Discrete mathematics: graphs, trees, lists

Paolo Camurati
Dip. Automatica e Informatica
Politecnico di Torino

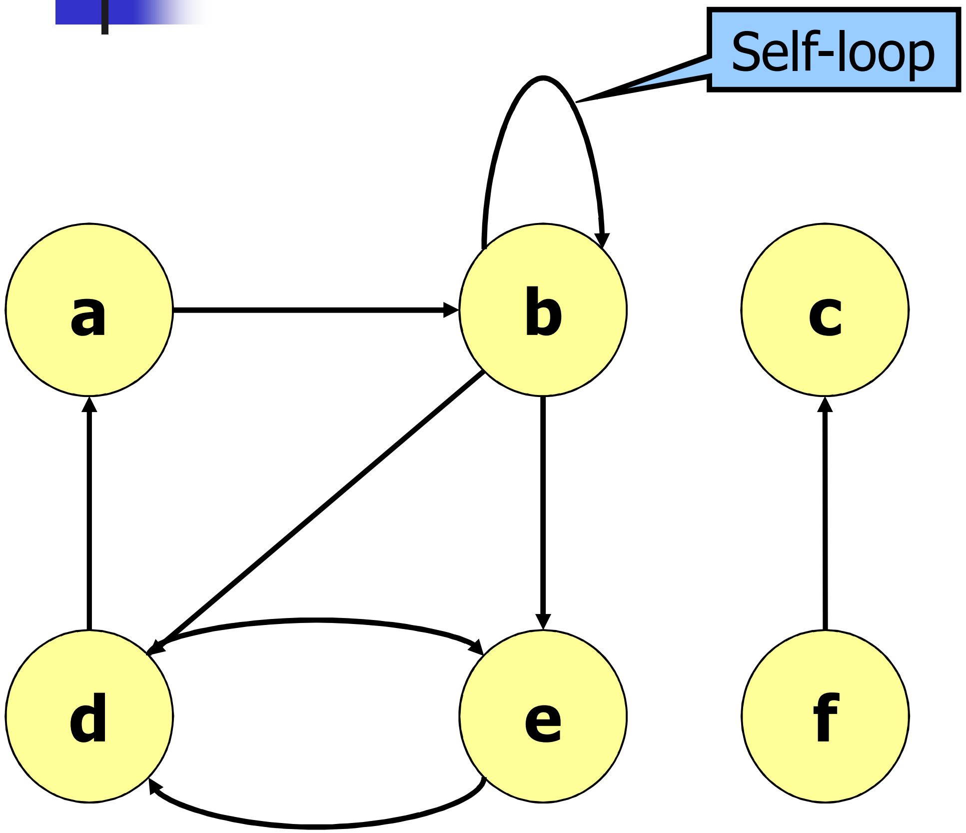
Graphs

- Definition: G = (V,E)
  - V: finite and non empty set of vertices (simple or complex data)
  - E: finite set of edges, that define a binary relation on V
- Directed/undirected graphs
  - Directed: edge = sorted pair of vertices (u, v) $\in$ E e u, v $\in$ V
  - Undirected: edge = unsorted pair of vertices (u, v) $\in$ E and u, v $\in$ V

Graphs as models

Domain	Vertex	Edge
communications	phone, computer	fiber optic, cable
circuits	gate, register, processor	wire
mechanics	joint	spring
finance	stocks, currencies	transactions
transports	airoport, station	air corridor, railway line
games	position on board	legal move
social networks	person	friendship
neural networks	neuron	synapsis
chemical compounds	molecules	link

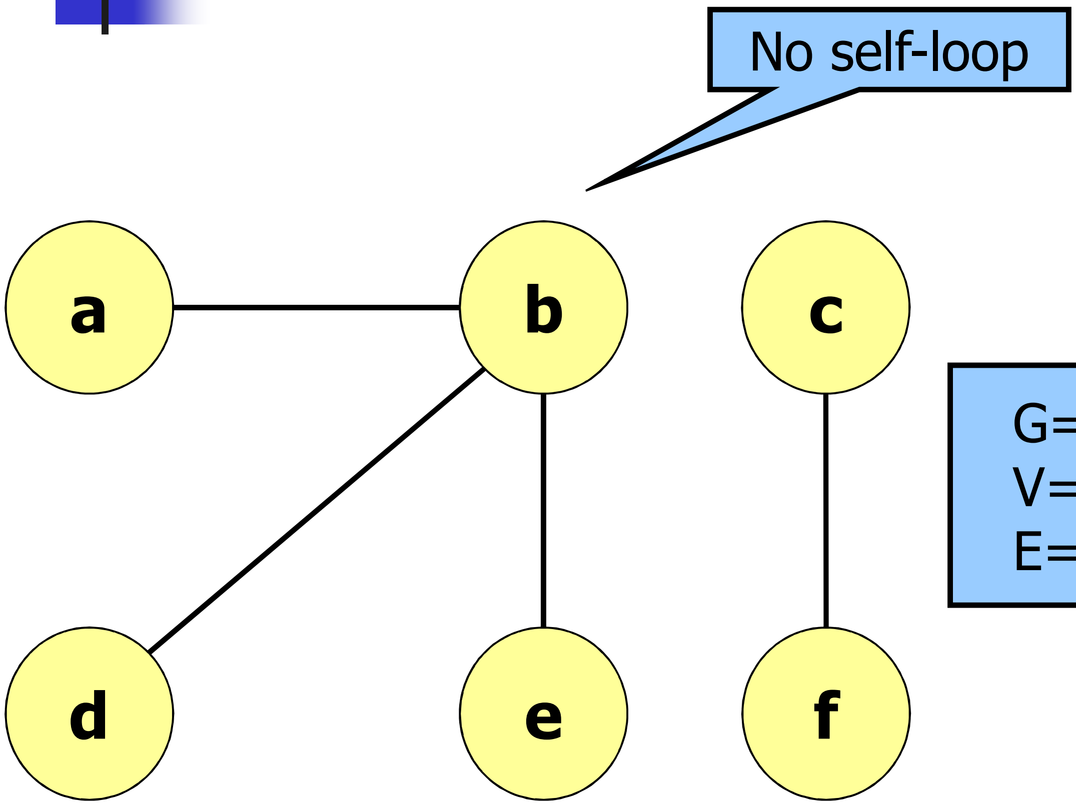
Example: directed graph



G={V, E}
V={a, b, c, d, e, f}
E={(a,b),(b,b),(b,d),(b,e),
(d,a),(d,e),(e,d),(f,c)}

In some contexts self-loops may be forbidden. If the context allows loops, but the graph is self-loop-free, it is called **simple**.

Example: undirected graph

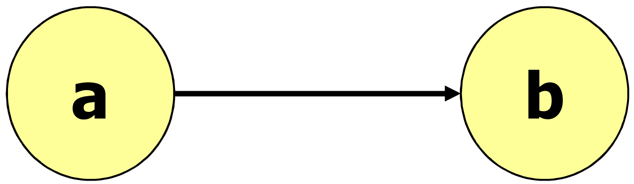


G={V, E}
V={a, b, c, d, e, f}
E={(a,b),(b,d),(b,e),(c,f)}

In some contexts self-loops may be forbidden. If the context allows loops, but the graph is self-loop-free, it is called **simple**.

