Chapter 2: Graph and network optimization

Many decision-making problems can be formulated in terms of graphs and networks.

Examples:

- transportation and distribution problems,
- network design (communication, electrical,..),
- location problems (services and facilities),
- project planning, resource management,
- timetable scheduling,
- production planning,...

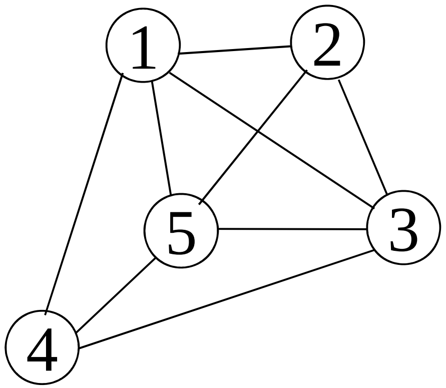
2.1 Graphs and algorithms

2.1.1 Graphs

Example

Road network which connects *n* cities

n=5



city ↔ node

connection ↔ edge

Model:

A ***graph*** $G = (N, E)$ which consists of a set $N=\{1,2,3,4,5\}$ of *nodes* (*vertices*) and a set $E=\{[1,2],[1,3],[1,4],[1,5],[2,3],[2,5],[3,4],[3,5],[4,5]\} \subseteq N \times N$ of *edges* connecting them.

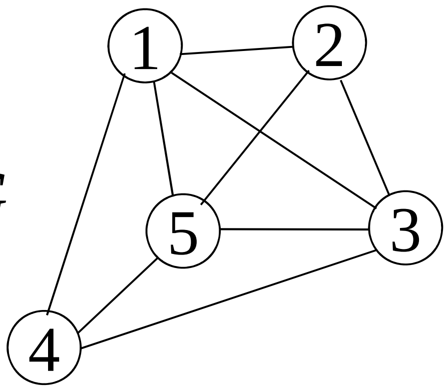
[,] indicates an unordered pair of nodes

Definitions

Two nodes are ***adjacent*** if they are connected by an edge.

An edge e is ***incident*** in a node v if v is an endpoint of e.

G



nodes 1 and 2 are adjacent

edge [1,5] is incident in nodes 1 and 5

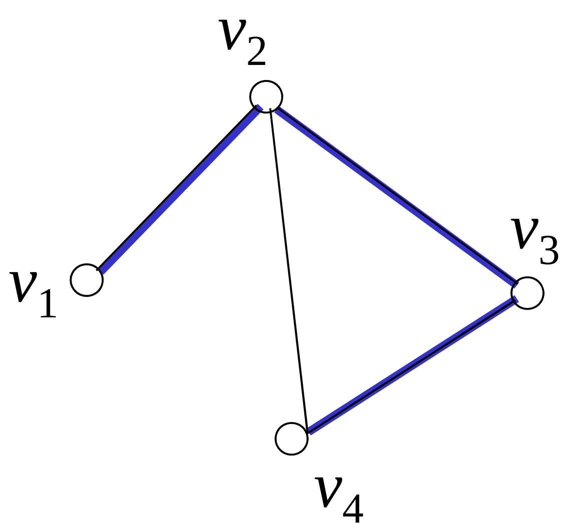
The ***degree*** of a node is the number of incident edges.

Example: node 1 has degree 4, node 4 has degree 3.

Given a graph $G=(N, E)$ with $n = |N|$ and $m = |E|$

$n=4$

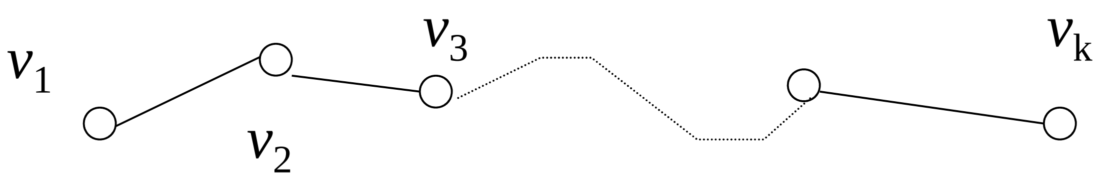
$m=4$



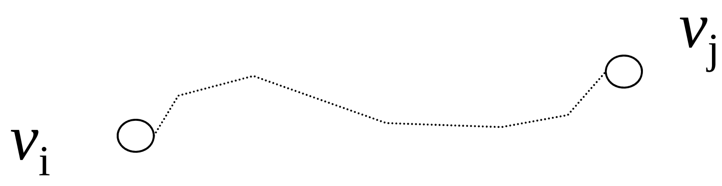
$N = \{ v_1 , v_2 , v_3 , v_4 \}$

$E = \{ [v_1, v_2], [v_2, v_3], [v_2, v_4], [v_3, v_4] \}$

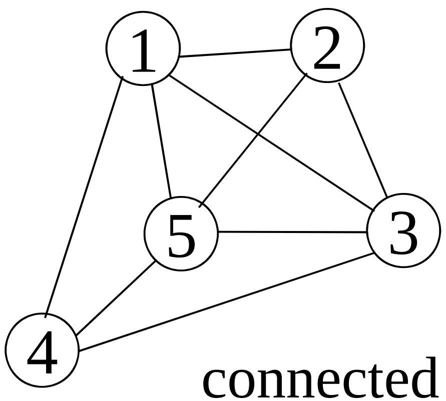
A sequence of consecutive edges $[v_1, v_2], [v_2, v_3], \ldots, [v_{k-1}, v_k]$ is a *path* which connects nodes v_1 and v_k



