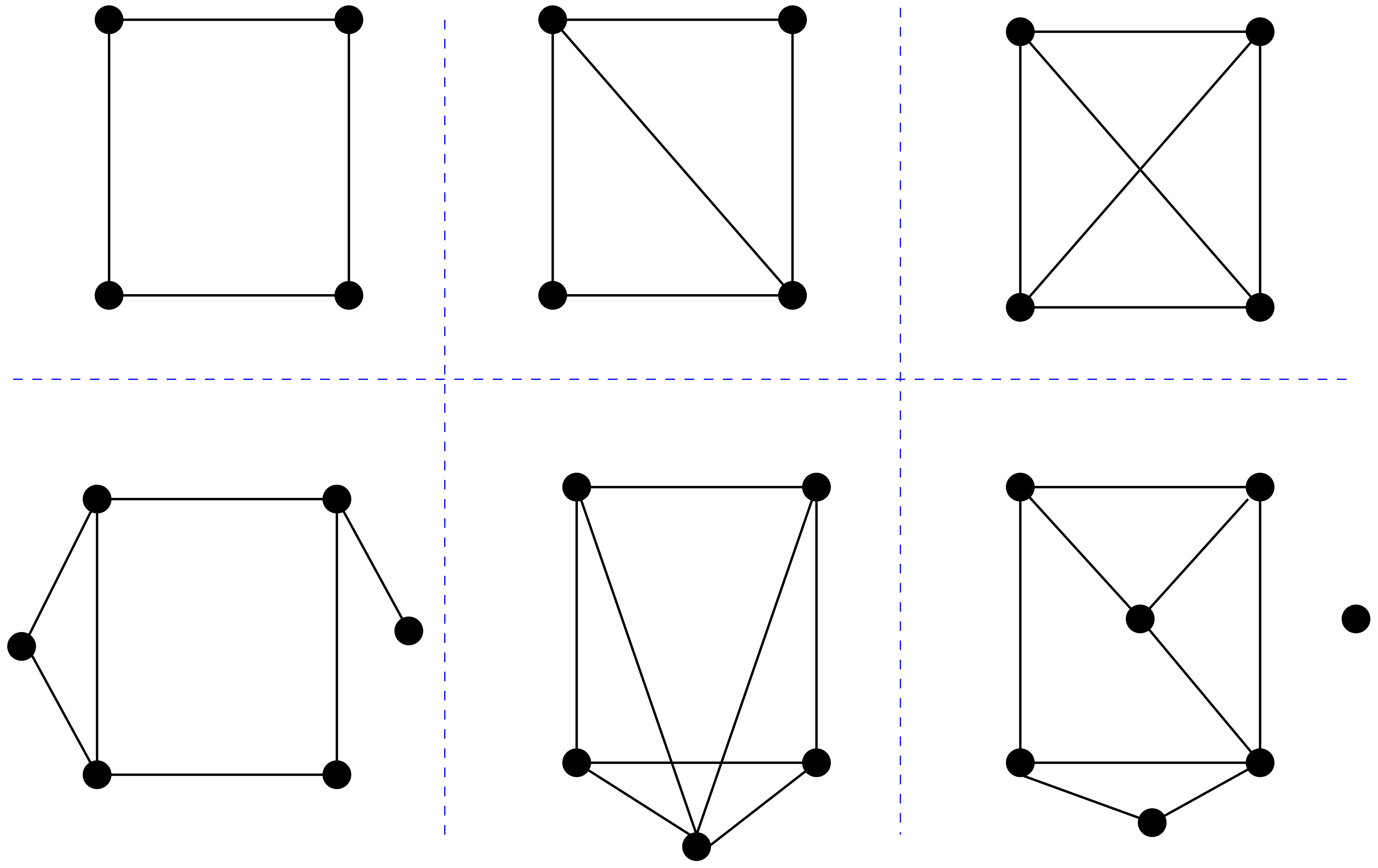
Basic terms:

Two nodes connected by an edge are called **adjacent**.

Nodes adjacent to a given node are called its **neighbours**.

Degree of a node is the number of its neighbours. The degree of node v is denoted by $d(v)$.

6 examples of simple graphs



Theorem 16 *In a graph $G = (V, E)$ the sum of the degrees of all the nodes is equal to $2|E|$.*

Proof

Each edge contributes 2 to the sum degrees of nodes it connects.

Theorem 17 *In every graph, the number of nodes with odd degree is even.*

Proof

If a graph contains odd number of nodes with odd degree, then the sum of the degrees of all the nodes would be odd, a contradiction to Theorem 29

Some graphs occur often and are thus given special names.

Edgeless graph: a graph in which E is empty.

Complete graph: a graph in which any pair of nodes is connected by an edge.

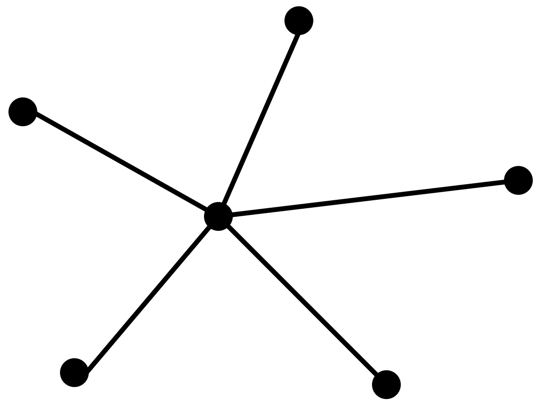
Such a graph is called a complete graph (or a clique).

A complete graph with n nodes has $\binom{n}{2} = n(n-1)/2$ edges and is often denoted as K_n.

The **complement** of a graph $G = (V, E)$ is the graph $\overline{G} = (V, V \times V - E)$.

$\overline{G}$ is the graph having the same nodes as G and containing all possible edges which are not in G.

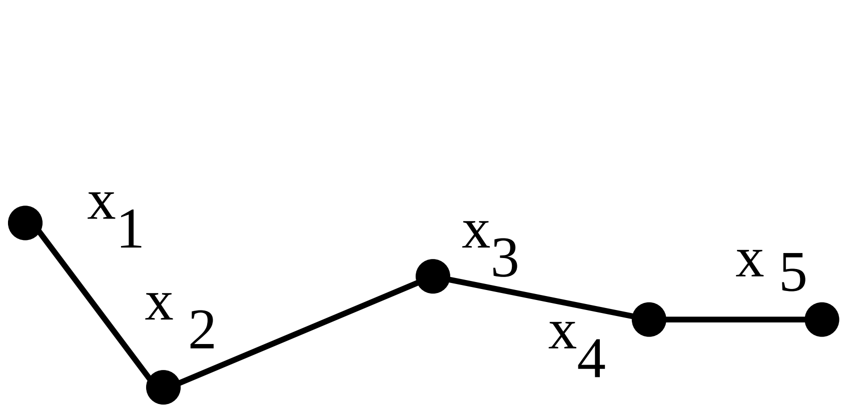
Star: a graph in which one node is connected to all other nodes and no other nodes are connected.

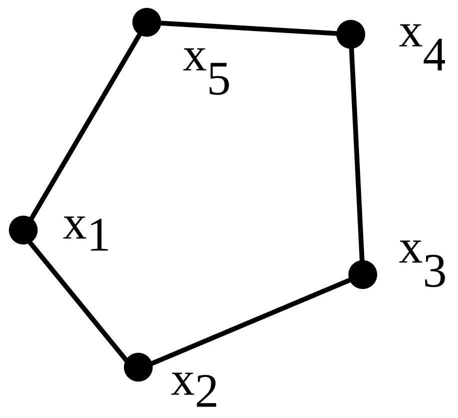


A star with n nodes has $n-1$ edges

Path: a sequence of distinct nodes in which there is an edge between consecutive nodes in the sequence.

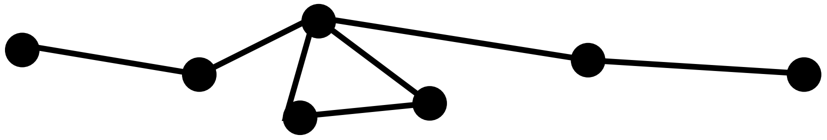
There are two nodes of degree 1 in a path, called the **endpoints**, all other nodes are of degree 2.





Cycle: a sequence of distinct nodes in which each node is connected to the next node and a previous node, and the last one is connected to the first one.

Walk: a sequence of nodes (not necessarily distinct) in which there is an edge between consecutive nodes in the sequence.



Since nodes can repeat, the walk can be **closed** if the first and last node of the walk are the same.

Paths, cycles are special cases of walks.

The **length** of a walk, path or a cycle is the number of edges in it.

A cycle of length k is often called a $k-$cycle.

A graph H is called a **subgraph** of a graph G if H can be obtained from G by omitting some nodes and edges. (If a node is omitted, we have to remove all edges to it.

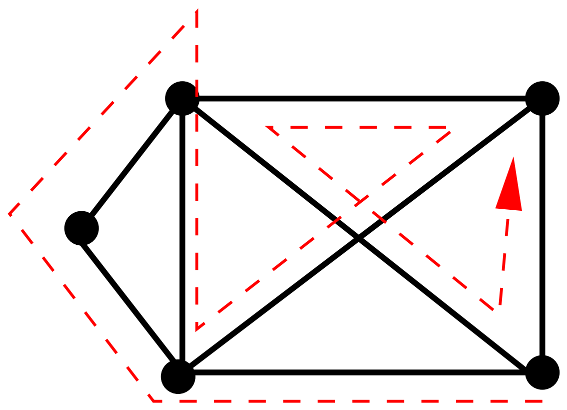
Graph G is **connected** if for any two nodes u and v of the graph there exists a path with endpoints u and v that is a subgraph of G.

A **connected component** of graph G is a maximal subgraph of G that is connected.

Eulerian Walks and Hamiltonian Cycles

Eulerian walk is a walk that contains each edge of the graph exactly once.

Example:



An Eulerian walk can be a useful way to

- verify all links in a network in a systematic manner,
- for drawing a graph in a continuous motion.

Theorem 18

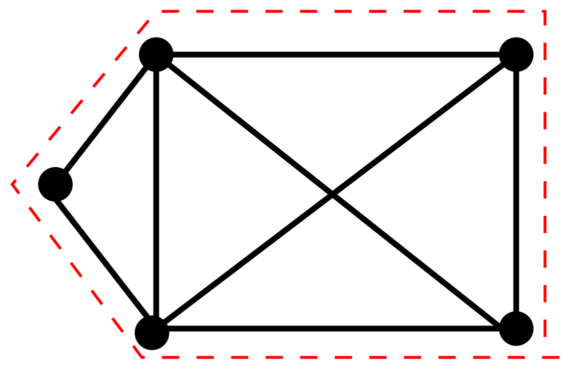
(a) If a connected graph has more than two nodes of odd degree then it has no Eulerian walk.

(b) If a connected graph has exactly two nodes of odd degree then it has an Eulerian walk. Every Eulerian walk starts in one of these and ends in the other one.

(c) If a connected graph has no nodes of odd degree then it has Eulerian walk. Every Eulerian walk is closed.