Incidence and adjacency

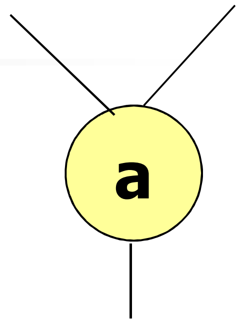
- Edge (a, b)
  - Incident from vertex a
  - Incident in vertex b
  - Incident on vertices a and b



- Vertices a and b adjacent
- $a \rightarrow b \Leftrightarrow (a, b) \in E$

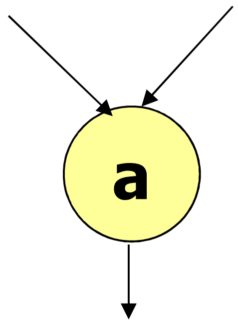
Degree of a vertex

degree(a) = 3



- Undirected graph
  - degree(a) = number of incident edges

in_degree(a) = 2
out_degree(a) = 1
degree(a) = 3



- Directed graph
  - in_degree(a) = number of incoming edges
  - out_degree(a) = number of outgoing edges
  - degree(a) = in_degree(a) + out_degree(a)

Paths and reachability

Path p

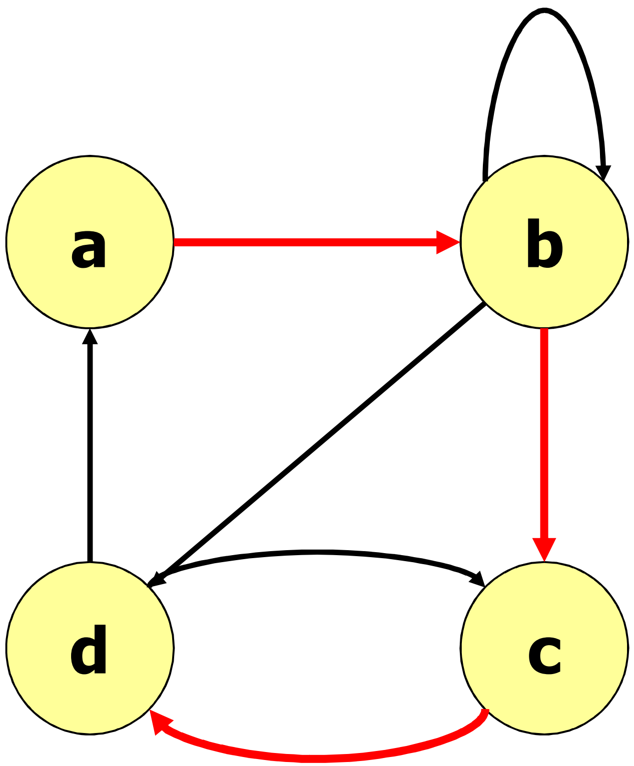
$u \to_p u'$ in $G=(V,E)$

$\exists (v_0, v_1, v_2, \ldots, v_k) \mid$

$u=v_0,\ u'=v_k,\ \forall i = 1,2,\ldots,k\ (v_{i-1}, v_i) \in E$

- k = length of the path
- u' is reachable from $u \Leftrightarrow \exists\, p\colon\ u \to_p u'$
- simple path p: distinct $(v_0, v_1, v_2, \ldots, v_k) \in p$

Example



$G = (V, E)$

p: $a \rightarrow_p d$: (a, b), (b, c), (c, d)

$k = 3$

d is reachable from a

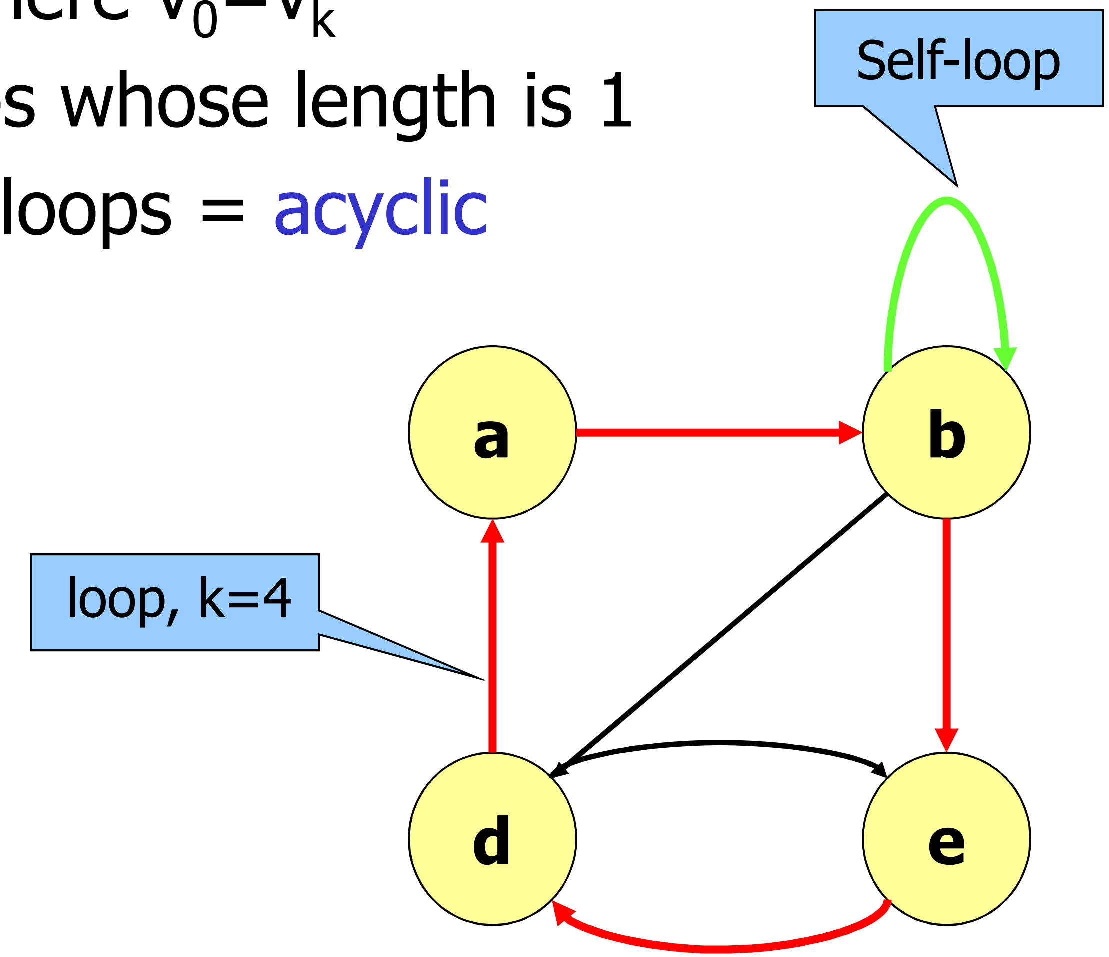
p is a simple path

Loops

Loop = path where $v_0=v_k$

Self-loop = loops whose length is 1

Graphs without loops = acyclic



Connection in undirected graphs

Undirected graph

- Connected
  - $\forall v_i, v_j \in$ V there $\exists p$ $v_i \rightarrow_p v_j$
- Connected component
  - Maximal connected subgraph, that is, there is no subset including it for which the property holds
- Connected undirected graph
  - Only one connected component

