Trees

Definitions

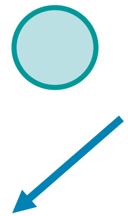
Stefano Quer
Dipartimento di Automatica e Informatica
Politecnico di Torino

Units 12 and 13

Trees

- In graph theory, a tree is an undirected graph in which any two vertices are connected by exactly one path
- More informally
  - A **tree** is composed by a set of vertices (or nodes) and a set of edges where any two vertices are connected by exactly one path

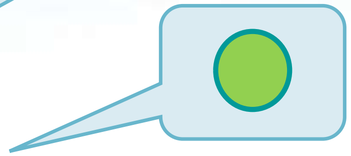
Vertices (nodes)
Edges

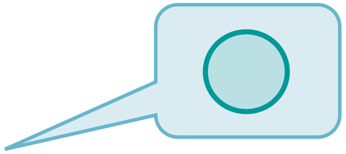


  - A **rooted tree** is a tree where there is a node called root
  - A **forest** is a disjoint union of trees

Rooted trees

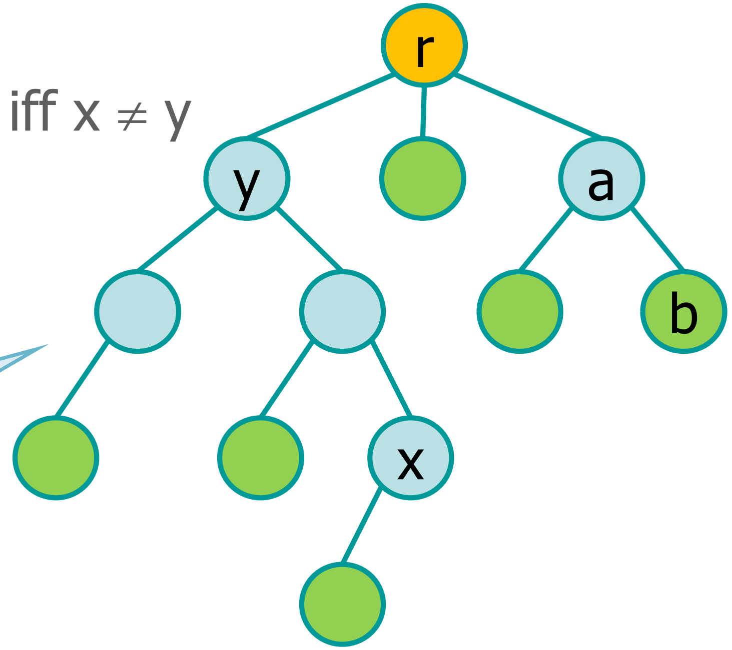
- The node(s) with
  - No parent is the root
  - No children are leaves
  - From 0 to n children are internal nodes





- Parent/child relationship
  - Node y is an ancestor of x if y belongs to the path from r to x
    - Node y is a proper ancestor of x iff x $\neq$ y
  - Node x is a descendant of y
  - Parent and a child are adjacent nodes