Hamiltonian cycle is a cycle that contains all nodes of a graph.

(Unlike in an Eulerian walk, no node can be visited more than once)

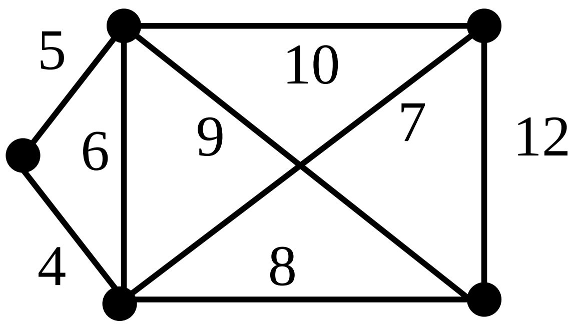


A Hamiltonian cycle can be a useful way to verify all nodes in a network in a systematic manner, without visiting any node twice.

There is no theorem that allow us to determine, for any graph, if it has a Hamiltonian cycle.

The Hamilton cycle problem is related to the **Traveling Salesman Problem**:

We have a graph G with distances assigned to the edges between nodes.



Problem: Find the lowest cost close walk through the graph that visits every node.

This problem is computationally very difficult, but it can be solved optimally for quite large graphs. (Chvatal et al.)

Trees

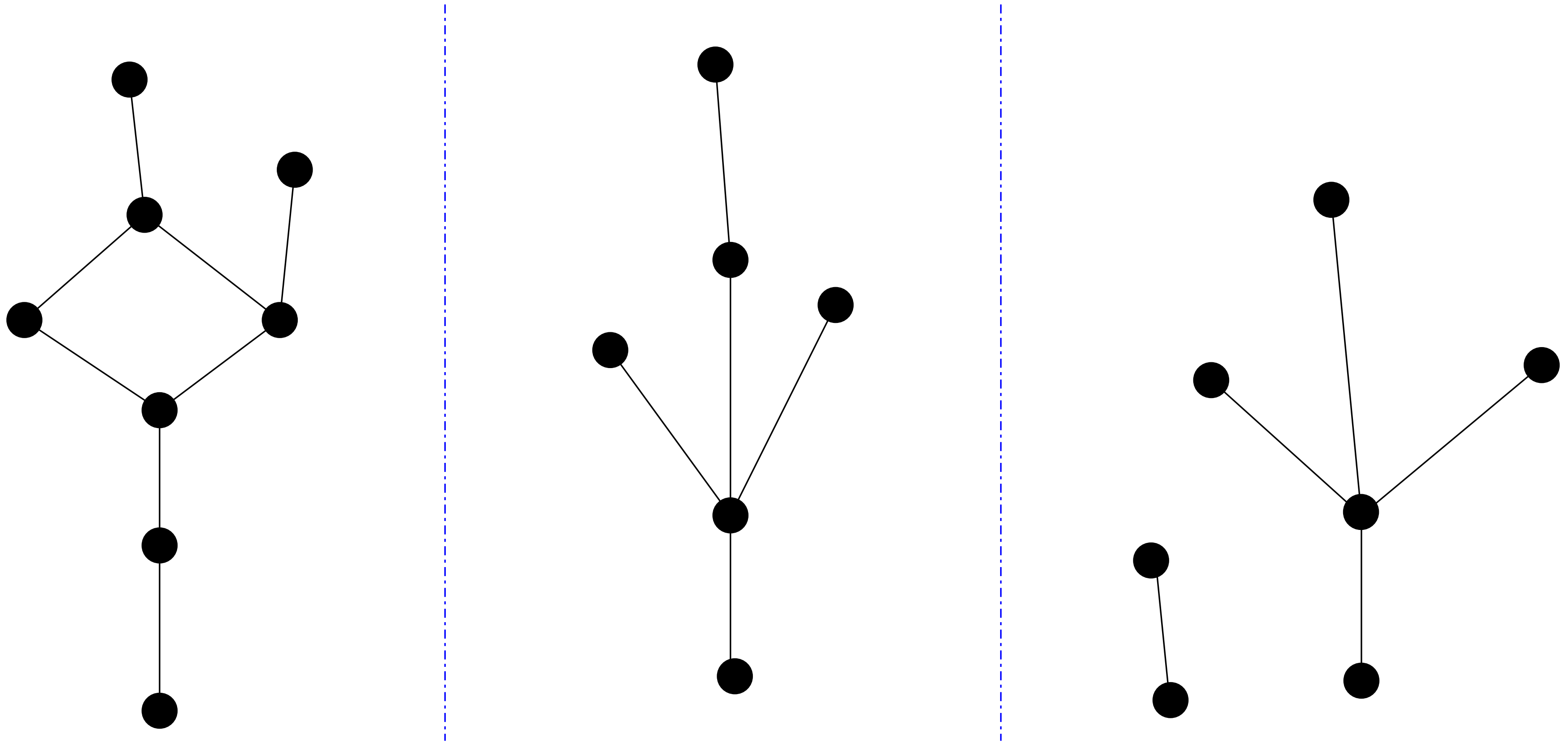
Tree is a special type of graph that is used in many computer applications:

- Enumeration procedures,
- Data structures

What do we intuitively understand to be a tree?

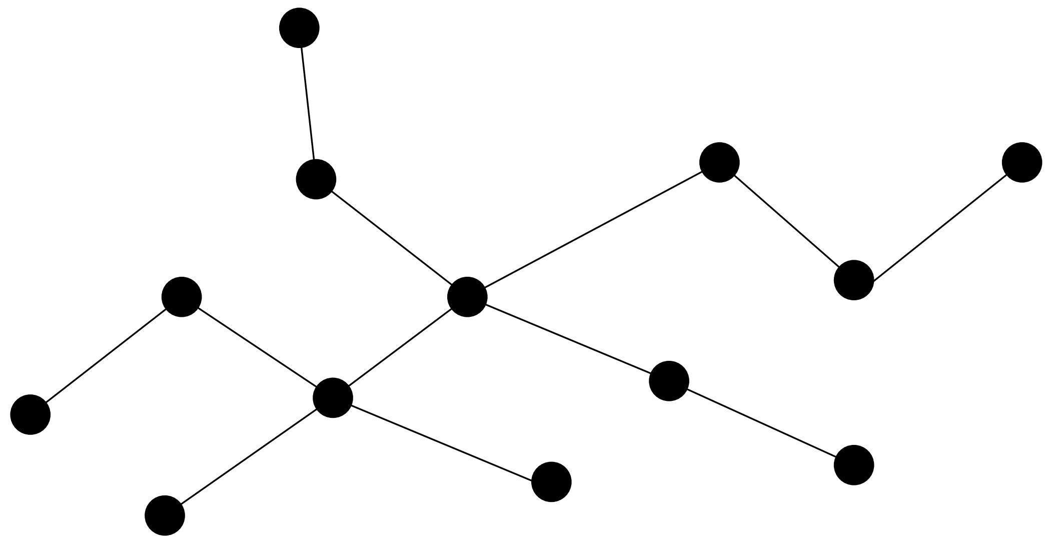
When should we call a graph to be a tree?

Which one of the graphs below should we call a tree?



Definition

A graph $G = (V, E)$ is called a tree if it is **connected** and **contains no cycle** as a subgraph.



It does not need to look like a tree in nature.

Theorem 19

(a) A graph G is a tree if and only if it is connected, but deleting any of its edges results in a disconnected graph.

(b) A graph G is a tree if and only if it contains no cycles, but adding any new edge creates a cycle.

Proof: Done in class

Theorem 20

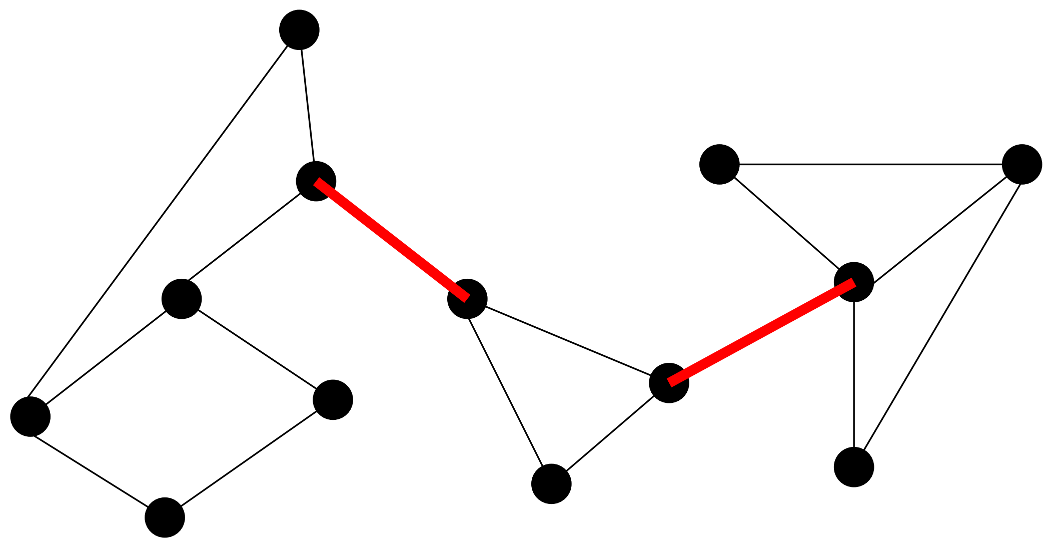
In a tree, every two nodes are connected by a unique *path.*

Conversely, if in a graph every pair of nodes is connected by by a unique path, then the graph is a tree.

Proof done in class.

An edge is called a **cut-edge** of a connected graph G if removing that edge makes the graph disconnected.