Connection in directed graphs

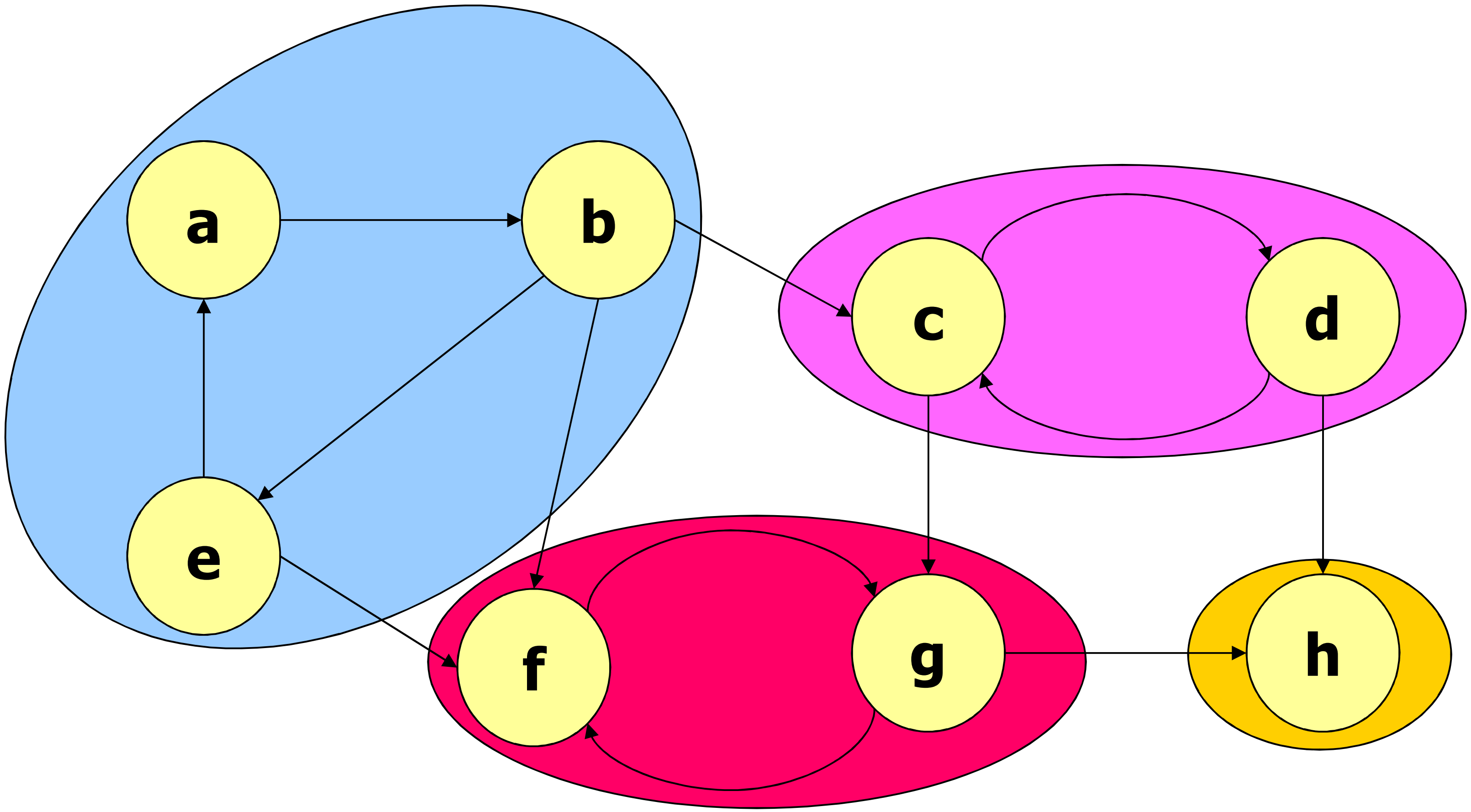
Directed graph

- Strongly connected
  - $\forall v_i, v_j \in V \quad \exists p, p' \;\; v_i \rightarrow_p v_j$ and $v_j \rightarrow_{p'} v_i$
- Strongly connected component
  - Maximal strongly connected subgraph

Strongly connected directed graph

- Only one strongly connected component

Example

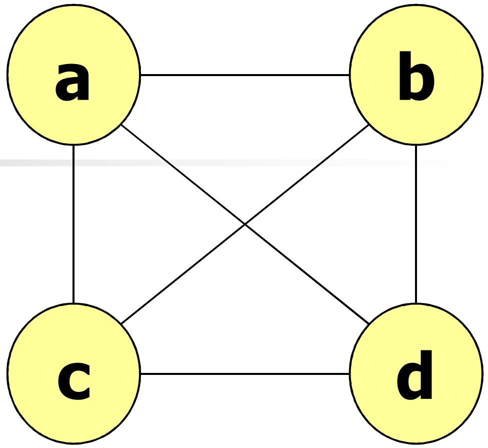


Dense/sparse graphs

- Given graph $G = (V, E)$
  - $|V|$ = cardinality of set V
  - $|E|$ = cardinality of set E

- Dense graph
  - $|E| \cong |V|^2$
- Sparse graph
  - $|E| << |V|^2$

Complete graph



Definition:

$\forall v_i, v_j \in V \ \exists (v_i, v_j) \in E$

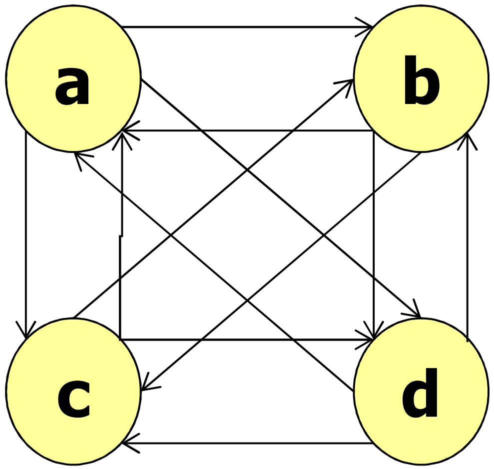
How many edges?

- Complete undirected graph
  - $|E|$ = number of **combinations** of $|V|$ elements taken 2 by 2
  - $|E| = \frac{|V|!}{(|V|-2)! \cdot 2!} = \frac{|V| \cdot (|V|-1) \cdot (|V|-2)!}{(|V|-2)! \cdot 2!} = \frac{|V| \cdot (|V|-1)}{2}$

Order does not matter

Order matters

- Complete directed graph
  - |E| = number of **dispositions** of |V| elements taken 2 by 2
  - $|E| = \frac{|V|!}{(|V|-2)!} = \frac{|V| \cdot (|V|-1) \cdot (|V|-2)!}{(|V|-2)!} = |V| \cdot (|V|-1)$

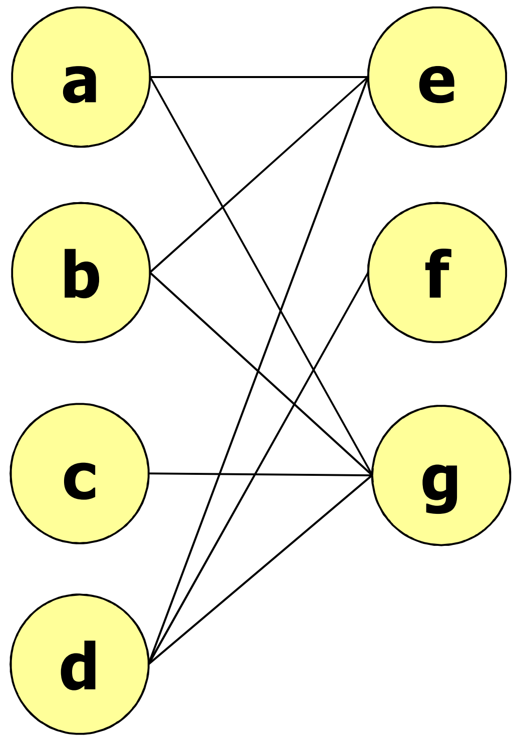


Bipartite graph

- Definition
  - Undirected graph where the V set may be partitioned in 2 subsets V_1 and V_2, such that