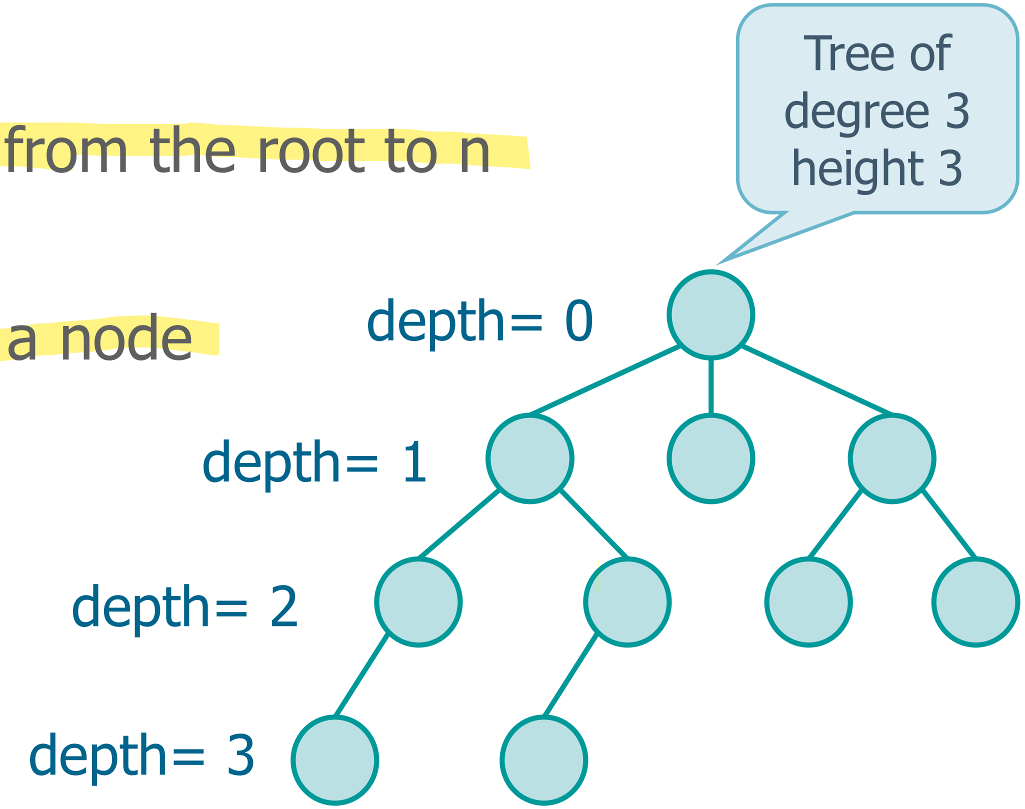
note: maximum one parent but can hsve more children



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

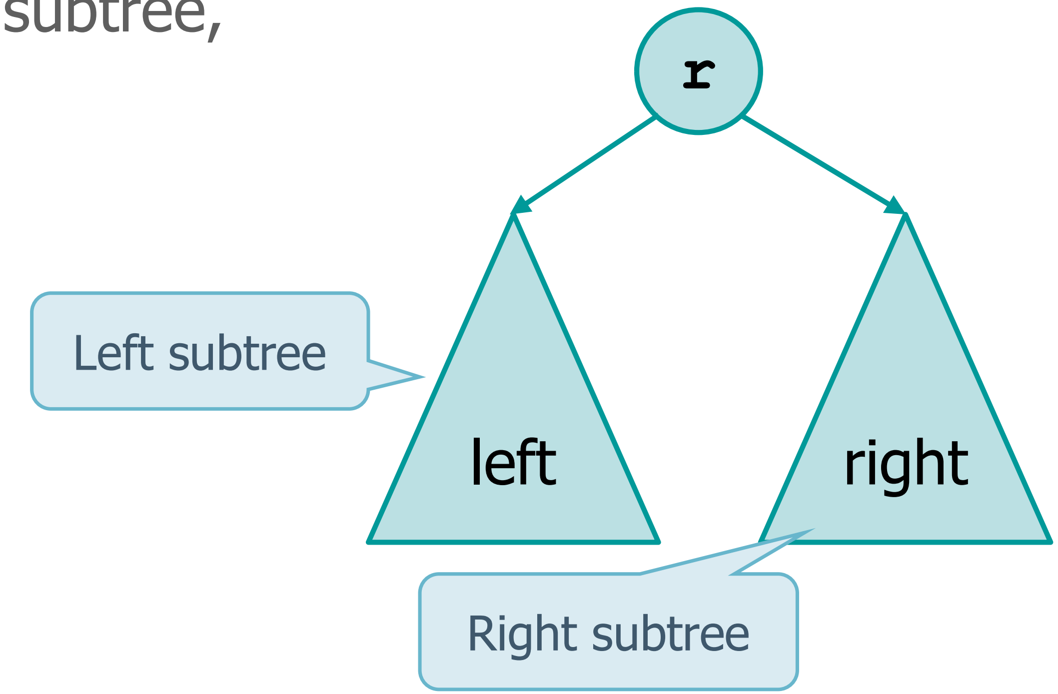
Properties of a rooted tree

- Given a rooted tree T and a node n the following are common definitions
  - Degree (T)
    - Maximum number of children
  - Depth (n)
    - Length of the path from the root to n
  - Height (T)
    - Maximum depth of a node



Binary trees

- A binary tree is
  - Tree of degree 2
    - Each node may have 0, 1 or at most 2 children
  - We can recursively express a tree T as
    - The empty set of nodes or
    - The root, the left subtree, the right subtree