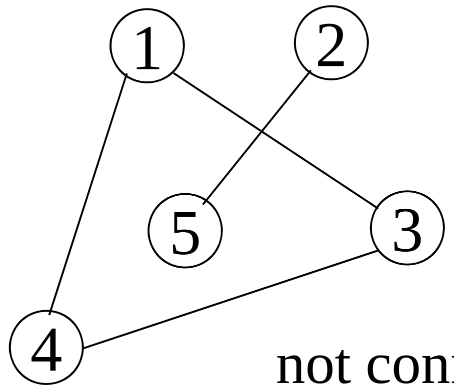
$v_i, v_j \in N$ are ***connected*** if there exists a path connecting them



$G = (N, E)$ is ***connected*** if v_i, v_j are connected $\forall\, v_i, v_j \in N$



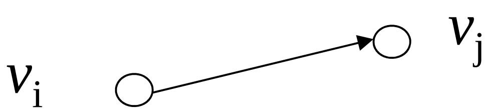
connected



not connected

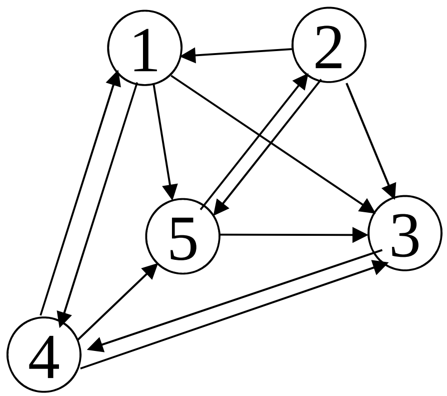
If some connections can be travelled only in one direction:

__Directed graph__ $G = (N, A)$, where A is a set of ordered pairs of nodes (v_i, v_j) called *__arcs__*

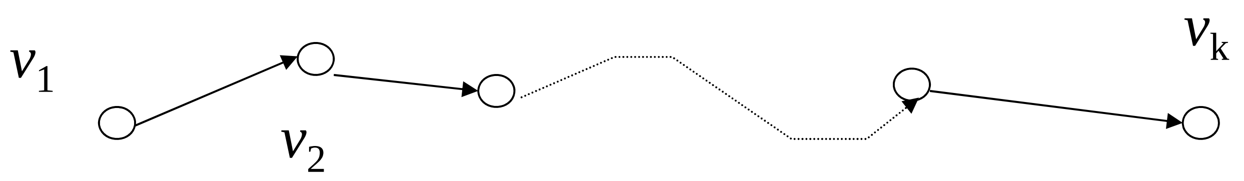


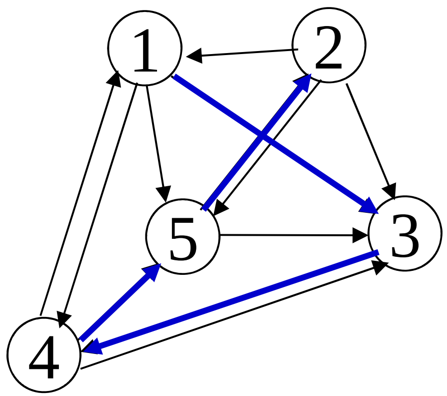
$(v_i, v_j) \in A$

G



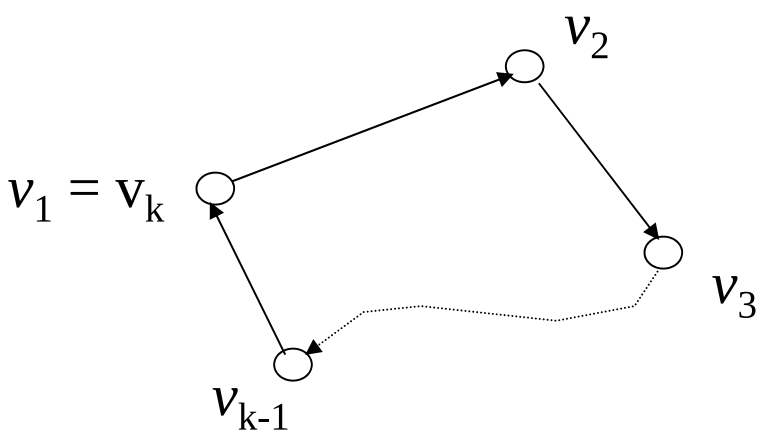
A sequence of consecutive arcs (v_1, v_2), (v_2, v_3),…, (v_{k-1}, v_k) is a ***directed path*** from v_1 to v_k