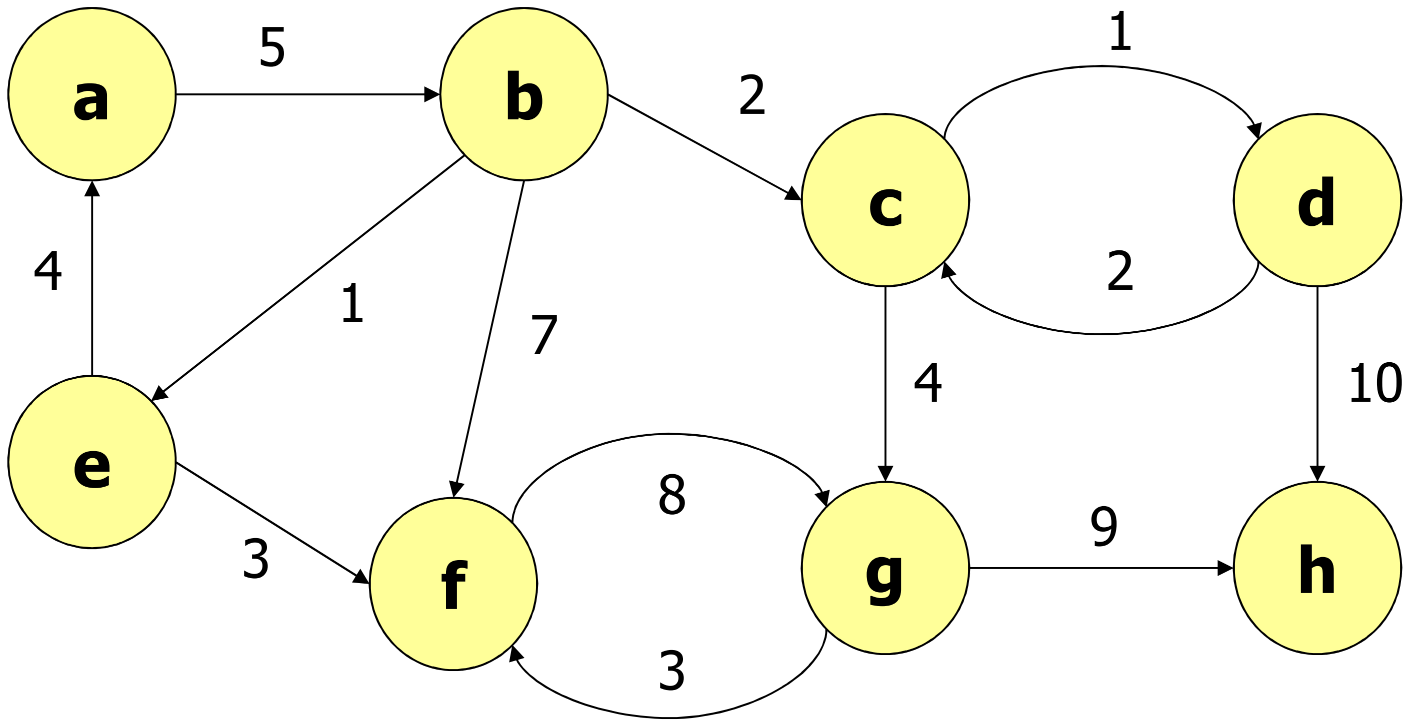
$\forall (v_i, v_j) \in E$

$v_i \in V_1 \;\&\&\; v_j \in V_2 \;||\; v_j \in V_1 \;\&\&\; v_i \in V_2$

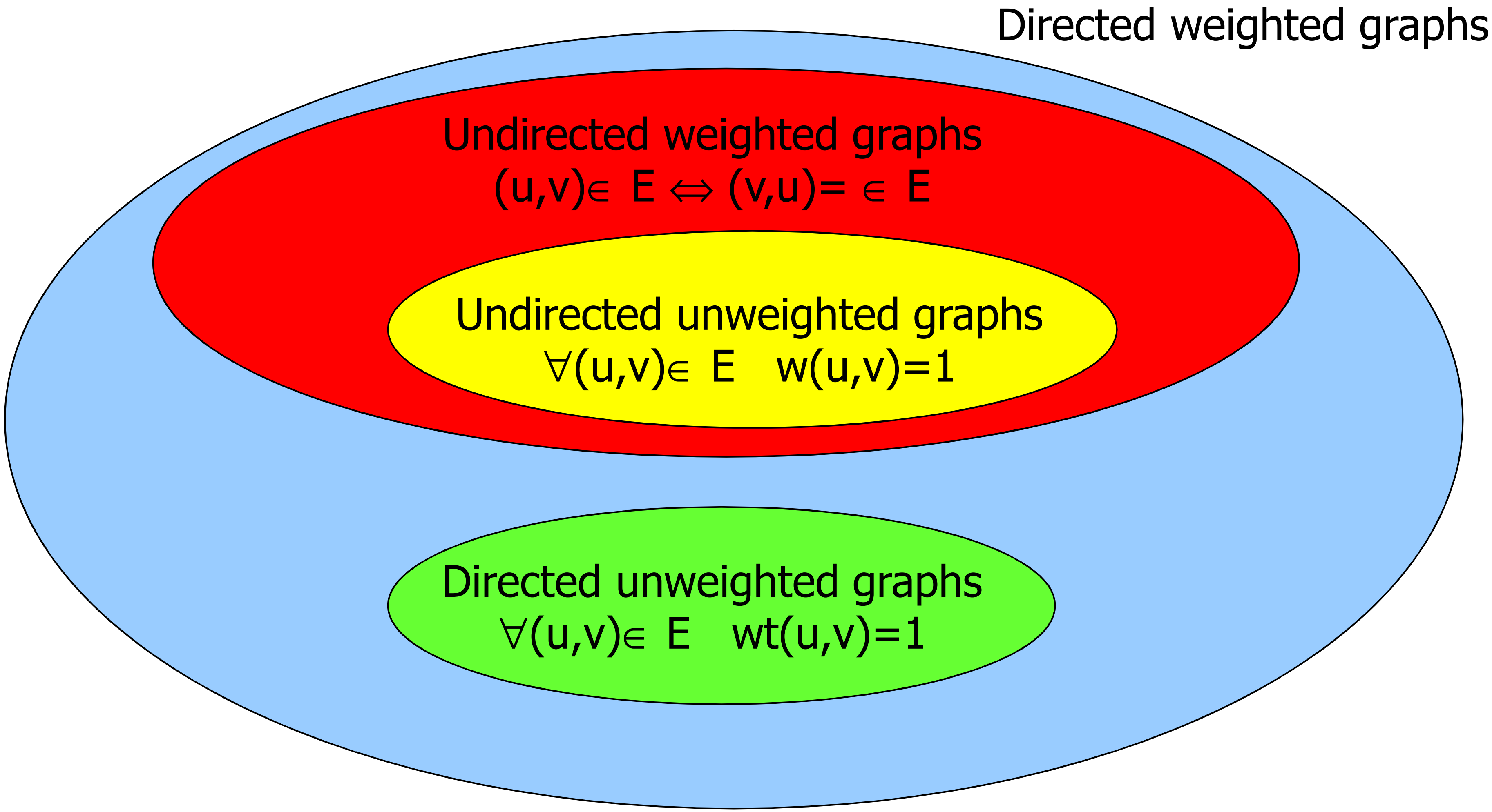


Weighted graph

$\exists\ w : E \rightarrow R \mid w(u,v)$ = weight of edge (u, v)