Cut-edges are in red color:



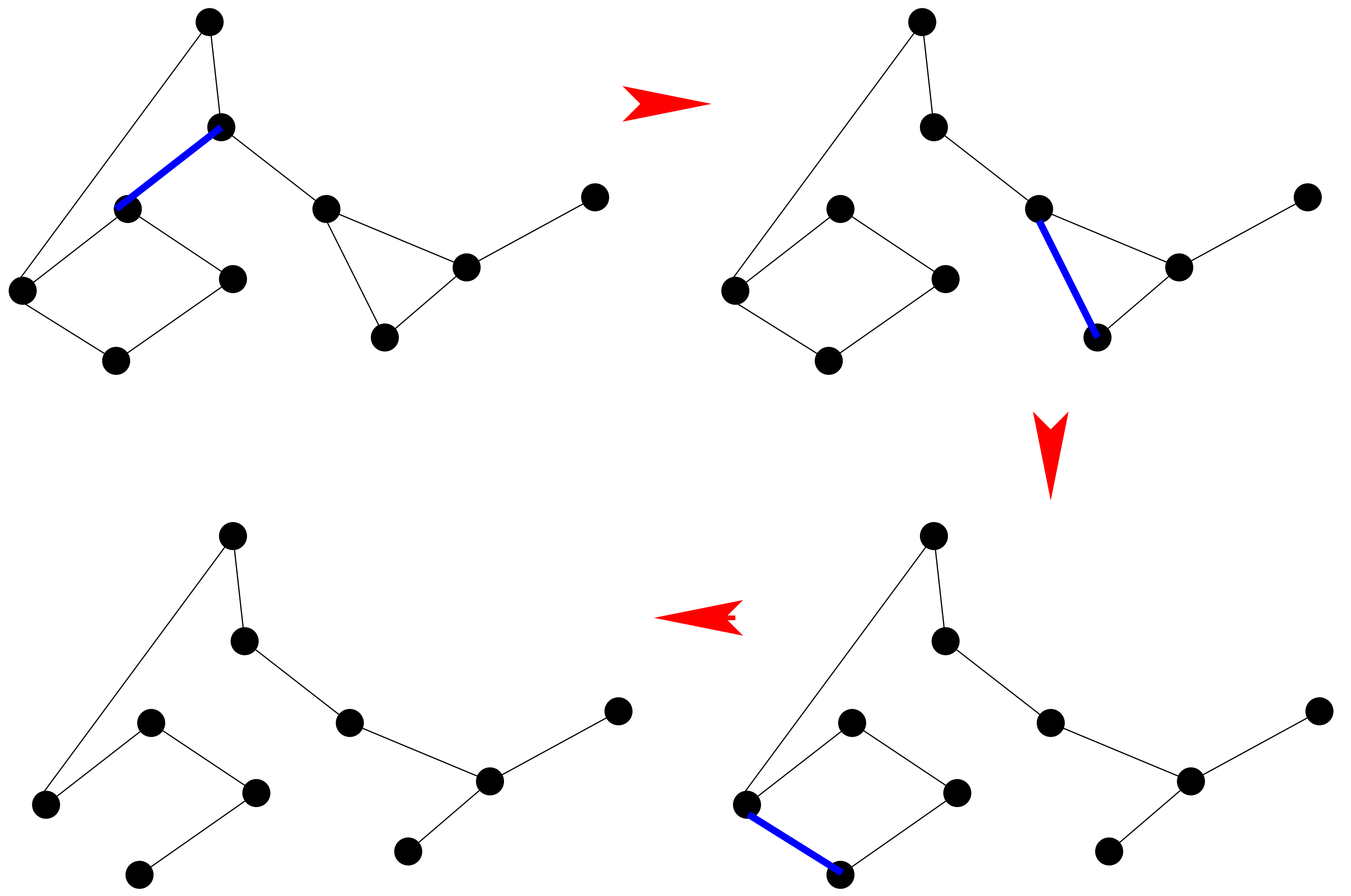
In a tree, every edge is a cut-edge.

Take a connected graph G and execute the following process:

```
finished = false;
do
   if (there is an edge $e$ that is not a cut edge)
      then remove edge $e$ from $G$;
   else finished = true;
until finished;
```

The graph G' which is obtained in this manner is connected and it does not contain any cut-edge.

G' is a tree that contains all nodes of G.



Such a tree is called a **spanning tree** of G.

In Computer Science, we often use trees in which one node is distinguished as the **root** of the tree.

The root is usually the node from which the construction of a tree starts.

A tree with specified root is called a **rooted tree**.

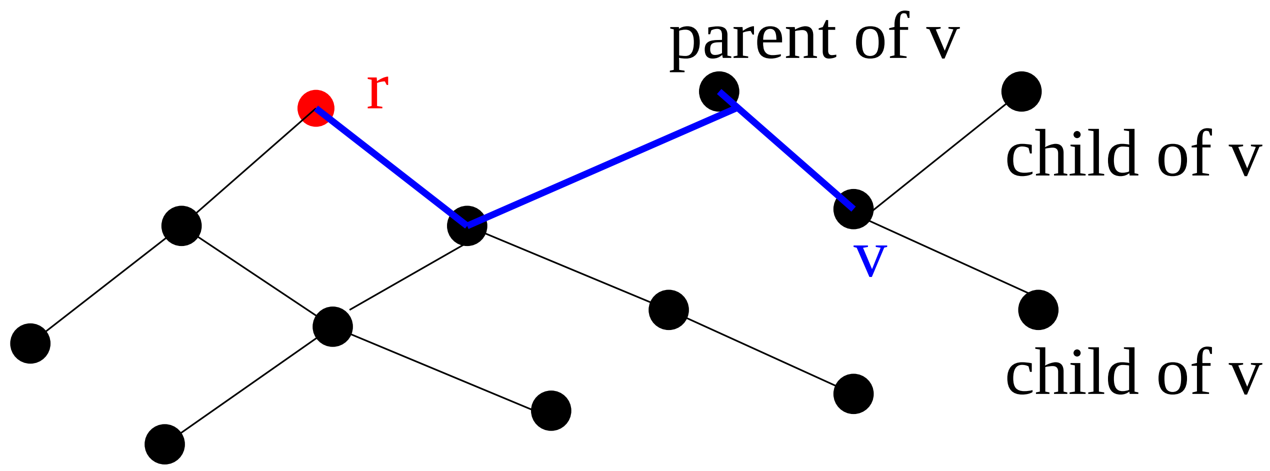
Rooted trees have been used as *family trees* to show the ancestry of related people.

For that reason a family terminology is used in rooted trees.

Let G be a rooted tree with root r.

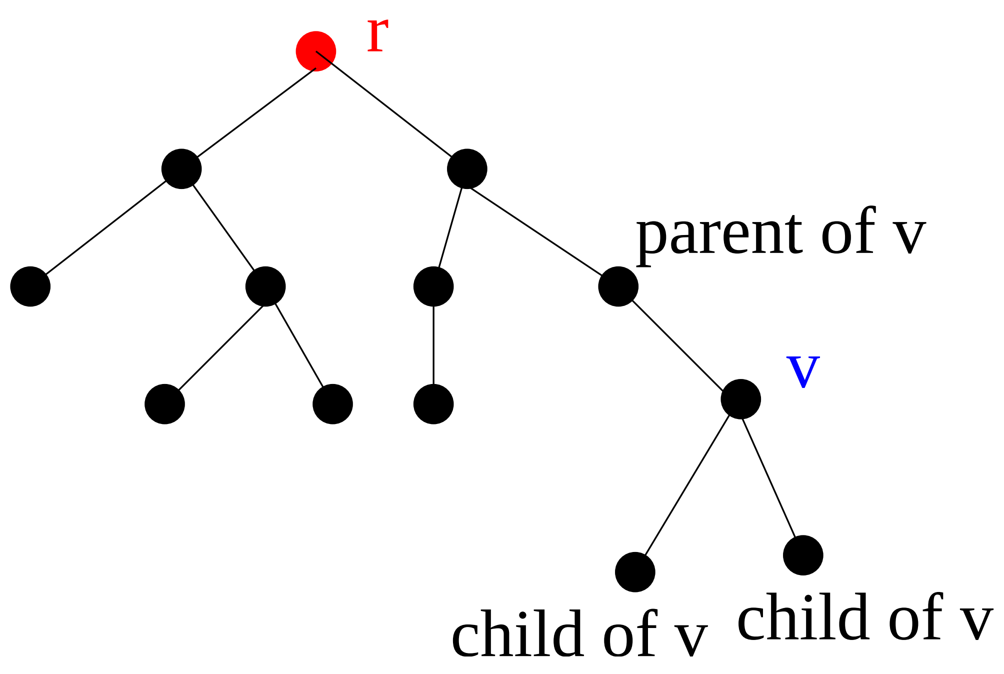
For a node v different from r, consider the unique path from r to v.

The node on the path that directly precedes v is called the **parent** of v. All other neighbors of v are called the **children** of v.



A **leaf** is a node without children (i.e., of degree 1).

In computer science we commonly represent a rooted tree by putting the root at the top and putting the children one level down from their parent.



Tree growing Procedure

- Start G with a single node.
- Repeat any number of times:

  take a new node and

  connect it to one of the existing nodes of G.

Theorem 21 *Every graph obtained by the* Tree growing Procedure *is a tree. Every tree can be obtained in this manner.*

Proof: Done in class

The *Tree-growing Procedure* can be used to establish a number of properties of trees.

Theorem 22 *Every tree on n nodes has n-1 edges.*

Proof:

When building a tree we start with one node and no edge.

At each step, one new node and one new edge is added, so this difference of 1 between the number of nodes and edges is maintained.

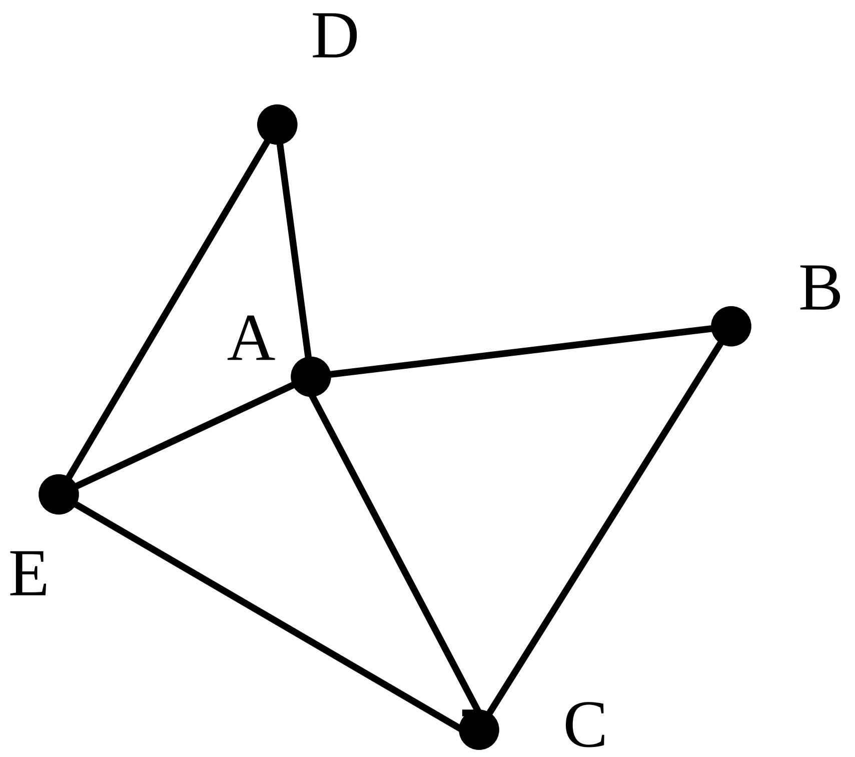
Computer Representation of Graphs

Adjacency matrix: a 2-dimensional matrix M.