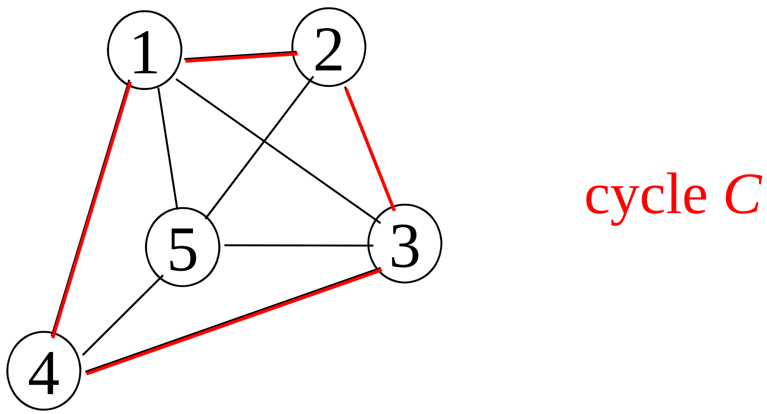




directed path from 1 to 2

A *cycle* (*circuit*) is a (directed) path with $v_k = v_1$

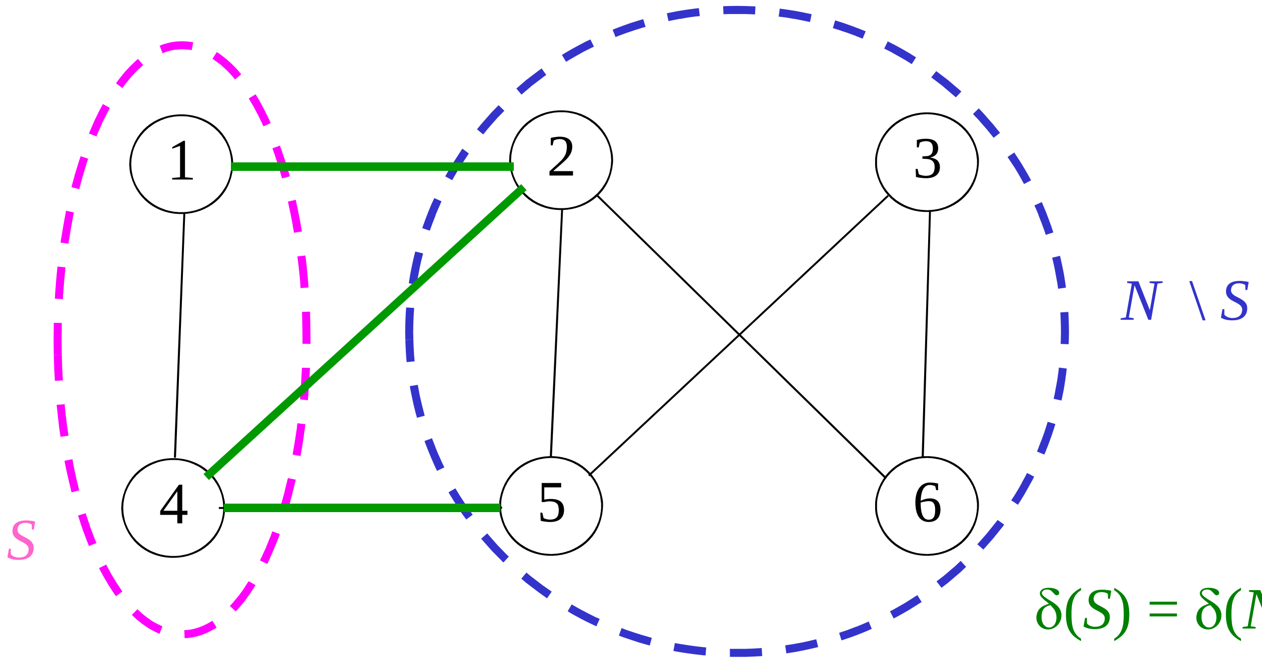




Given an undirected G and a subset of nodes $S \subset N$, the ***cut*** induced by S denoted by $\delta(S)$, is the subset of edges with an endpoint in S and the other endpoint in $N \setminus S$.

$$\delta(S) = \{ [v,w] \in E : v \in S,\ w \in N \setminus S \text{ or } w \in S,\ v \in N \setminus S \}$$

Example



$\delta(S) = \delta(N \setminus S)$

Given directed $G = (N, A)$ and a subset of nodes $S \subset N$,

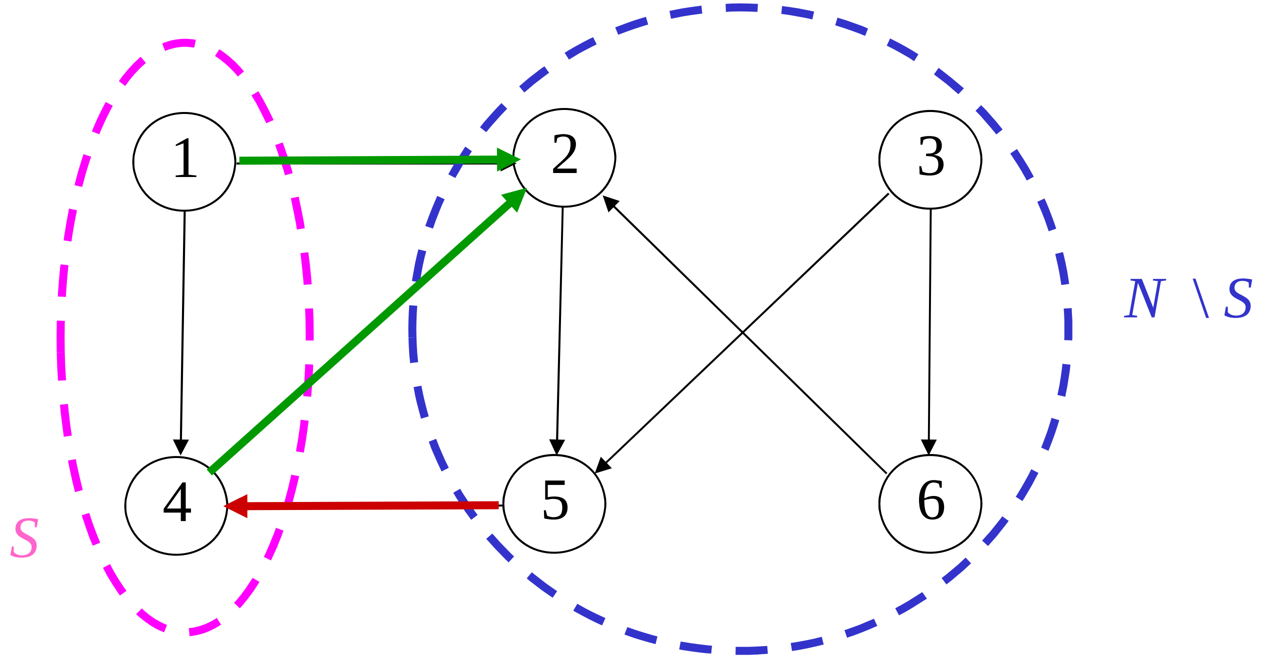
the *outgoing cut* induced by S :