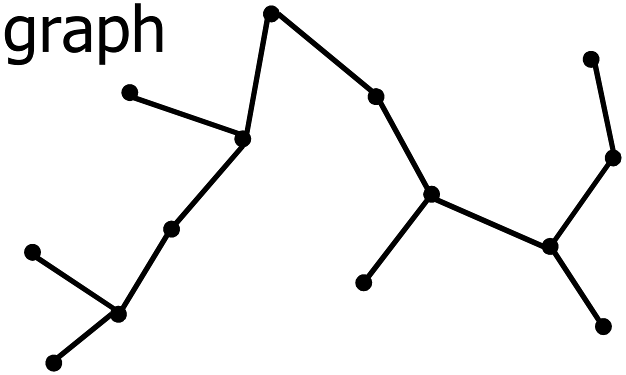


Types of Graphs

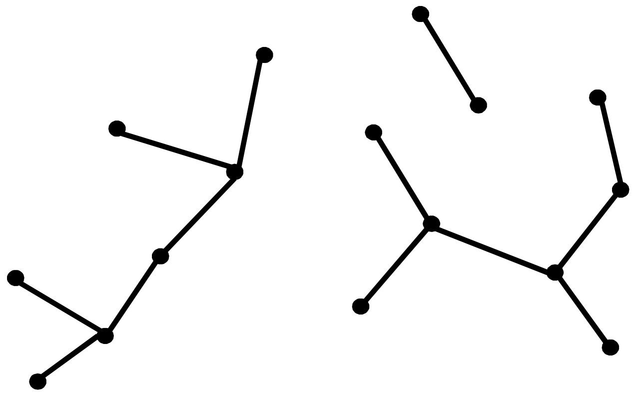


Non rooted trees

- Non rooted trees
  - Undirected, connected, acyclic graph



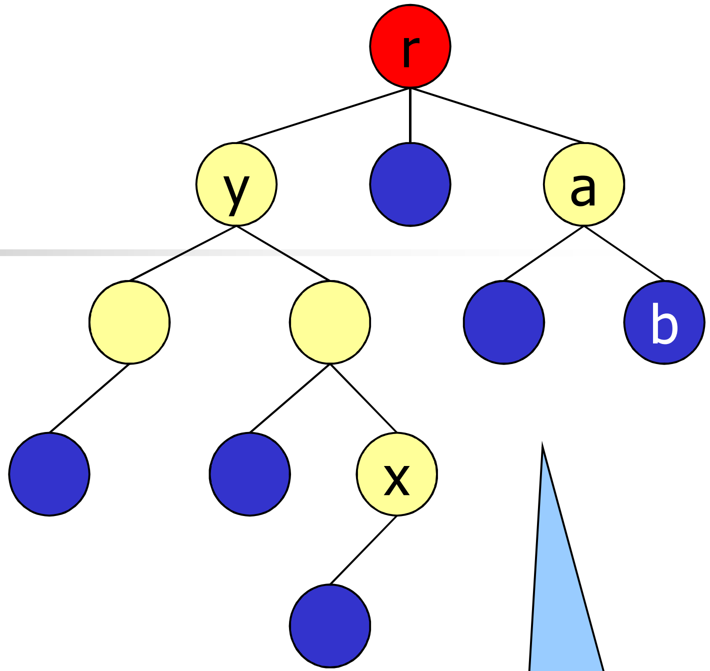
- Forest
  - Undirected acyclic graph



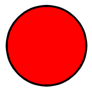
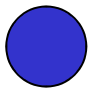
Properties

- G = (V, E) undirected graph $|E|$ edges, $|V|$ nodes
  - G = non rooted tree
  - Every pair of nodes is connected by a single simple path
  - G connected, removing an edge disconnects the graph
  - G connected and $|E| = |V| - 1$
  - G acyclic and $|E| = |V| - 1$
  - G acyclic, adding an edge introduces a loop

Rooted trees



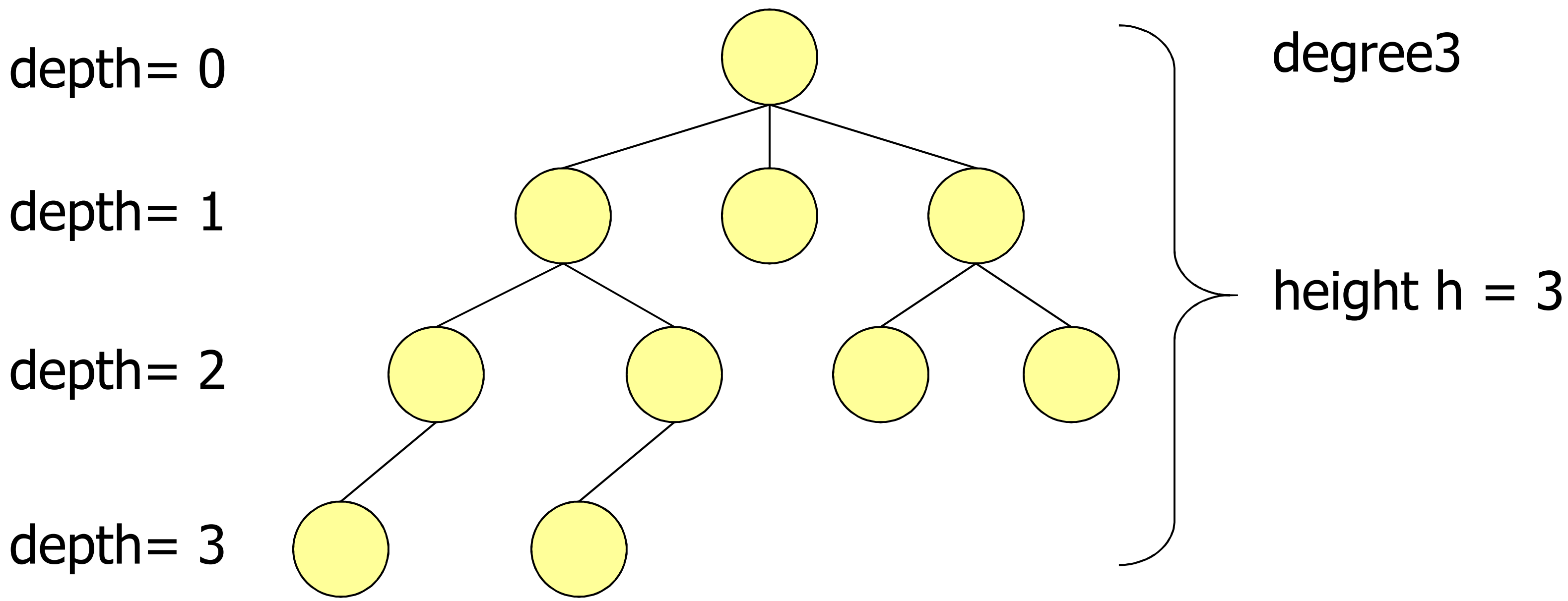
∃ a node r called root

- Parent/child relationship
  - y ancestor of x
    - if y belongs to the path from r to x
    - x is a descendant of y
  - Proper ancestor if $x \neq y$
  - Parent/child: adjacent nodes
- Root: no parent 
- Leaves: no children 

y ancestor of di x
x descendant of y
a parent of b
b child of a

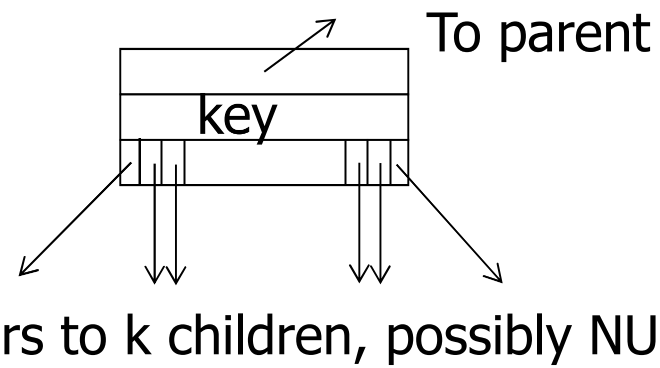
Properties of a tree T

- degree(T) = maximum number of children
- depth(x) = length of the path from r to x
- height(T) = maximum depth



Representation of trees

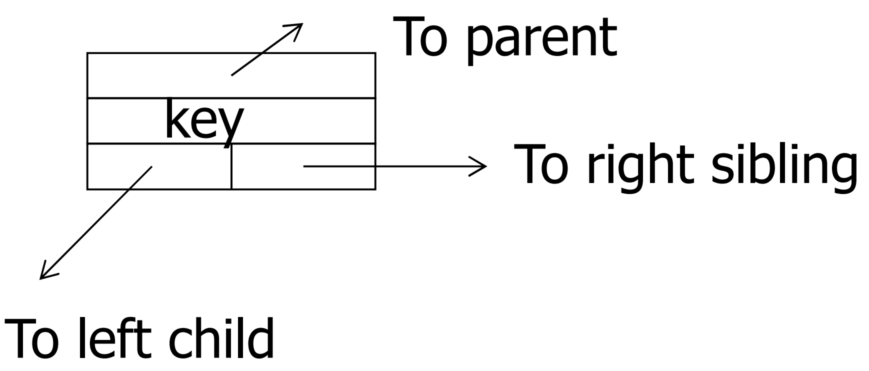
- Representation of a node of a tree of degree(T) = k
- Pointer to parent, key, k pointers to k children



- Unefficient if only few nodes have indeed degree k
  - Space is allocated for all k pointers, but many are NULL)