For each node we have one row and one column.

$$M_{i.j} = \begin{cases} 1 \text{ if there is an edge between } i\text{th and } j\text{th nodes,} \\ 0 \text{ if there is no edge between } i\text{th and } j\text{th nodes.} \end{cases}$$

Example:



	A	B	C	D	E
A	0	1	1	1	1
B	1	0	1	0	0
C	1	1	0	0	1
D	1	0	0	0	1
E	1	0	1	1	0

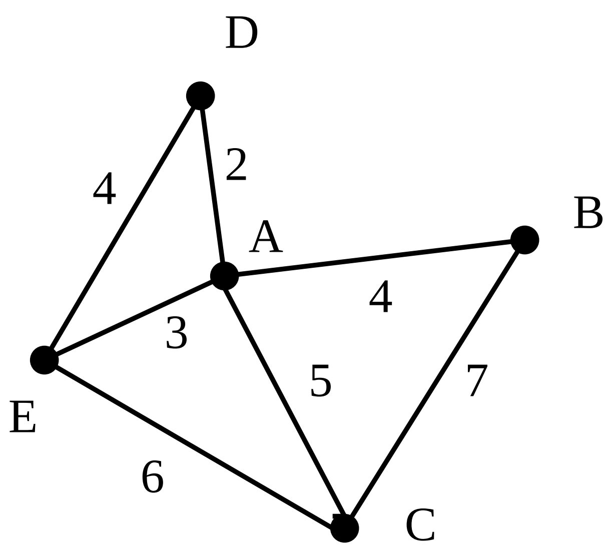
Adjacency matrix is a common computer representation of graphs.

For a simple graph its adjacency matrix is *symmetric* so we only need to store the part below the diagonal.

In some applications, the entries in the adjacency matrix are used to store some properties associated of the edges.

Example: In a transportation network we can put there the actual distance between the nodes.

$$M_{i.j} = \begin{cases} 0 \text{ if there is no edge between } i\text{th and } j\text{th nodes,} \\ > 0 \text{ if there is an edge between } i\text{th and } j\text{th nodes,} \\ \quad \text{the value gives the distance in kilometers.} \end{cases}$$



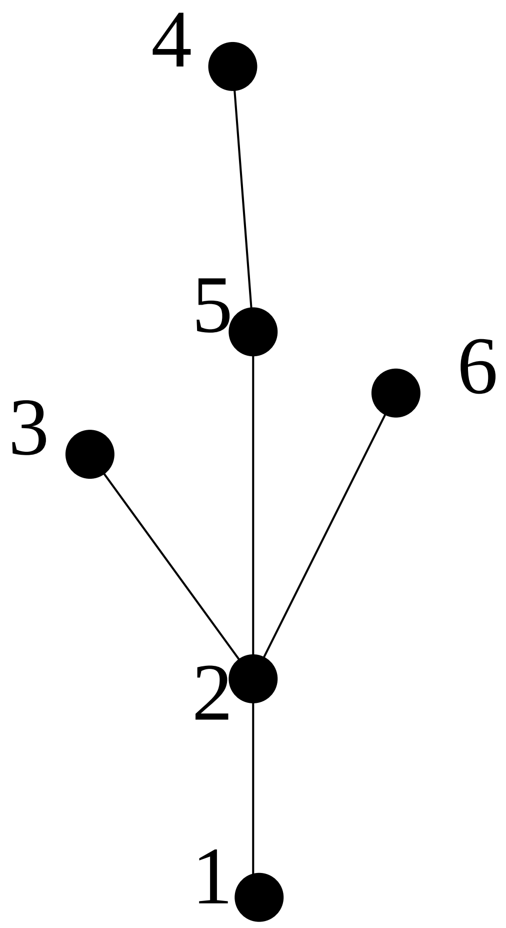
	A	B	C	D	E
A	0	4	5	2	3
B	4	0	7	0	0
C	5	7	0	0	6
D	2	0	0	0	4
E	3	0	6	4	0

For a tree, an adjacency matrix is very sparse: it contains $n(n-1)+1$ zeros and only $n-1$ non-zero elements.

Thus it is not a very efficient way to represent a tree, even when the entries in the table are only bits.

Since the number of edges in a tree is small, we can store a tree by given a **list of edges**

Such a list can be represented in several ways.



Edge−list: (12, 23, 25, 26, 45)

Matrix representation of the edge list:

$$\begin{matrix} 1 & 2 & 2 & 2 & 4 \\ 2 & 3 & 5 & 6 & 5 \end{matrix}$$

Such a representation is not unique.

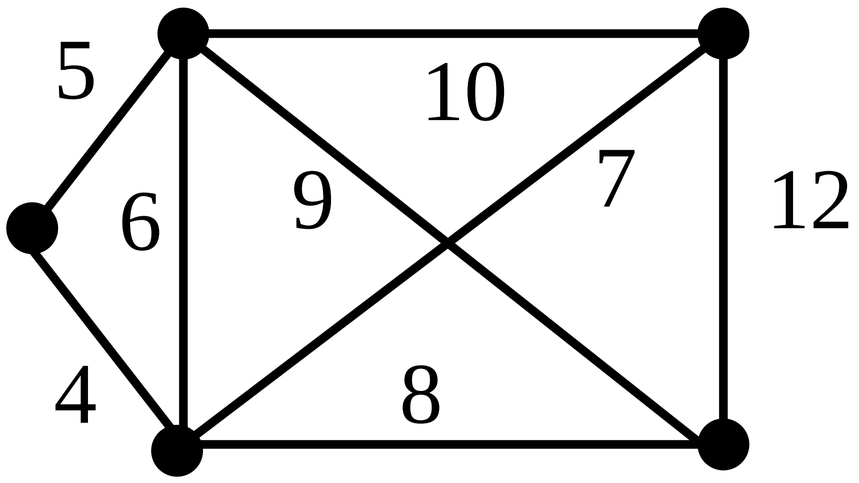
Graph Algorithms

Finding the best tree

Problem to solve: A country with n towns wants to construct a new fiber-optic telephone network for these towns. The cost should be minimized.

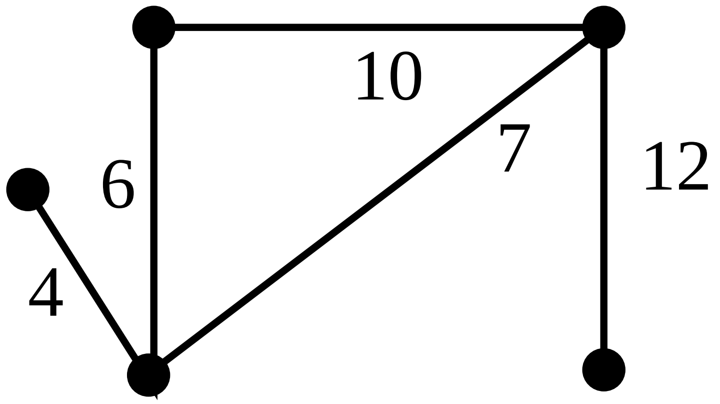
The cost of building a line between any two cities is known.

We can represent the problem as a graph G with values attached to edges corresponding to the cost such a line.

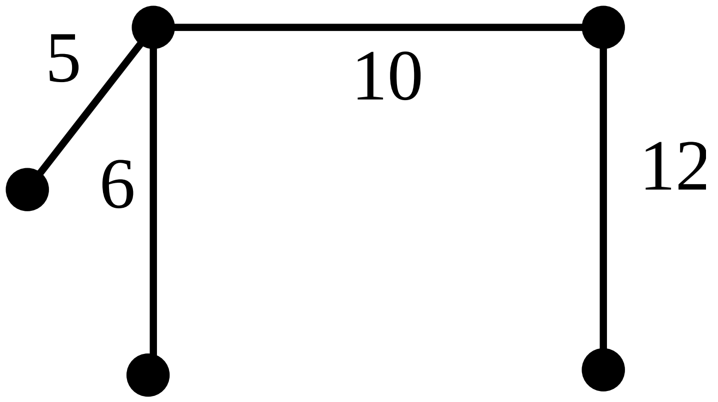


Solution: a **connected subgraph** of G containing **all nodes** of G such that the sum of all the **costs** on the edges is **minimum** from all such subgraphs.

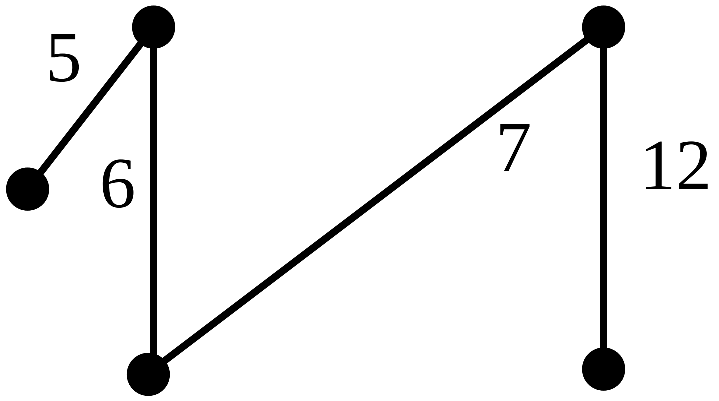
Which one to take?



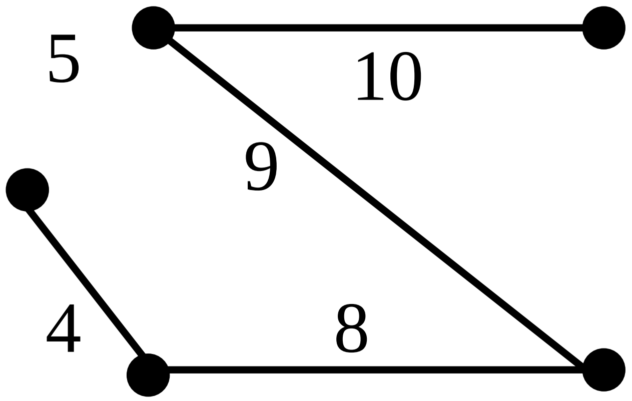
cost = 39



cost = 33



cost = 29



cost = 31

Is there a solution better than any of the above?

The solution should be

- a **spanning tree** of G,
- **minimizing the cost**.

The **minimum spanning tree** of G.

One cannot obtain the answer by trying out all the spanning trees if the graph is large.

There are too many different spanning trees for a large graph on n nodes with m edges: $\approx \binom{m}{n-1}$

Kruskal's Algorithm

```
//$n$ is the number of nodes;
$E$ = sequence of all edges of $G$;
$sort(E)$; // by the weights;
$S = empty$; // set of min. sp. edges;
$i = 1$;
while $|S| < n - 1)$ do
    { if ($e_i$ and $S$ do not create a cycle)
            $S = S \cup e_i$;
        $i++$;
    }
```

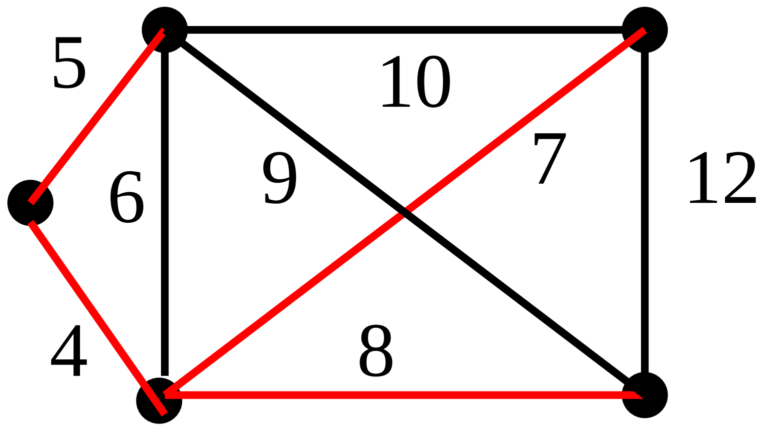
Solution:

take edge 4, take edge 5

skip edge 6 (creates a cycle)

take edge 7, take edge 8

Total cost = 24



Theorem 23 *The Kruskal's algorithm calculates the minimum spanning tree*

Proof: Done in class

This is an example of a **greedy algorithm**.