$$\delta^+(S) = \{ (v, w) \in A : v \in S, w \in N \setminus S \}$$

the *incoming cut* induced by S :

$$\delta^-(S) = \{ (v, w) \in A : w \in S, v \in N \setminus S \}$$

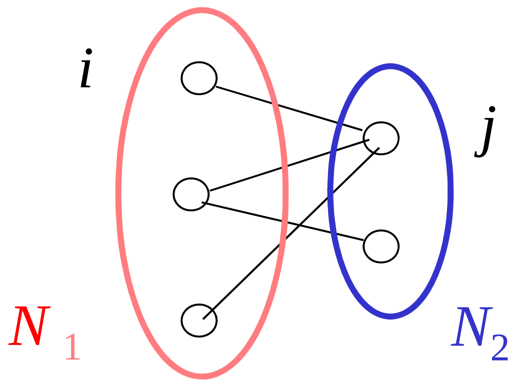
Example



Model for (in)compatibility relations

Example

3 tasks and 2 engineers

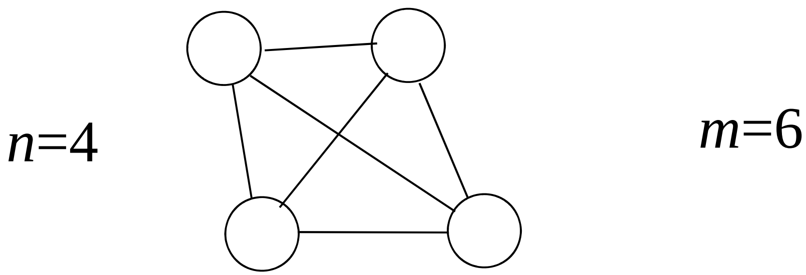


$N_2 = N \setminus N_1$

edge $[i, j]$ indicates that task i can be executed by engineer j

Definition: G is *bipartite* if there exists a partition (N_1, N_2) of N such that no edge connects nodes in the same N_i $(i = 1, 2)$.

Defintion: G *is* <u>*complete*</u> if $E = \{ [v_i, v_j] : v_i, v_j \in N, i \leq j \}$.



Property

For any graph G with n nodes, the number of edges satisfies:

- $m \leq \frac{n(n-1)}{2}$ if G undirected
- $m \leq n(n\text{-}1)$ if G directed.

In both cases, we have equality for complete graphs.

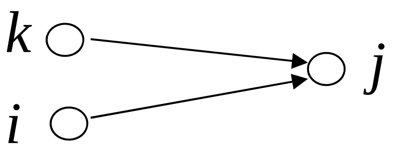
Model for precedence constraints between entities

A project is composed of n activities $\{a_i\}_{1 \leq i \leq n}$ with m precedence relations between pairs of activities $a_i \prec a_j$ (a_j cannot start before a_i is completed).

<u>Model 1</u>: directed graph

node $\leftrightarrow$ activity

arc $\leftrightarrow$ precedence



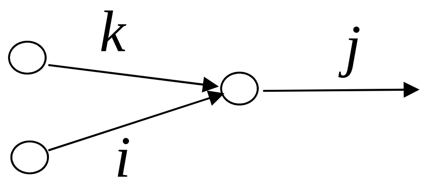
$a_i \prec a_j$

$a_k \prec a_j$

<u>Model 2</u>: directed graph

arc $\leftrightarrow$ activity

node $\leftrightarrow$ outgoing activities can start when all incoming activities are completed



$a_i \prec a_j$

$a_k \prec a_j$

Graph representation

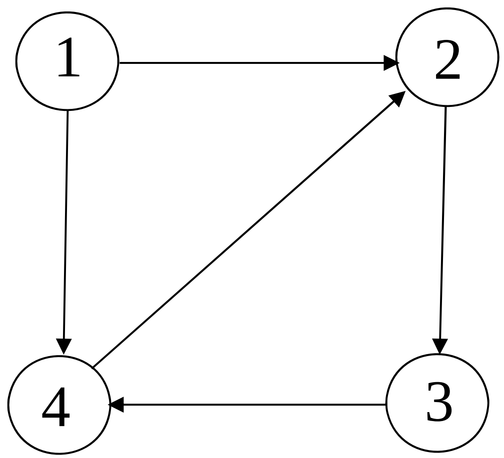
A graph with n nodes and m arcs is *dense* if $m \approx n^2$ and *sparse* if $m \ll n^2$.

Definitions: (similar for undirected graphs)

- For dense directed graphs, $n \times n$ *adjacency matrix* :

$$a_{ij} = 1 \text{ if } (i,j) \in A \text{ and } a_{ij} = 0 \text{ otherwise.}$$