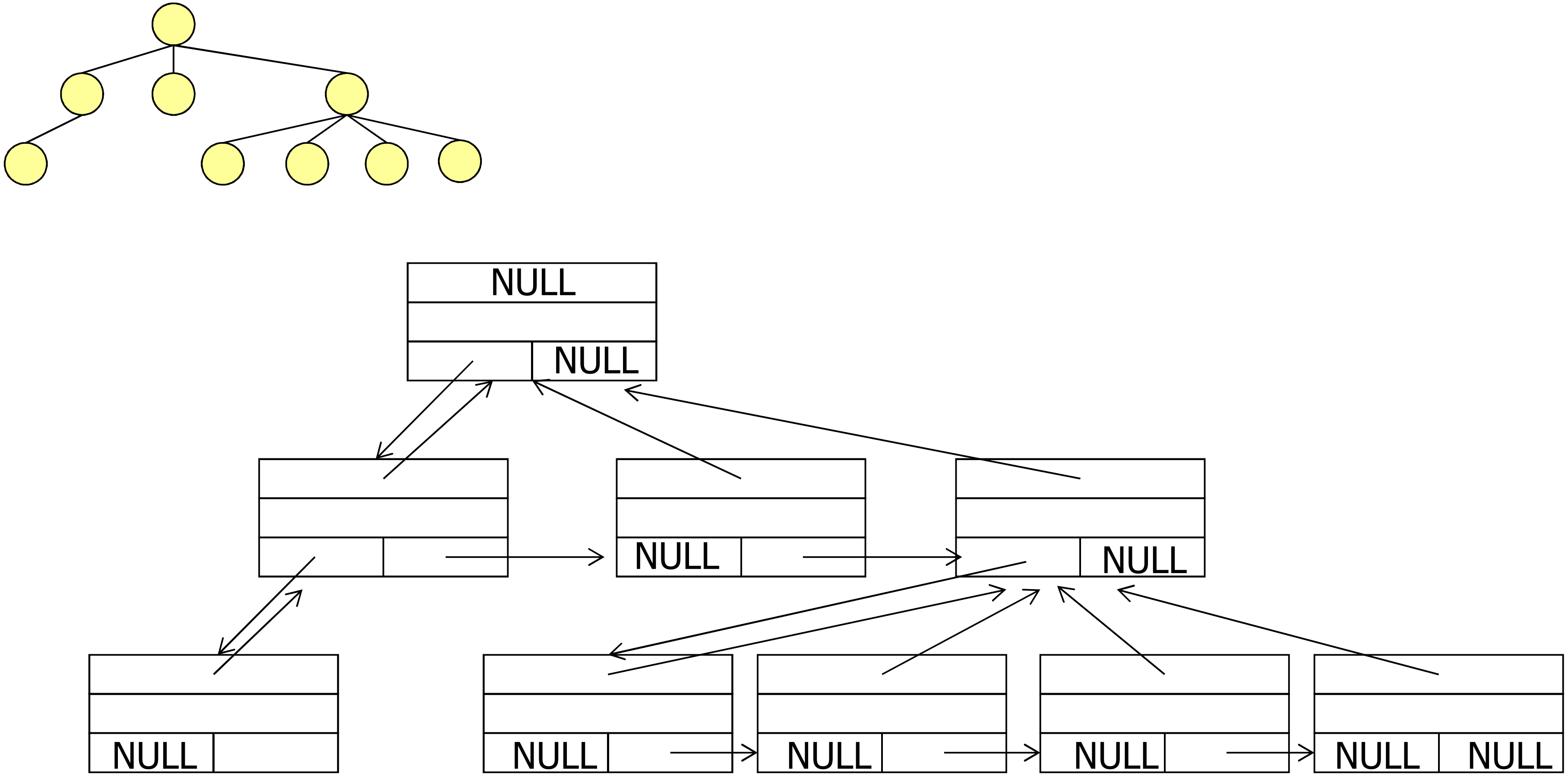
Left-child right-sibling representation

- Representation of a node of a tree of degree(T) = k
- Pointer to parent, key, 1 pointer to left child, 1 pointer to right sibling



- Efficiency
  - Always 2 pointers, no matter the degree of the tree

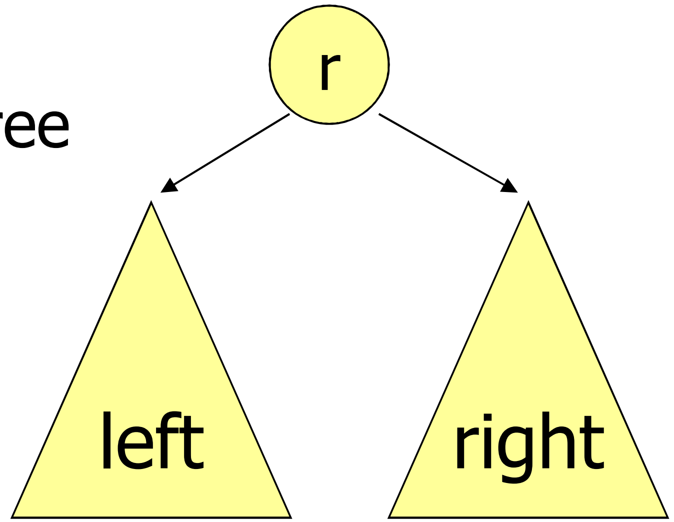
Left-child right-sibling representation



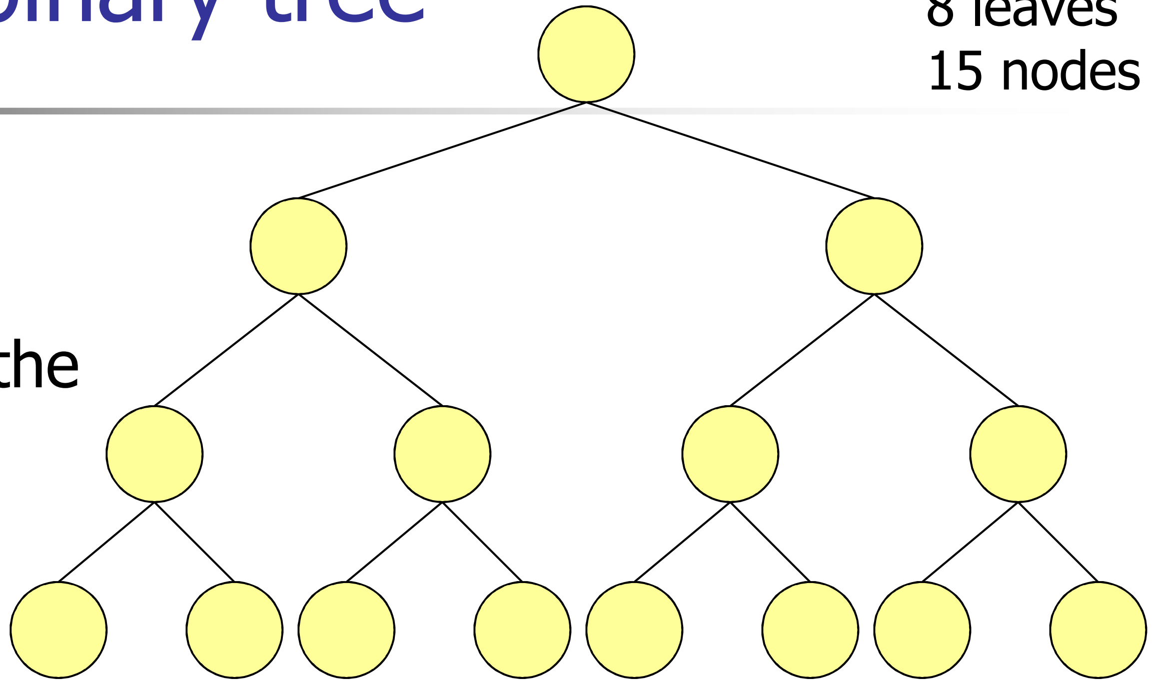
Binary trees

Definition

- Tree of degree 2
- Recursively T is
  - Empty set of nodes
  - Root, left subtree, right subtree