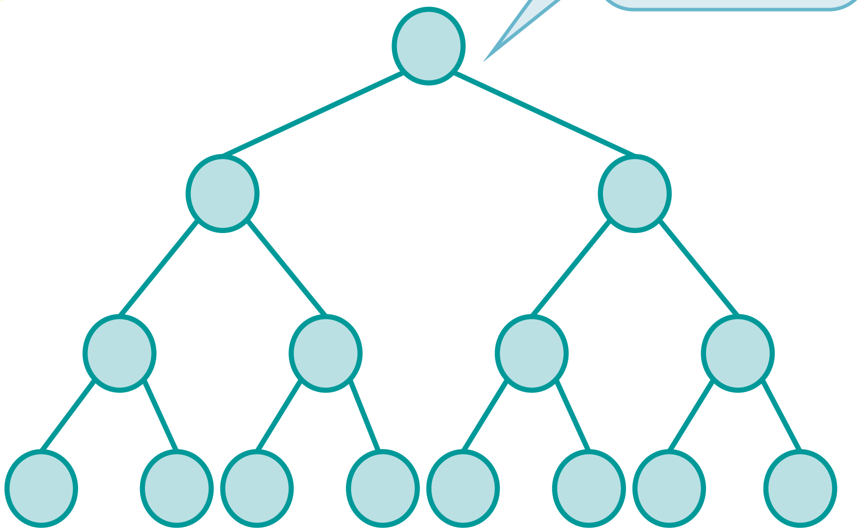


Complete binary trees

- A complete binary tree must satisfy two conditions
  - All leaves have the same depth
  - Every node is either a leaf or it has 2 children
- In a complete binary tree of height h
  - The number of leaves is 2^h
  - The number of nodes is

$$N = \sum_{i=0}^{h} 2^i =$$
$$= 2^0 + 2^1 + 2^2 + \cdots + 2^h =$$
$$= 2^{h+1} - 1$$

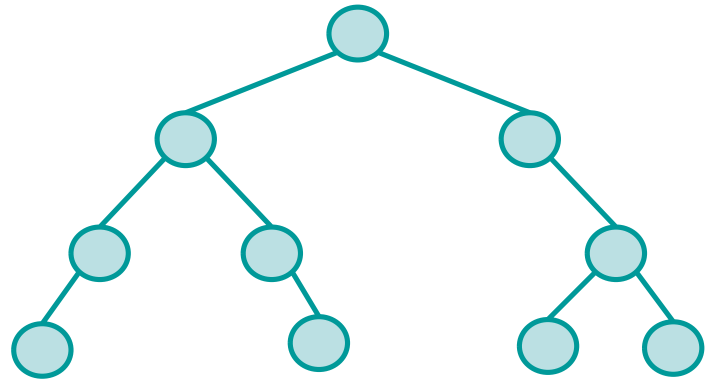
Finite geometric progression with ratio = 2



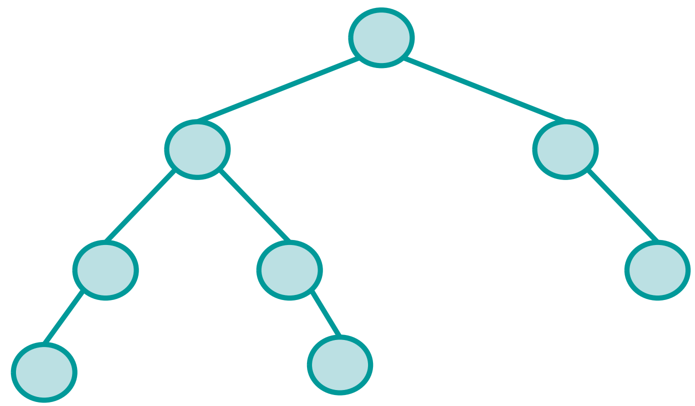
$h = 3$
8 leaves
15 nodes

Balanced binary trees

- A binary tree is balanced if all paths root-leaves have the same length
  - If T is complete, then T is also balanced
  - The opposite is not necessarily true



- A binary tree is almost balanced if the length of all paths from root to leaves differs at most by 1



Node structure

- Each node must store
  - The data fieds (including the key)
  - The pointers to children
  - Possibly, a pointer to the parent
    - The pointer to the parent is necessary **only** for specific operations
- Pointers may be critical for generic trees, as the number of children varies
  - There are at least two representations for nodes of a tree of degree n