- For sparse directed graphs, *list of successors* or predecessors



$S(1) = \{ 2, 4 \}$

$S(2) = \{ 3 \}$

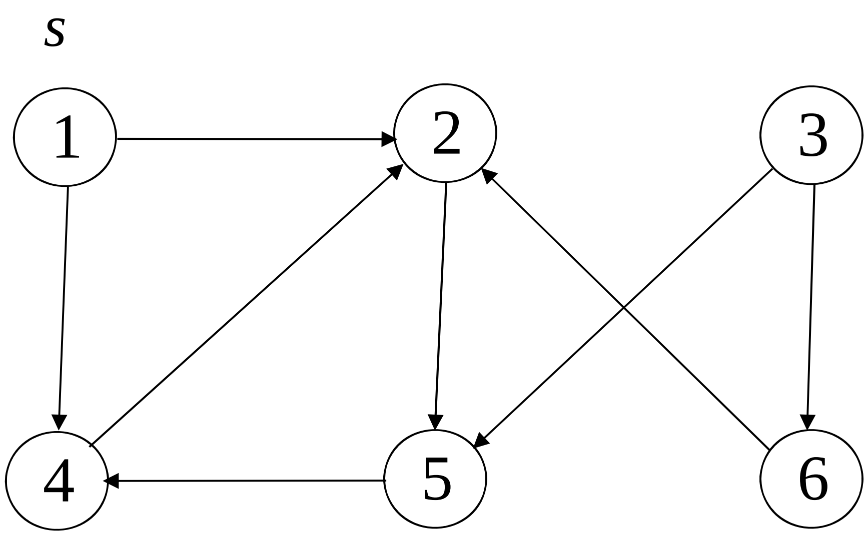
$S(3) = \{ 4 \}$

$S(4) = \{ 2 \}$

2.1.2 Graph reachability problem

Problem

Given a directed graph $G = (N, A)$ and a node s, determine all the nodes that are reachable from s.



Successor lists :

$S(1) = \{ 2, 4 \}$

$S(2) = \{ 5 \}$

$S(3) = \{ 5, 6 \}$

$S(4) = \{ 2 \}$

$S(5) = \{ 4 \}$

$S(6) = \{ 2 \}$

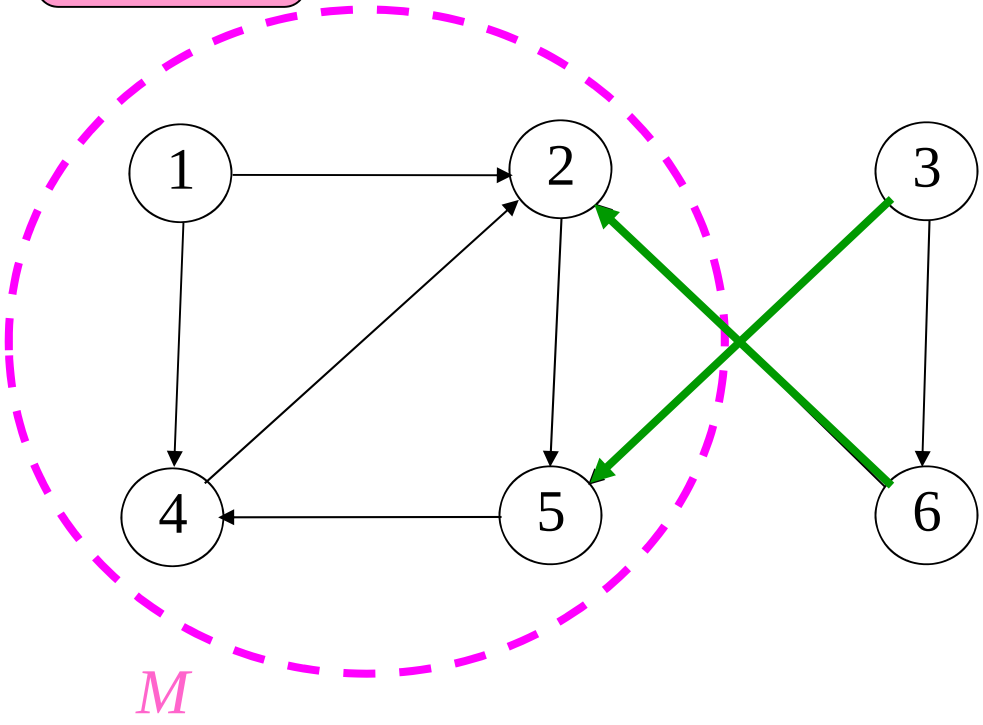
Devise an (efficient) algorithm that allows to find all nodes reachable from s.

input $G = (N, A)$ with $n = |N|$ and $m = |A|$, described by the successor lists, and a node s.

output Subset $M \subseteq N$ of nodes of G reachable from s.

We use a "queue" Q containing the nodes reachable from s and not yet processed (First-In First-Out policy).

Example



$Q = \{ 1 \}$ and $M = \emptyset$

$Q = \{ \cancel{1} \}$ and $M = \{1\}$

$Q = \{ \cancel{2}, 4 \}$ and $M = M \cup \{2\}$

$Q = \{ \cancel{4}, 5 \}$ and $M = M \cup \{4\}$

$Q = \{ \cancel{5} \}$ and $M = M \cup \{5\}$

$Q = \emptyset$

Subset $M = \{ 1, 2, 4, 5 \}$ of nodes that have been labeled is the subset of nodes reachable from $s = 1$.

Observation: No arcs exit M and enter $N \setminus M$!