It takes, at every step what seems to be the cheapest way to extend the network.

It works for spanning trees, but not in some other problems

Notice that the algorithm gives a solution.

There is no unique solution in some cases, like when some of the edges have the same cost.

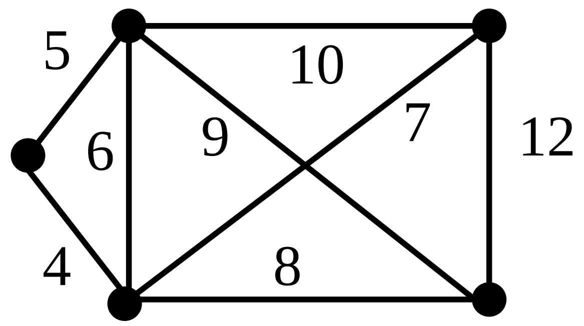
There are other algorithms for producing a minimum spanning tree, but they are not substantially any better or faster (Prim, Boruvka).

Chazelle's algorithm is the fastest.

Traveling Salesman Problem:

(related to the Hamilton cycle problem)

We have a graph G with distances assigned to the edges between nodes.



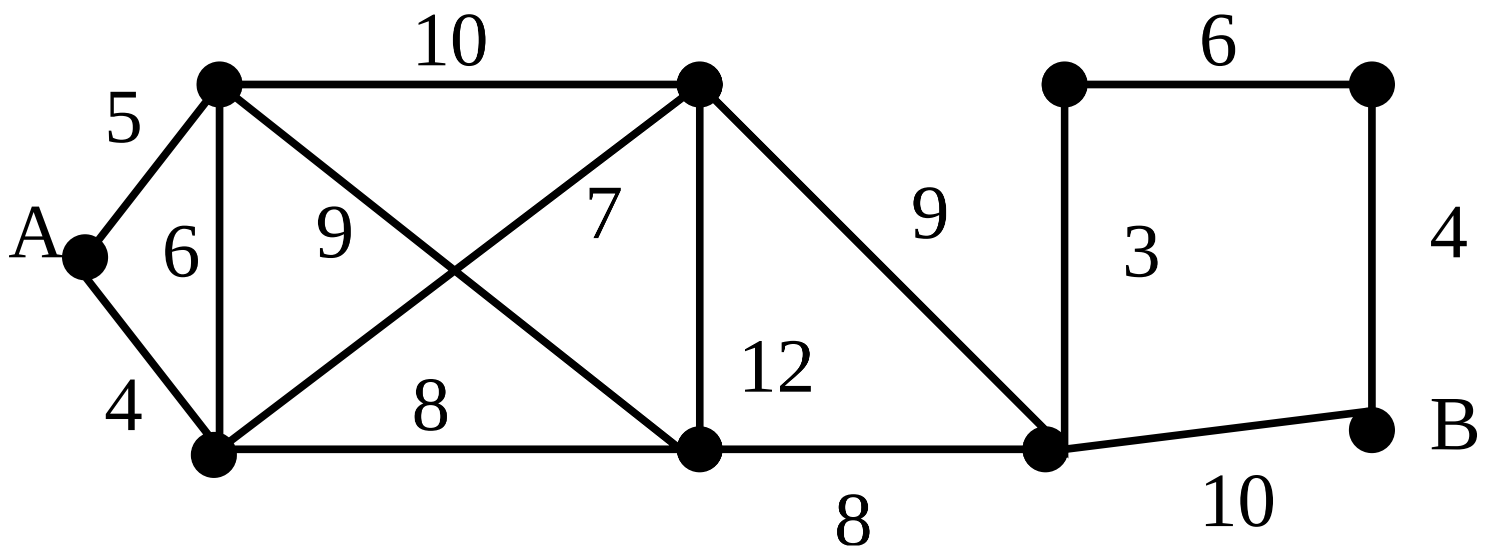
Problem: Find the lowest cost close walk through the graph that visits every node.

This problem is computationally very difficult, but it can be solved optimally for quite large graphs (Chvatal et al.) and approximately in other cases.

Shortest Path Algorithm

Problem: We have a road network. Find the shortest route between two given nodes.

Example: Find the shortest path between A and B



Shortest Path Algorithm is one of the very frequently used algorithms.

Many other problems can be solved using this algorithm.

This algorithm can be used to find the length of a shortest path from a given node A to all other nodes.

Restriction: The cost on edges should be positive numbers (this is the case of almost all applications).

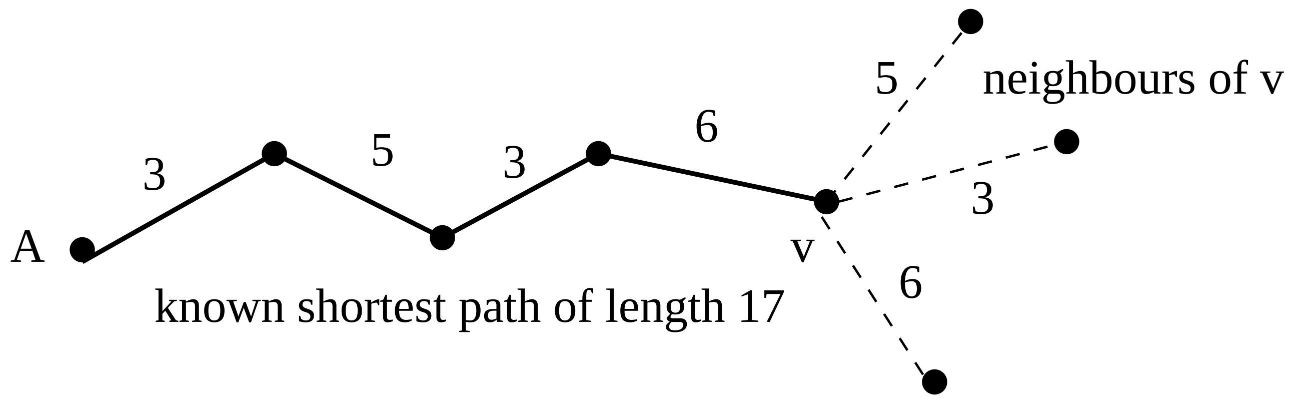
Main idea: Maintain a list S of nodes for which the shortest path is already known. (Initially, $S = \oslash$)

Maintain a list L which contains for each node the path length from A **known at this time**.
$L(u) =$ path length from A to u known at this time.
(Initially, $L(A) = 0$ and $L(u) = \infty$ for any other node.)

At each phase of the algorithm, find in L node v such that
$L(v)$ is the **smallest value** for nodes **not in** S.

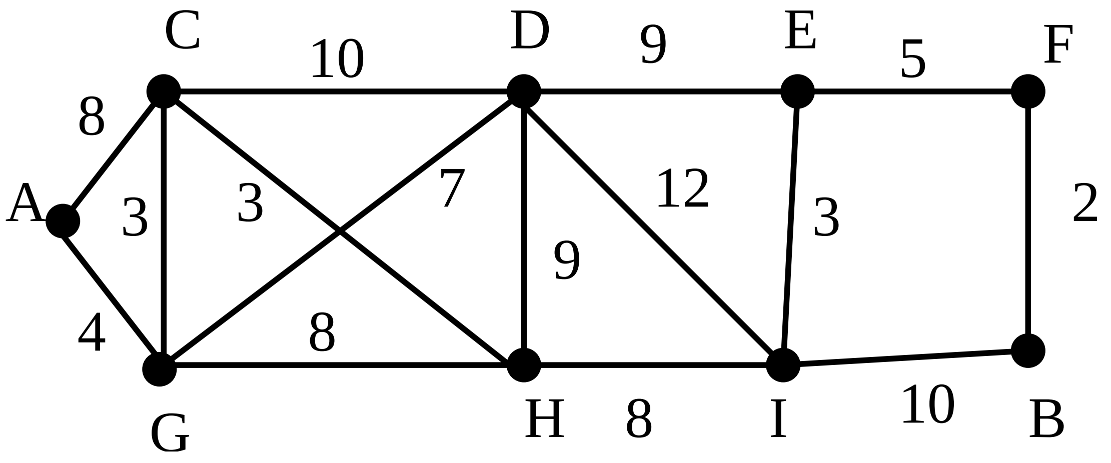
This $L(v)$ is the length of a **shortest path** from A to v. Thus we **add** v S.

Since $L(v)$ is the length of a shortest path from a A, a shortest path form A to the neighbors of $L(v)$ could be obtained by extending this path by edges incident to v.



If any of these extensions produces a path shorter than the known one in L,
update the information in L.

Then repeat the phase again if $S \neq V$.



	A	B	C	D	E	F	G	H	I	S
L_1	**0**	∞	∞	∞	∞	∞	∞	∞	∞	$\emptyset$
L_2	**0**	∞	8	∞	∞	∞	**4**	∞	∞	$\{A\}$
L_3	**0**	∞	**7**	∞	∞	∞	**4**	12	∞	$\{A, G\}$
L_4	**0**	∞	**7**	11	∞	∞	**4**	**10**	∞	$\{A, G, C\}$
L_5	**0**	∞	**7**	**11**	∞	∞	**4**	**10**	18	$\{A, G, C, H\}$
L_6	**0**	∞	**7**	**11**	20	∞	**4**	**10**	**18**	$\{A, G, C, H, D\}$
L_7	**0**	28	**7**	**11**	**20**	∞	**4**	**10**	**18**	$\{A, G, C, H, D, I\}$
L_8	**0**	28	**7**	**11**	**20**	**25**	**4**	**10**	**18**	$\{A, G, C, H, D, I, E\}$
L_9	**0**	**27**	**7**	**11**	**20**	**25**	**4**	**10**	**18**	$\{A, G, C, H, D, I, E, F\}$

Dijkstra's Algorithm for a path between A and B in a graph with nodes V. The $cost(uv)$ is the value associated with edge uv.

```
for $(i = 1; i <= n; i++)$ initialize the lists
   $L(v_i) = maxint$;
$L(A) = 0$;
$S = \oslash$;
while $B \notin S$ do
    $\{u =$ node in $V - S$ with minimal $L$ value;
     $S = S \cup \{u\}$; //we have shortest path to $u$
     for every $v$ adjacent to $u$
      if $(L(u) + cost(uv) < L(v))$
        $L(v) = L(u) + cost(u, v)$; //update list L
    $\}$
```

The number of operations needed in the Dijkstra's algorithm is cn^2 for some constant c.

A very common case in many applications (e.g., computer networks), graph cost of each edge is taken as 1.

Dijkstra's Algorithm then usually produces several nodes with the minimal cost in each phase and the number of phases is smaller.