Complete binary tree

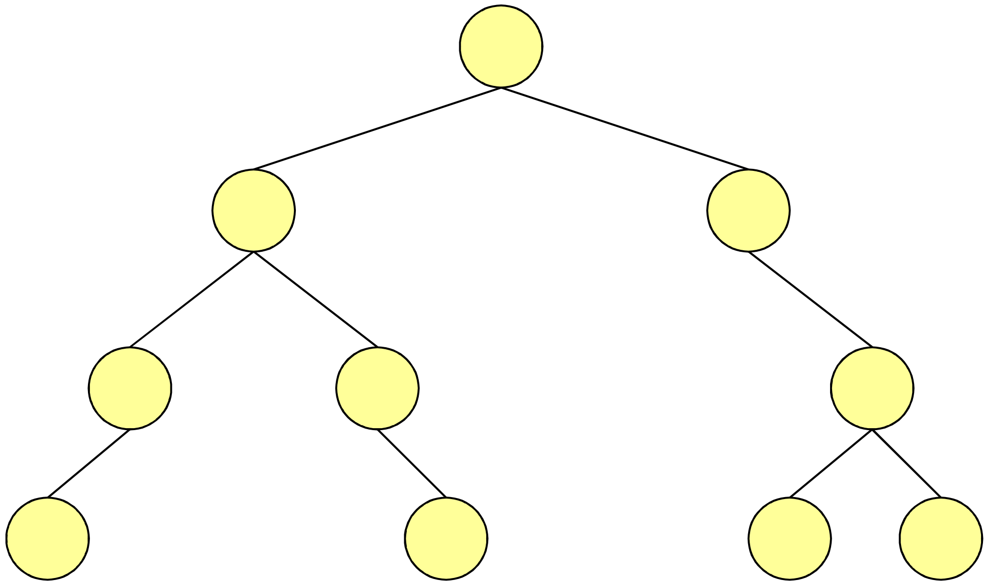


- Two conditions
  - All leaves have the same depth
  - Every node is either a leaf or it has 2 children
- Complete binary tree of height h
  - Number of leaves 2^h
  - Number of nodes = $\Sigma_{0 \leq i \leq h} 2^i = 2^{h+1} - 1$

Finite geometric progression with ratio =2

Balanced binary tree

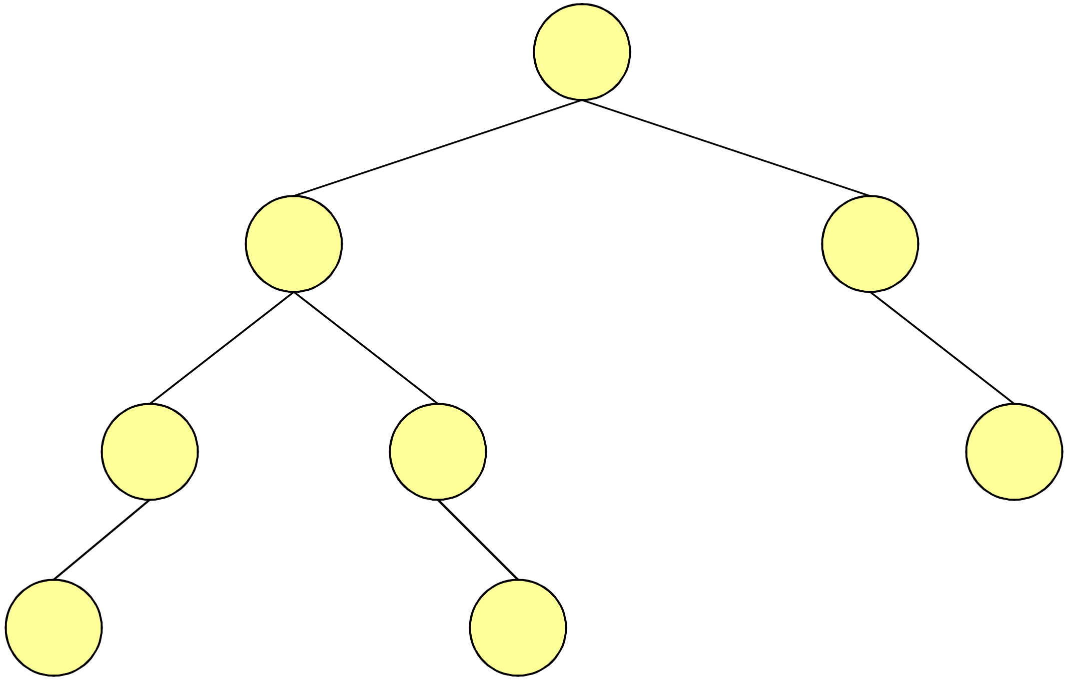
- All paths root-leaves have same length



- If T is complete, then T is balanced
- The opposite is not necessarily true

Almost balanced binary tree

- The length of all paths root-leaves differs at most by 1



Linear Sequences

- Finite set of consecutive elements
  - A unique index is associated to each element
  - $a_0, a_1, \dots, a_i, \dots, a_{n-1}$

- A predecessor/successor relation is defined on couples of elements
  - $a_{i+1} = succ(a_i)$
  - $a_i = pred(a_{i+1})$

Storage and access

- Array
  - Storage: contiguous data in memory
- Direct access
  - Given index i, we access element a_i without any need for scanning the whole sequence
  - The cost of an access does not depend on the position of the element in the linear sequence, thus it is O(1)

Storage and access

- List
  - Storage: non contiguous data in memory
- Sequential access
  - Given index i, we access element a_i scanning the linear sequence starting from one of its boundaries, usually the left one
  - The access cost depends on the position of the element in the linear sequence, thus it is O(n) in the worst case

Operations on lists

- Search
  - For an element whose search key field equals a given key
- Insert an element
  - At the head of an unsorted list
  - At the tail of an unsorted list
  - At a position such as to guarantee that the invariance propoerty of a sorted list is satisfied

Operations on lists

- Extract an element
  - From the head of an unsorted list
  - That has a field whose contents equals a deletion key (such an operation usually requires a a search for the element to be deleted).

Collections of data

- Generalized queues
  - Collections of objects (data) with insert, search, and delete as main operations
- Insert
  - Insert a new object into the collection
- Search
  - Search for an object in the collection
- Delete
  - Delete an object from the collection

Collections of data

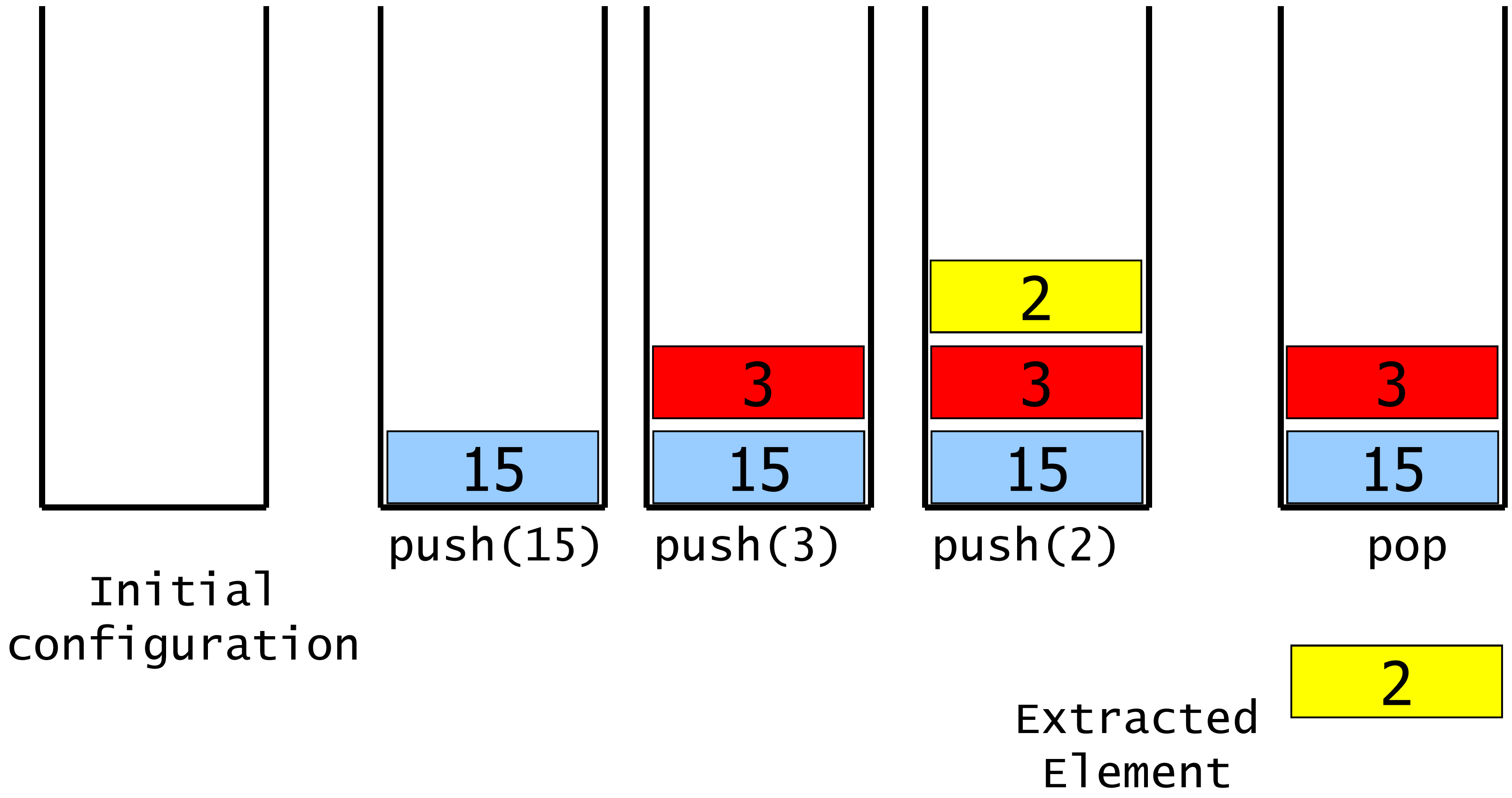
- Other operations
  - Initialize the generalized queue
  - Count objects (or check if collection empty)
  - Destroy generalized queue
  - Copy generalized queue