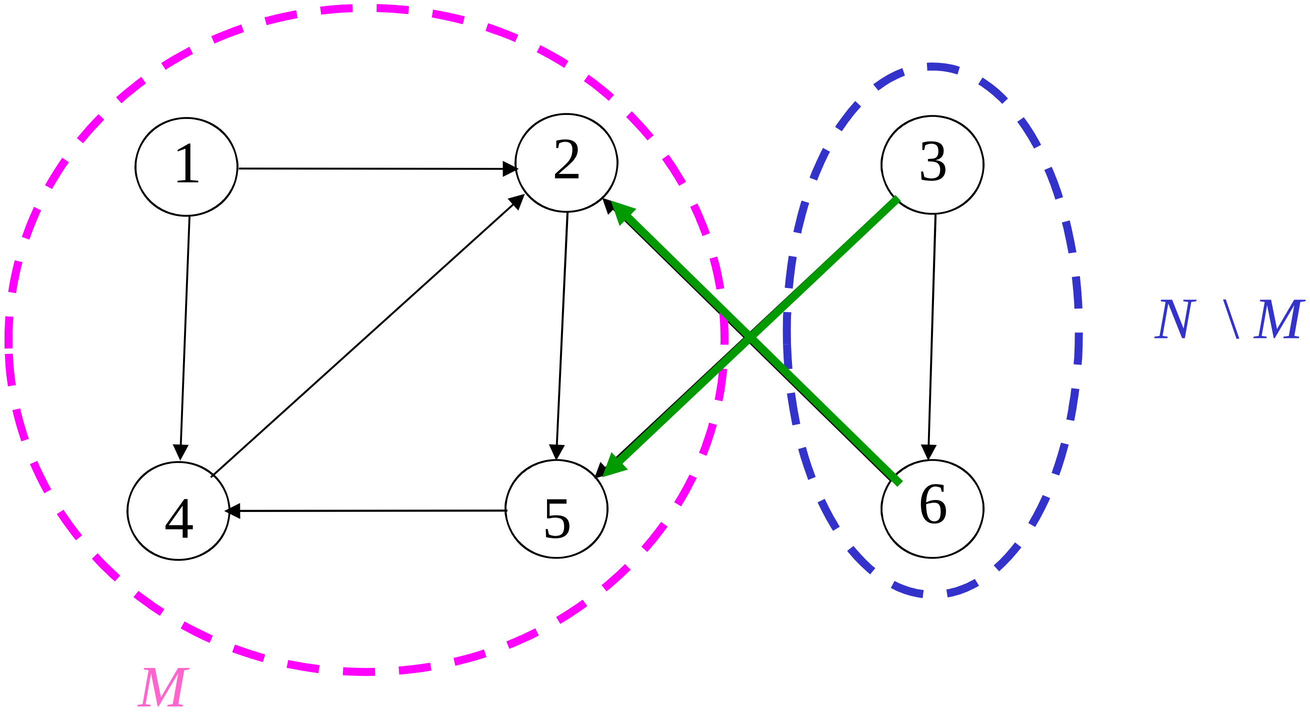
Pseudocode for the graph reachability algorithm

output Subset of nodes M (reachable from s)

```
BEGIN
 Q := {s}; M := ∅;
   WHILE Q ≠ ∅ DO   /* process a node h ∈ Q */
     Select a node h ∈ Q e set Q := Q \ {h};
     M := M ∪ {h};    /* label h */
     FOR EACH j ∈ S(h) DO
       IF j ∉ M AND j ∉ Q THEN  Q := Q ∪ {j} END-IF
     END-FOR
   END-WHILE
 END
```

FIFO queue $Q \Rightarrow$ breadth-first search node exploration.

Example



The algorithm (exploration) stops because $\delta^+(M)=\emptyset$,

$\delta^-(M)$ is the set of arcs with head in M and tail not in M.

Observation: $\delta^+(M)=\emptyset$ certifies that the algorithm is correct.

2.1.3 Complexity of algorithms

Definition: An *algorithm* for a problem is a sequence of instructions that allows to solve any of its instances.

The execution time of an algorithm depends on

- the instance
- the computer.

We want to evaluate the complexity of the algorithm as a function of the *size of the instance* (e.g., n or m) independently from the hardware.

Therefore we consider the number of *elementary operations* (e.g., arithmetic operations, comparisons, memory accesses...)

we assume they all have the same cost

Examples:

1) Dot product of $\underline{a}$, $\underline{b} \in \mathbb{R}^n$ requires n multiplications and n-1 additions
$\Rightarrow$ $2n$-1 elementary operations.

2) Given two nxn matices A and B, the product AB requires $(2n-1)n^2$ elementary operations.

Since it is usually hard to determine the exact number of elementary operations (as a function of the instance size), we consider

the asymptotic number of elementary operations (speed of growth) in the worst case (for the worst instances).

We look for a function $f(n)$ which is (asymptotically) an upper bound on the number of elementary operations needed to solve every instance of size at most n.

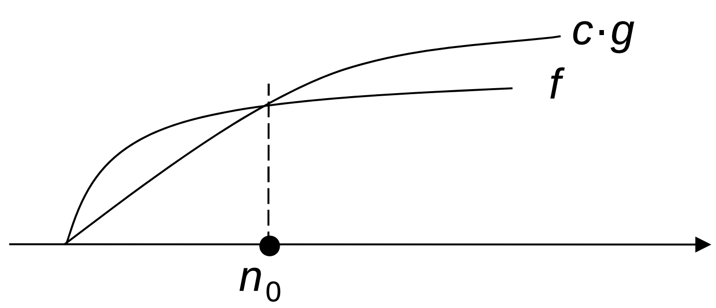
Big-O notation

Definition: A function $f(n)$ is *order of* $g(n)$ and we write

$$f(n) = O(\, g(n)\,)$$

if $\exists\, c > 0$ such that $f(n) \leq c\, g(n)$, for n sufficiently large.