Theorem 24 *Given a graph G and two nodes in the same component of G, the Dijkstra's algorithm calculates the shortest part between these nodes.*

Proof : Done in class

This is not a greedy algorithm. A greedy algorithm would not find a shortest path in some cases.

Sometimes we need to find, given a node A of a graph G, all the nodes in the same component as A.

Dijkstra's algorithm can be modified to calculate the component of the given graph containing A.

In this case the costs of edges are not important and we count each edge being of cost 1.

We keep for each node whether it belongs to the component.

Calculating the component containing a given node A in a graph with n nodes.

```
for $(i = 1; i <= n; i++)$
    $belong(v_i) = false$; initialize marks
$S[1] = A$; S is a list of nodes in the component
$belong(A) = true$; mark it is in the component
$i = 1$; // number of nodes in the component
$j = 1$; // which node is processed
while $(j <= i)$ do
     {for (every $v$ adjacent to $S[j]$)
          if $(belong(v) == false)$
               {$S[++i] = v$; //add it to the list
                $belong(v) = true$; mark it.
               }
      $j++$;
     }
//value of $i$ gives the size of the component, $S$ its nodes.
```

This algorithm finds the nodes of a component containing A in the order of the distance of the nodes from A, starting with nodes that are closest to A.

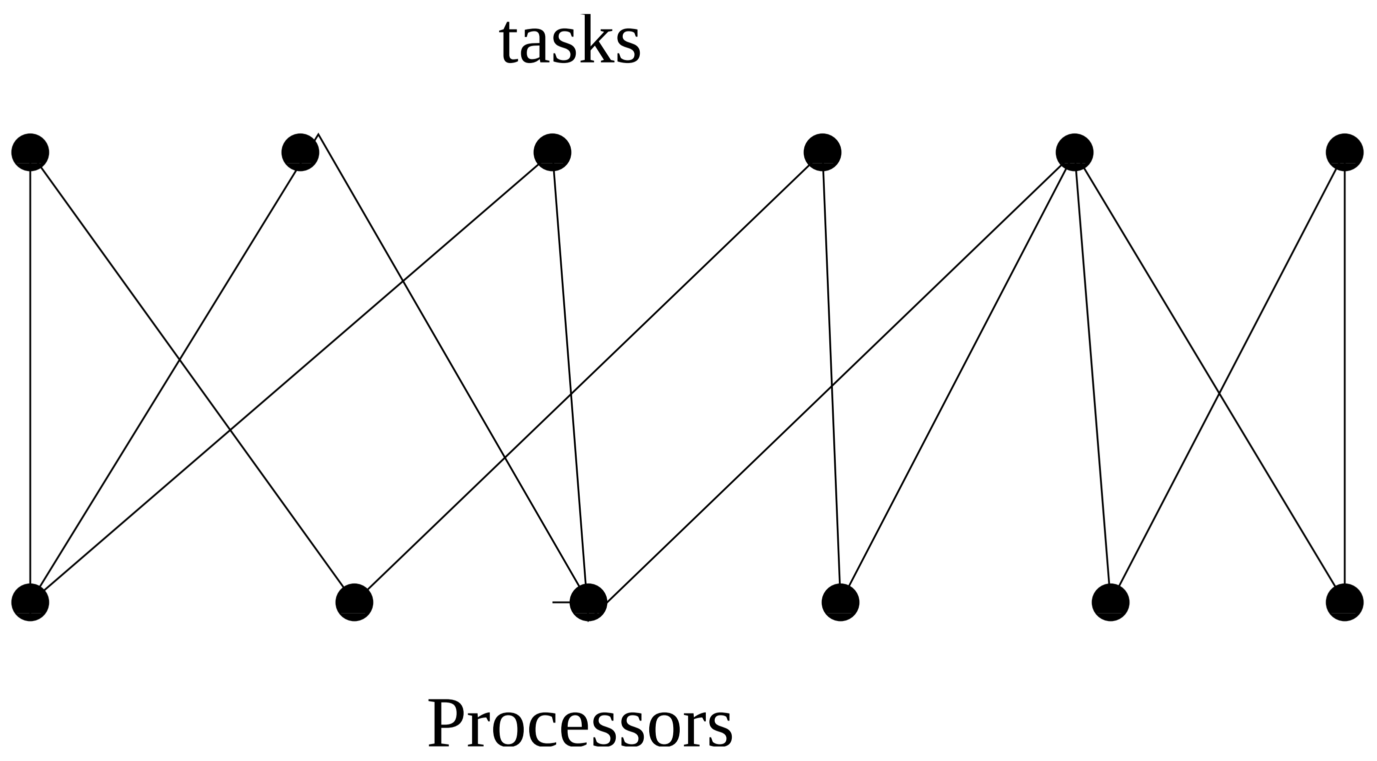
This type of processing of nodes is called **breadth-first** order.

More on this is seen in the course on Data Structures and Algorithms.

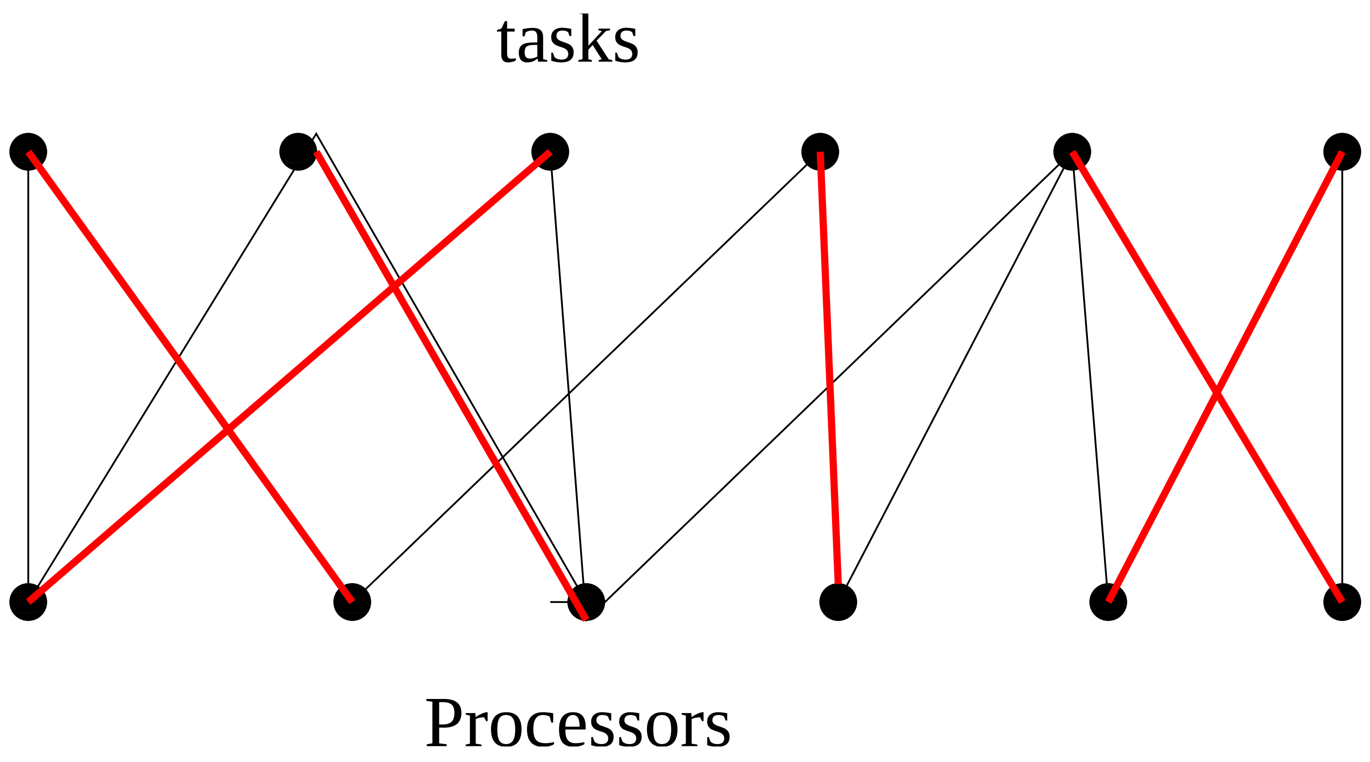
There are many other important problems that can be represented by a graph and solved by graph algorithms.

Perfect Matching

Given some tasks and processors and an indication on which processors are best suited for each task. find the perfect matching of tasks and processors.



Red edges indicate a possible solution:



There is an algorithm for this problem.

Scheduling

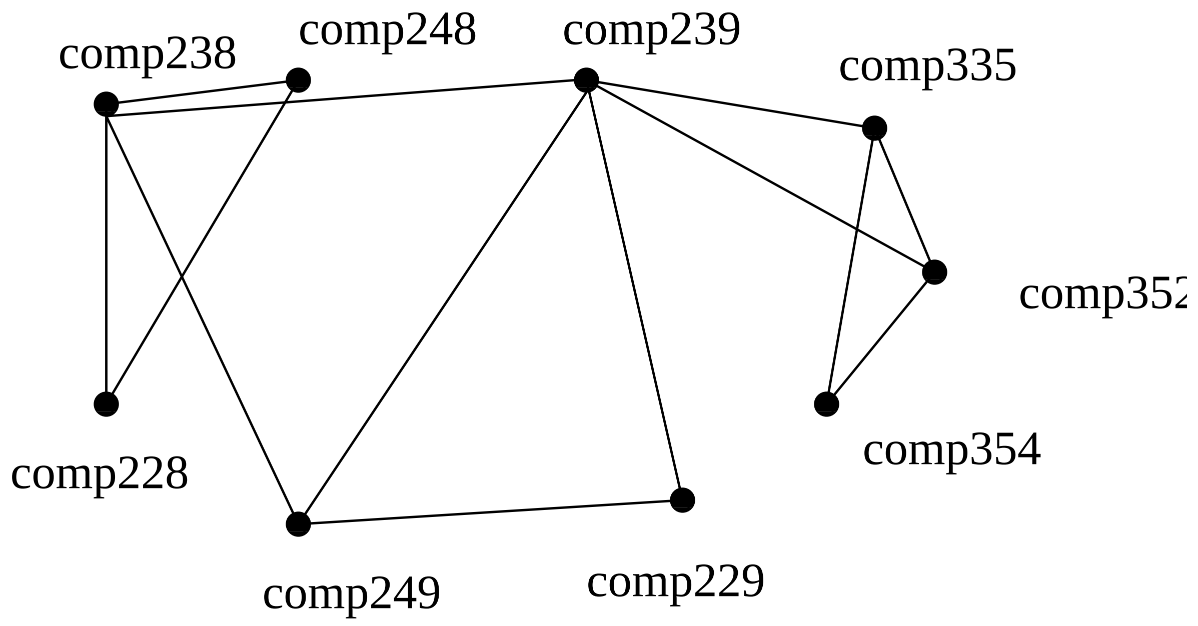
We are are to design a schedule of courses so that:

- any two courses usually taken by same students are scheduled at different times,