Node structure

- Each node stores 1 pointer to every child, i.e., n pointers overall
  - For binary trees, we may store only two pointers (i.e., left or l and right or r)
  - For n-ary trees
    - If the degree n is reached by the majority of the nodes we may use a static array of pointers (children or child)
    - If only few nodes have the maximum degree, we may use a dynamic array of pointers to children

Node structure

Binary tree

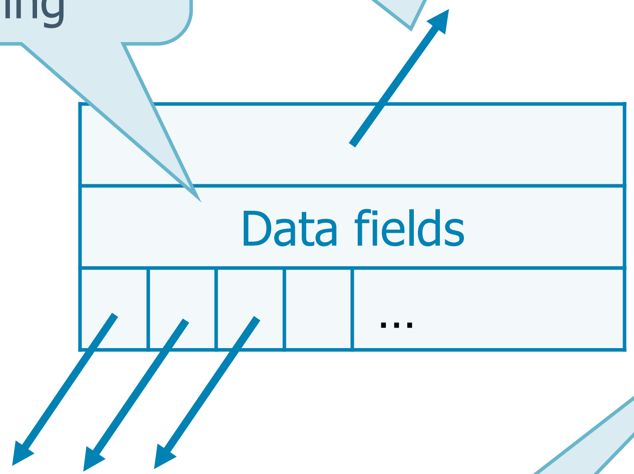
```
typedef struct node_s node_t;
struct node_s {
  int key;
  ...
  node_t *l, *r;
};
```

binary

most used

The key can be an integer or a string

The pointer to the father is optional




```
typedef struct node_s node_t;
struct node_s {
  int key;
  ...
  node_t *children[N];
};
```

if fixed degree

Pointers to children

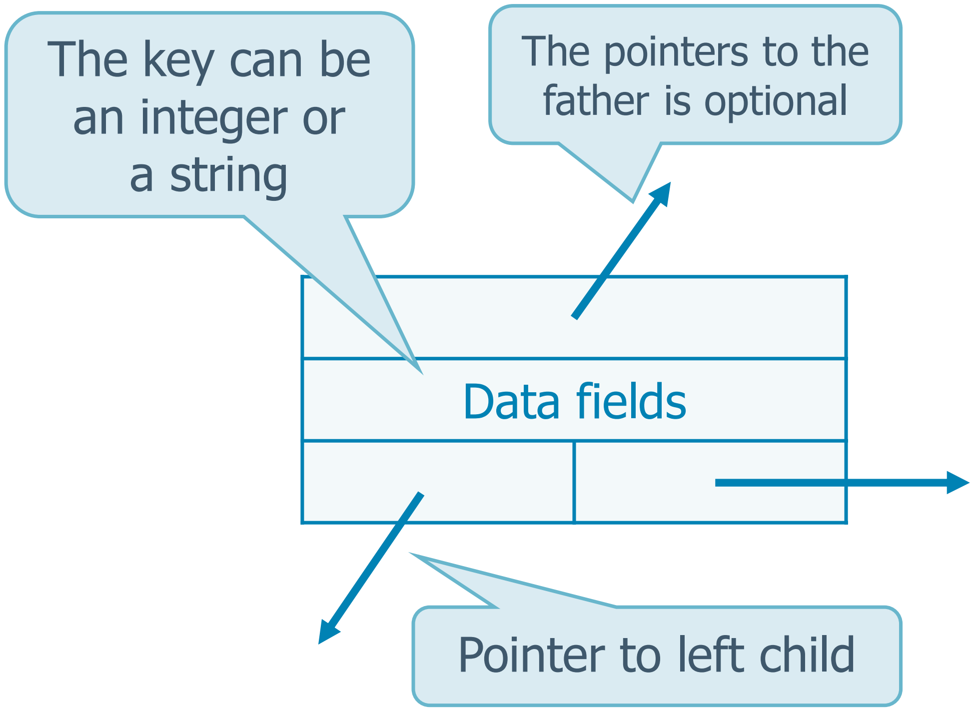
N-ary tree

```
typedef struct node_s node_t;
struct node_s {
  int key;
  ...
  int degree;
  node_t **children;
};
```

general

Node structure

- Each node stores 1 pointer to the left child and 1 pointer to the siblings
  - We always have two pointers per node and we save memory



another representation

```
typedef struct node_s node_t;
struct node_s {
  int key;
  ...
  node_t *l, *s;
};
```

Node structure