asymptotically



Examples

$$3n^3 + n^2 + 10 \quad = O(n^3)$$

$$m = n(n\text{-}1)/2 \quad = O(n^2)$$

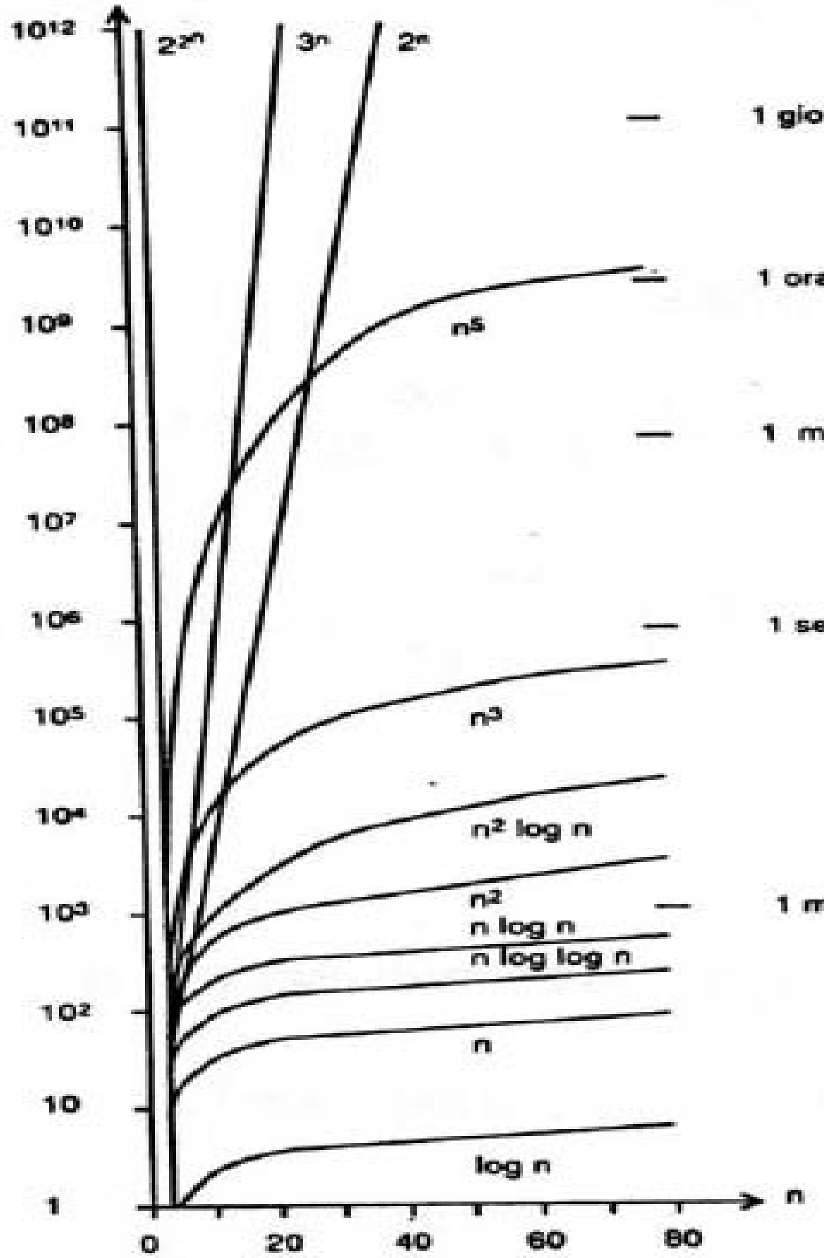
We distinguish between algorithms whose order of complexity (in the worst case) is

- *polynomial*: $O(n^d)$ for a given constant d

N.B.: The algorithms with a higher order polynomial complexity (such as $O(n^8)$) are not efficient in practice!

- *exponential*: $O(2^n)$

Polynomial versus exponential growth



Assume a 1 microsecond is needed per elementary operation

$\|I\|$	$f(\|I\|) = \|I\|^2$	$f(\|I\|) = 2^{\|I\|}$
1	0.000001 secondi	0.000002 secondi
10	0.0001 secondi	0.001 secondi
20	0.0004 secondi	1 secondi
30	0.0009 secondi	17.9 minuti
40	0.0016 secondi	12.7 giorni
50	0.0025 secondi	35.7 anni
60	0.0036 secondi	366 secoli

Example: complexity of the reachability algorithm

At each iteration of the cycle `WHILE`:

- select one node $h \in Q$, extract it from Q and insert it in M,
- for all nodes j directly reachable from h and not already in M or Q, insert j in Q.

Since each node h is inserted in Q at most once and each arc (h, j) is considered at most once, we have

overall complexity $O(n + m)$, where $n = |N|$ and $m = |A|$.

Observation: for dense graphs $m = O(n^2)$

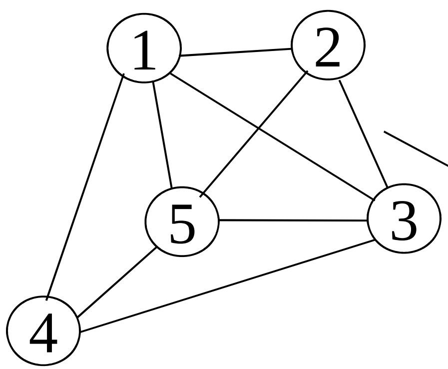
2.1.4 Subgraphs, trees and spanning trees

Example

Design a communication network that connects n cities.

Model: Undirected graph $G = (N, E)$ with $n = |N|$, $m = |E|$

$n=5$

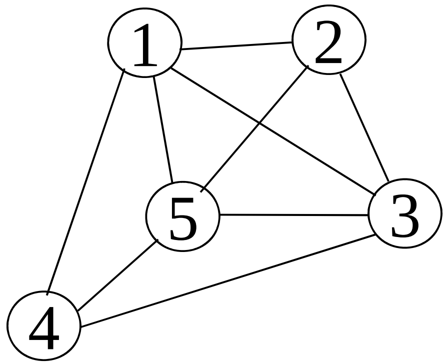


The edges represent the possible links

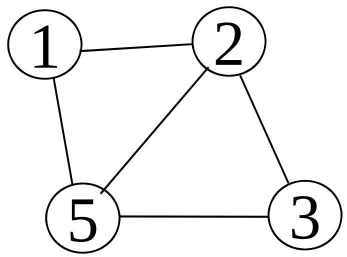
Definition: G' = (N', E') is a ***subgraph*** of G = (N, E) if

- $N' \subseteq N$
- $E' \subseteq E$ only contains edges with both endpoints in N'.