- any two courses taught by the same professor are scheduled at different times,

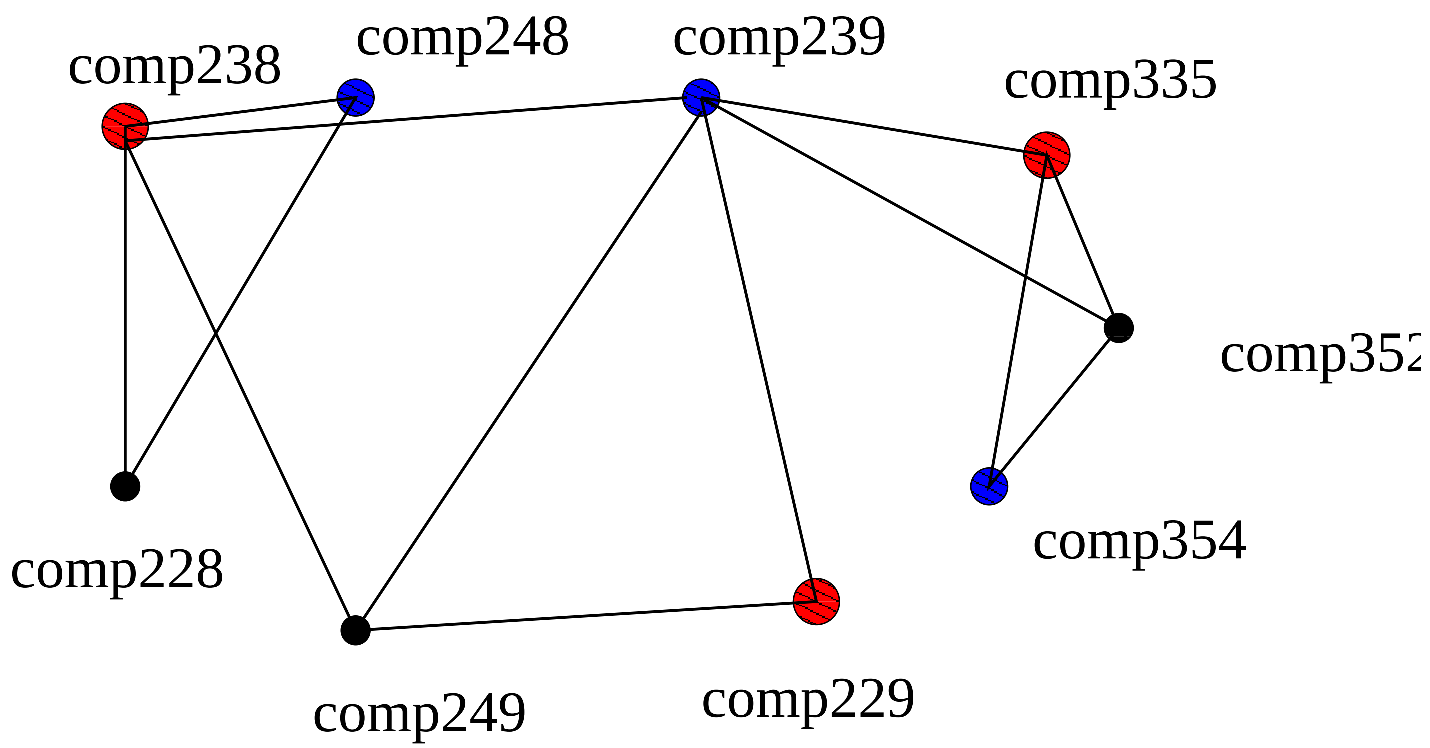
- we use the smallest number of time-slots.

This can be represented by a graph.



Solution: Color the nodes of the graph so that no two nodes of the same color are connected. Each color represents a different time-slot.

Color using the minimum number of colors.



The **graph coloring** problem is a difficult one (similarly to the travelling salesman problem)

There are some good approximation algorithms for the coloring of graphs.

There are many other problems that can be

- represented by graphs and
- and solved by graphs

There is a large number of algorithms to solve graph problem.

We don't have time to discuss them.

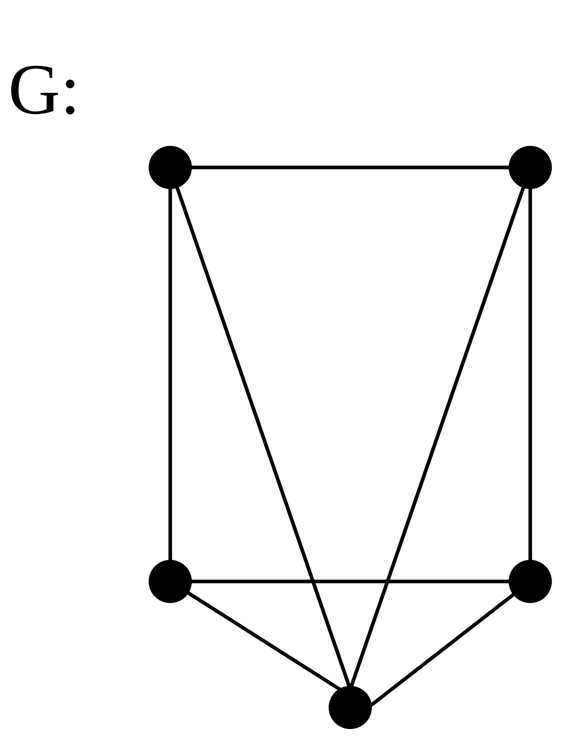
Planar Graphs

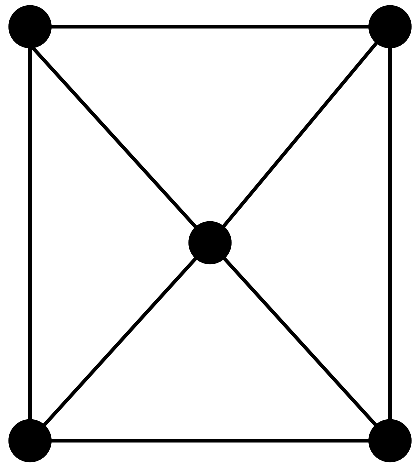
A graph is called **planar** if we can draw nodes and edges of the graph in the plane so that no two edges of the graph cross each other.

A **planar map** of a graph is a drawing of the graph where no two edges of the graph cross each other.

Planar graphs have some important properties, for that reason they are studied in more detail.

Graph G and a planar map of G:

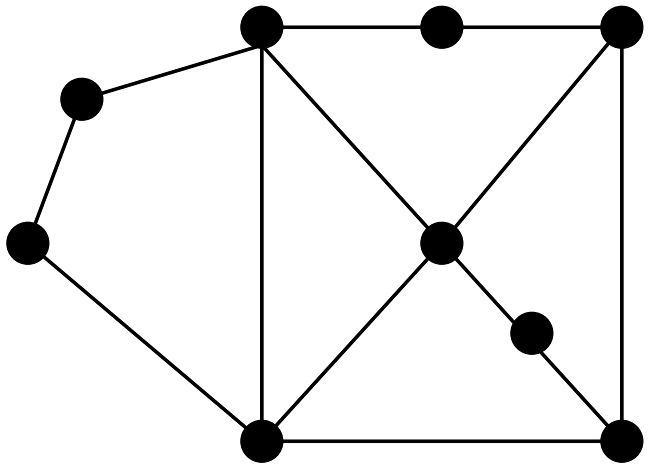




Since there is a planar map of G, it is a planar graph.

A planar map divides the plane into separate regions, called **faces**. (Think of it as countries).

Outer face is the region that “surrounds” the graph.



The map above has 6 faces.

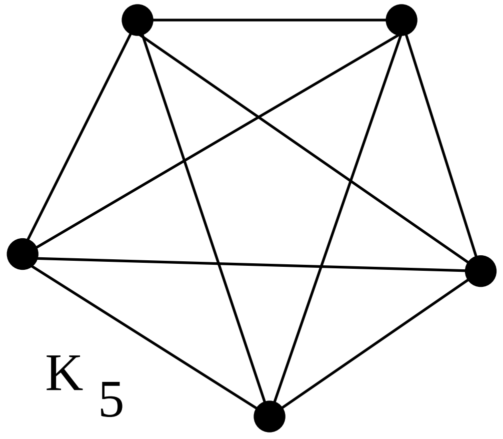
Theorem 25 *Let G be a connected planar map that contains n nodes, e edges and f faces.*

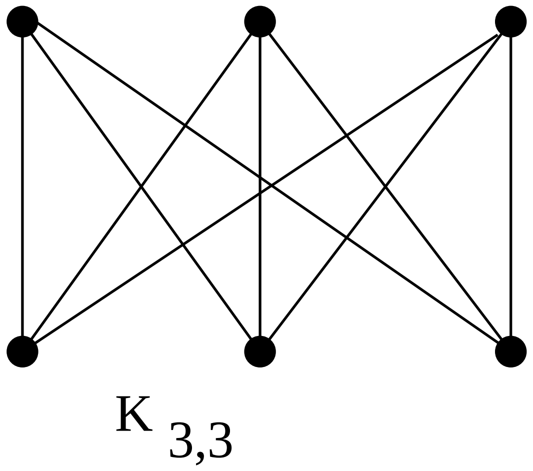
$$f + v = e + 2$$

Proof done in class.

Theorem 26 *The complete graph K_5 on five nodes is not a planar graph.*
The graph $K_{3,3}$ on 6 nodes below is not a planar graph.

Proof done in class.





Theorem 27 *A planar graph on n nodes has at most* $3n-6$ *edges.*

Proof done in class.

Thus in a planar graph, similarly as in a tree, the number of edges is linearly proportional to the number of nodes.

Theorem 28 *A graph is planar if and only if it contains no subdivision of K_5 or $K_{3,3}$.*

This very interesting theorem is due to Kuratowski (1930)

However, it does not give us a nice algorithm for testing whether a given graph is planar.

There are several algorithms that can decide, for a given graph G,

- whether G is planar and
- how to place the nodes in the plane and draw the edges to get a planar map of the graph.

(Demoucron, Malgrange, Pertuiset, (1965)
Hopcroft and Tarjan, (1974))

These algorithms are quite efficient.