Collections of data: Stack

Criteria to extract elements

- Extract the most recently inserted element
  - LIFO policy: Last-In First-Out
  - Stack
  - Insertion (push) and extraction (pop) from head

Collections of data: Stack

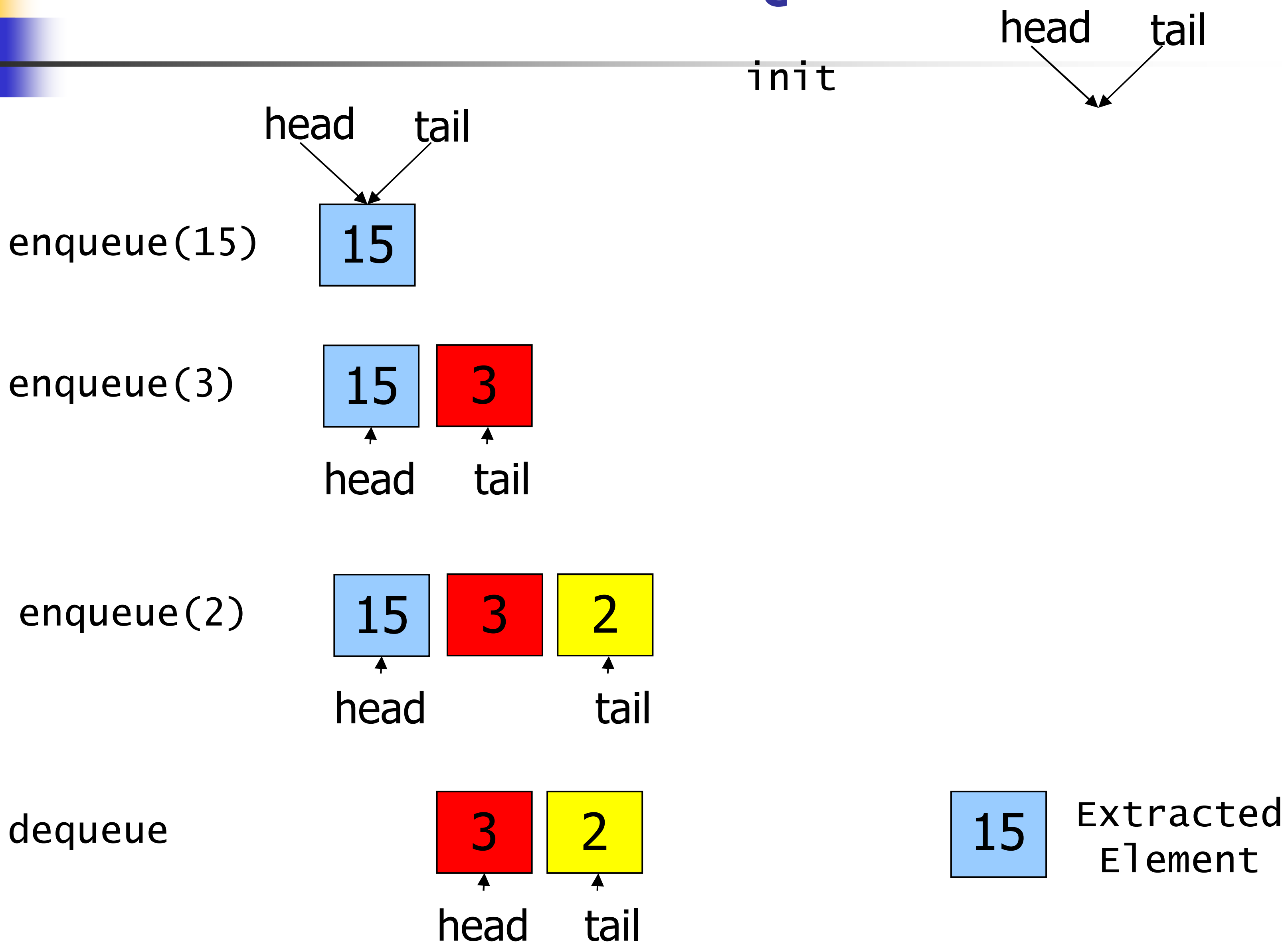


Collections of data: Queue

Criteria to extract elements

- Extract the least-recently inserted element
  - FIFO policy: First-In First-Out
  - Queue
  - Insert (enqueue) at the tail and extract (dequeue) from the head

Collections of data: Queue

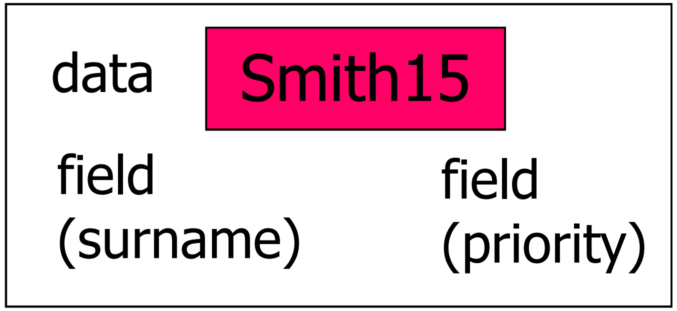


Collections of data: Priority Queue

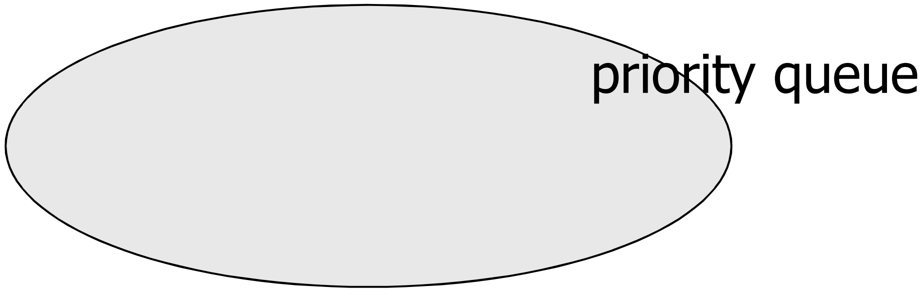
Criteria to extract elements

- Insertion guarantees that, when extracting, the highest (lowest) priority element is extracted
  - Priority queue

Collections of data: Priority Queue



initially



Collections of data: Priority Queue

```
insert(Black, 5)

insert(Black, 15)

insert(Black, 2)

extract()
```

Smith 15