G=(N, E)



G'=(N', E')



$N' = \{1,2,3,5\} \subseteq N$

$E' = \{[1,2],[1,5],[2,3],[2,5],[3,5]\} \subseteq E$

Desired properties of a communication network:

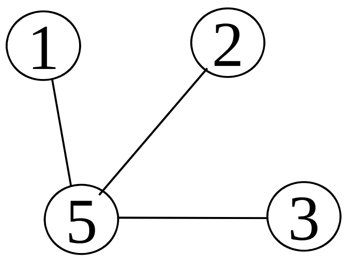
1) Since every pair of cities must be connected, $N' = N$ and G' must be a <u>connected subgraph</u> of G.

2) Since we do not want to waste resources, G' must be an <u>acyclic subgraph</u> (without cycles) of G.

Definitions:

- A <u>*tree*</u> $G_T = (N', T)$ of G is a subgraph of G that is both <u>connected</u> and <u>acyclic</u>.

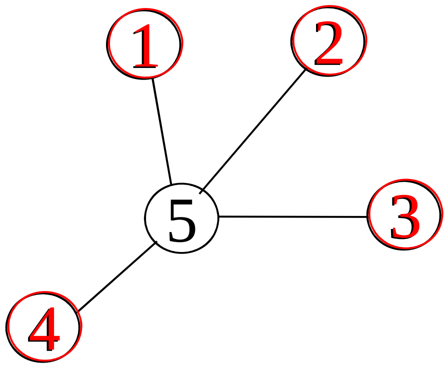
G_T



$T=\{[1,5],[2,5],[3,5]\}$

- $G_T = (N', T)$ is a *spanning tree* of $G = (N, E)$ if it contains all the nodes of G (namely $N' = N$).

G_T



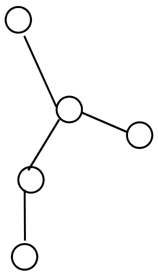
- The *leaves* of a tree are the nodes of degree 1.

Properties of trees

Property 1 Every tree T with $n \geq 2$ nodes has at least 2 leaves.

Proof By contradiction: Suppose T has 0 or 1 leaf.

Travel along its edges starting from the leaf (if any) or from any node, using each edge at most once.



Since a tree has no cycles, the nodes cannot be visited twice.

If there is no (other) leaf, we can leave each node along an unused incident edge.

$\Rightarrow$ an infinite path in a finite graph!

Property 2

Every tree with n nodes has $n - 1$ edges.

Proof

By induction:

- *Inductive base* : true for $n = 1$ (1 node and 0 edges)

- *Inductive step* : if it true for the trees with n nodes, it is also true for those with $n + 1$ nodes.

Consider a tree T_1 with $n + 1$ nodes.

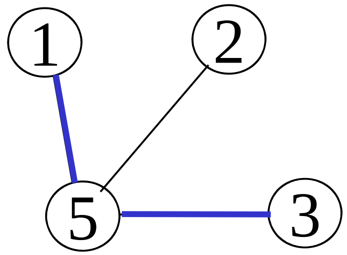
By deleting one leaf and its incident edge, we obtain a tree T_2 with n nodes.

Since by assumption Property 2) holds for T_2, T_2 has $n - 1$ edges.

$\Rightarrow T_1$, which has one more edge than T_2, has n edges.

Property 3

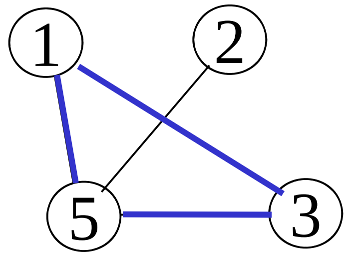
Any pair of nodes is connected via a unique path.



...otherwise there would be a cycle!

Property 4

By adding to a tree any edge that it does not contain, we create a unique cycle.



...consisting of the path of Property 3) and the new edge.

Property 5

Exchange property

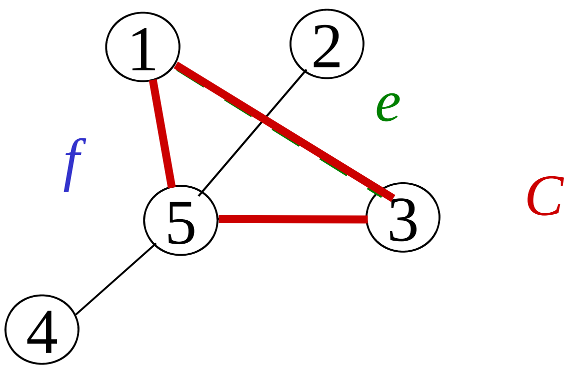
Let $G_T = (N, T)$ be a spanning tree of $G = (N, E)$

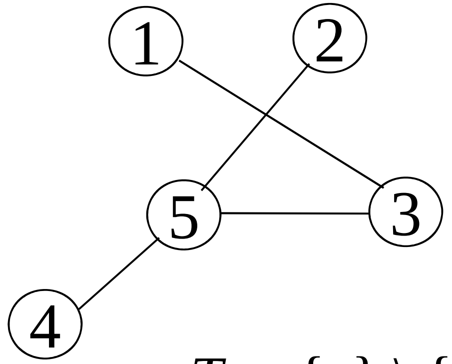
Consider an edge $e \notin T$

and the unique cycle C of $T \cup \{e\}$ (Property 4).

For each edge $f \in C \setminus \{e\}$, the subgraph $T \cup \{e\} \setminus \{f\}$ is also a spanning tree of G.

G_T





$T \cup \{e\} \setminus \{f\}$