Directed Graphs

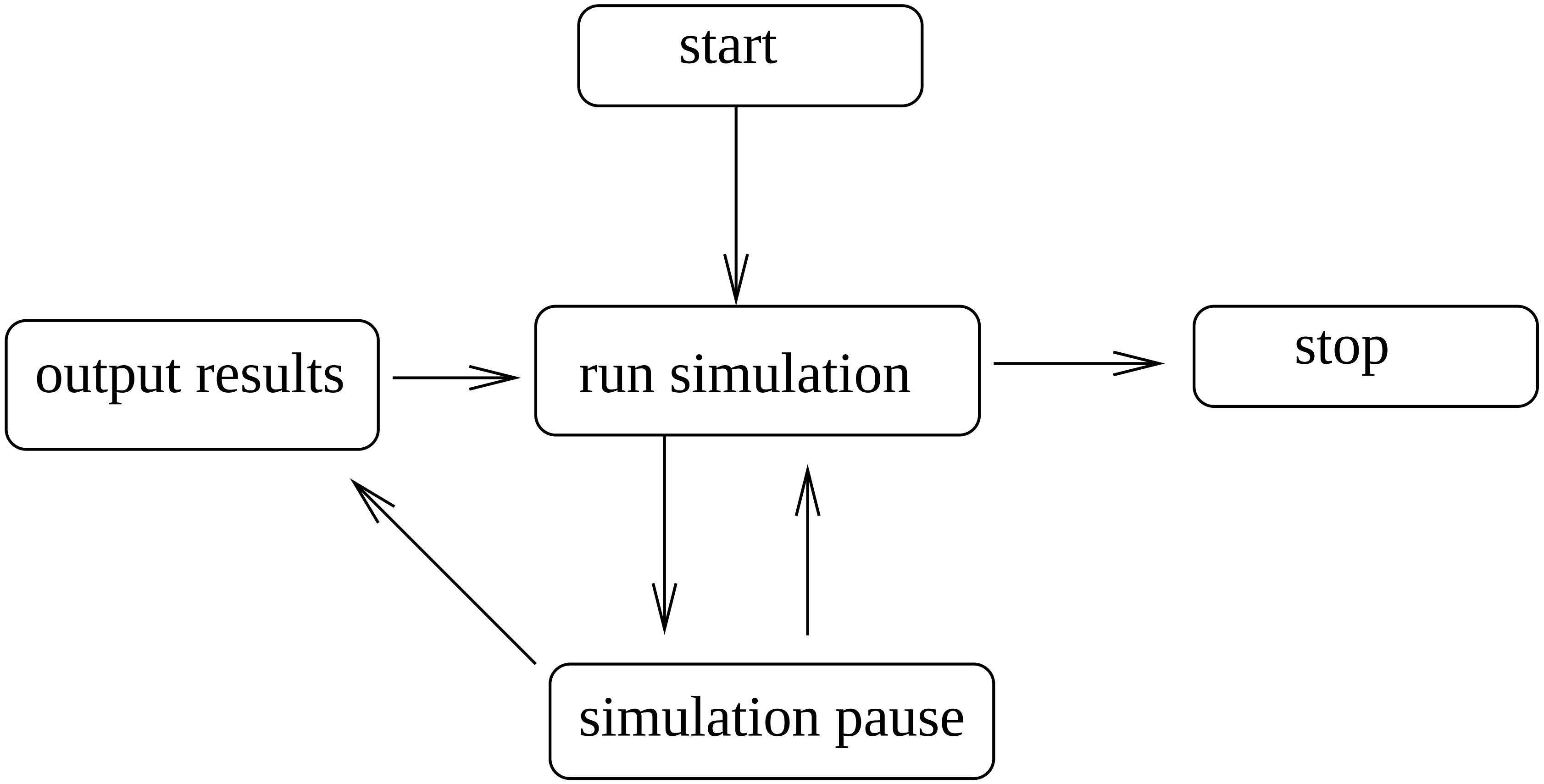
Graphs that we discussed so far were **simple graphs**. In a simple graph edge uv is the same as edge vu.

Such a graph is often called **undirected**.

In some applications we have situations when we can go from node u to node v, but not from v to u. (one-way street for cars))

In these applications we use a **directed graph** to model the traffic.

Example of a directed graph representing the behavior of a system:



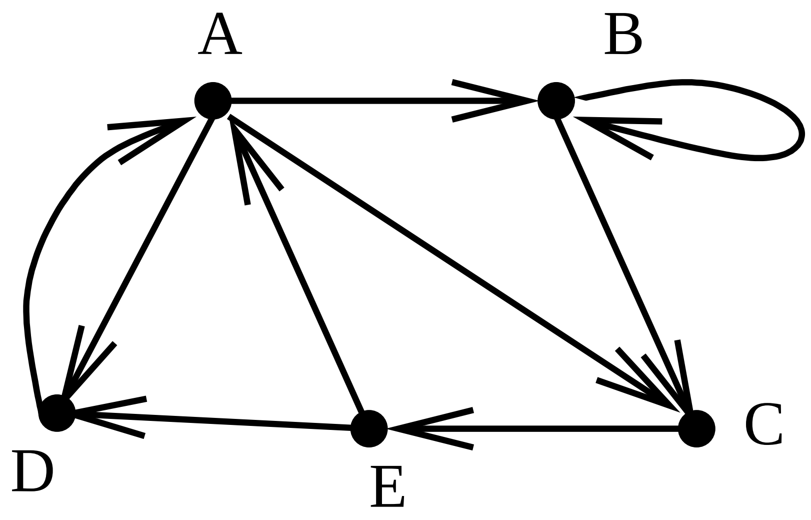
A **directed** graph consists of a set of **nodes** and a set OF **directed edges**, each edge being an **ordered pair** of nodes.

The set of nodes of a graph G is often denoted by V and the set of edges by E.

So we write $G = (V, E)$

A specification of a directed graph looks very much like a specification of an undirected graph. Thus we must alway specify the term directed when dealing with a directed graph.

Example of a directed graph:



Directed graph $G = (V, E)$ where $V = \{A, B, C, D, E\}$ and $E = \{AB, AD, AC, EA, BB, BC, CE, DA, ED\}$.

In a directed graph, **loop edges** are often allowed.

Most of the terms introduced for undirected graphs are also defined with appropriate modifications for directed graph.

For a directed edge uv we say that u is its **initial node** and v is its **terminal node**.

in-degree of a node is the number of edges that terminate in it,
out-degree of a node is the number of edges that have the node as its initial node.

Directed path: a sequence of distinct nodes in which there is a directed edge from each node of the sequence to the next node of the sequence.

Directed cycle: a sequence of distinct nodes in which there is a directed edge from each node of the sequence to the next node of the sequence and from the last one to the first one.

Directed walk: a sequence of nodes (not necessarily distinct) in which there is a directed edge from each node of the sequence to the next node of the sequence.

We have an obvious generalization of a result for undirected graphs:

Theorem 29 *In a directed graph $G = (V, E)$ the sum of the in-degrees of all the nodes is equal to the sum of the out-degrees of all the nodes and is equal $|E|$.*

Dijkstra's algorithm can be applied to directed graph with little modifications.

There are some useful algorithms dealing exclusively with directed graphs, e.g **maximal flow** algorithm.

Directed graphs have been used for a specification and a construction of simple computational devices and algorithms.

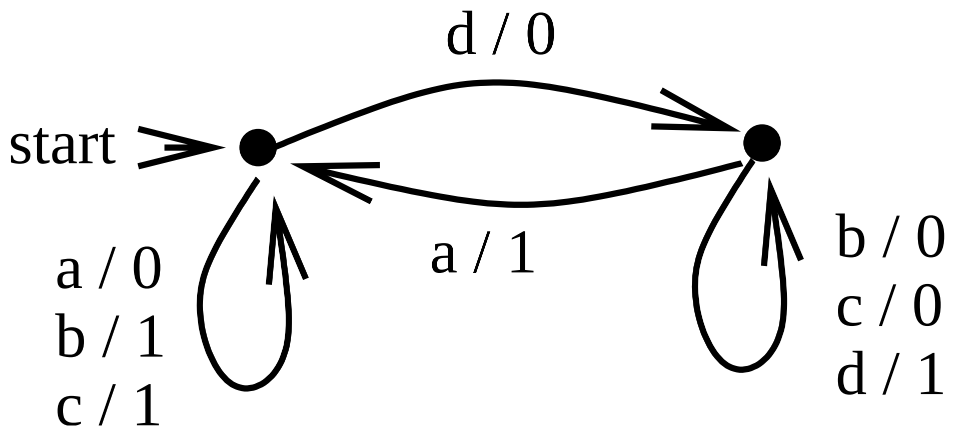
They are used in the design of computer chips.

A **finite state machine** is a directed graph in which

- each edge is labeled by an input and an output symbol (/ separates input and output symbols), and
- one state is designated as the state where the computation starts.

For a given input string $a_1 a_2 \cdots a_k$, the computation follows the directed walk from the start state in which the ith edge of the walk is labeled by a_i.

The output of the computation is the string $b_1 b_2 \cdots b_k$ where b_i is the output label of the ith edge of the walk.



input: aabcdbdaba

output: 0011001110

The directed graph is an algorithm for adding two binary numbers.

a represents 00

b represents 01

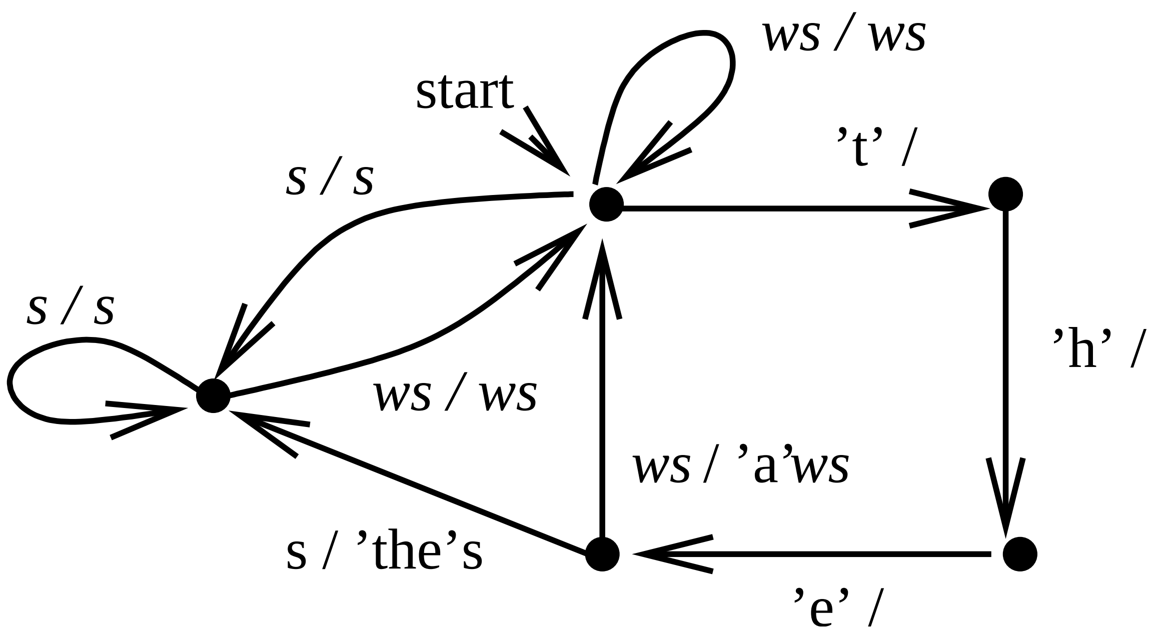
c represents 10

d represents 11

The adding is done by starting with the least significant digits of the two numbers.

An algorithm for scanning a text and replacing each preposition “the” by “a” is represented by a directed graph below.

s represents any character, ws is a blank or EOL, $s -' t'$ is any character different from $'t'$.



The two examples given here are meant to illustrate the use of directed graph for representing computations.

The course on theoretical computer science uses this type of representation extensively.