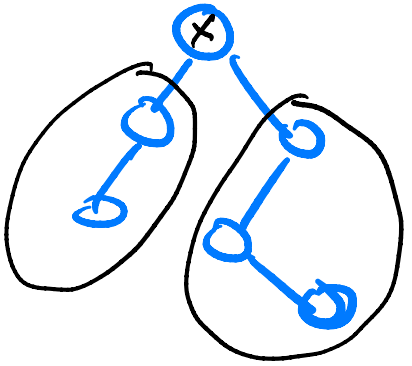
ALBERI BINARI (natura induttiva)



Sequenze: array, liste, vector

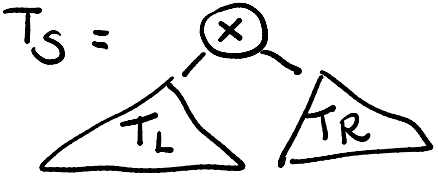
relazione implicita: precede/succede

S = insieme di n elementi ⟶ T = albero di n elementi (di S)

partizione ricorsiva

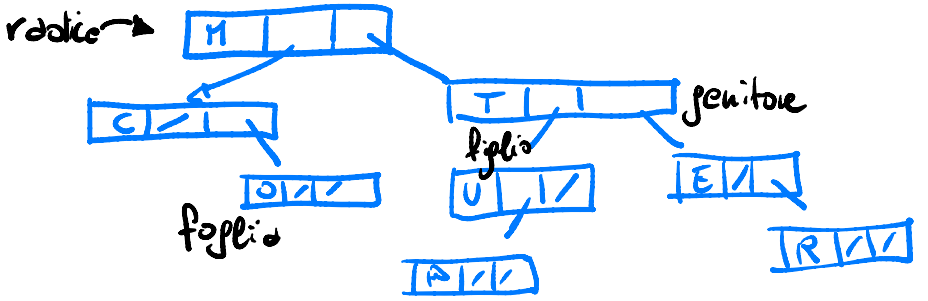
albero binario → 3-partizione

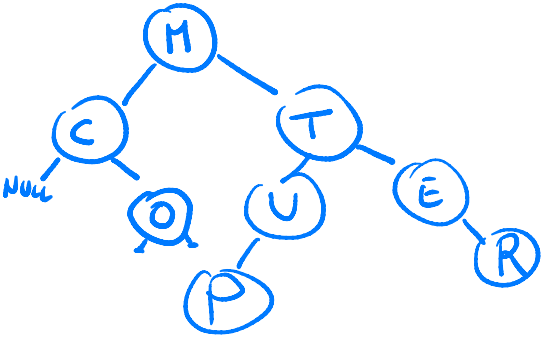
$S = L \cup \{x\} \cup R$ se $S \neq \emptyset$



(è possibile che L o R, o entrambe, siano vuoti)

Caso base = albero vuoto NULL

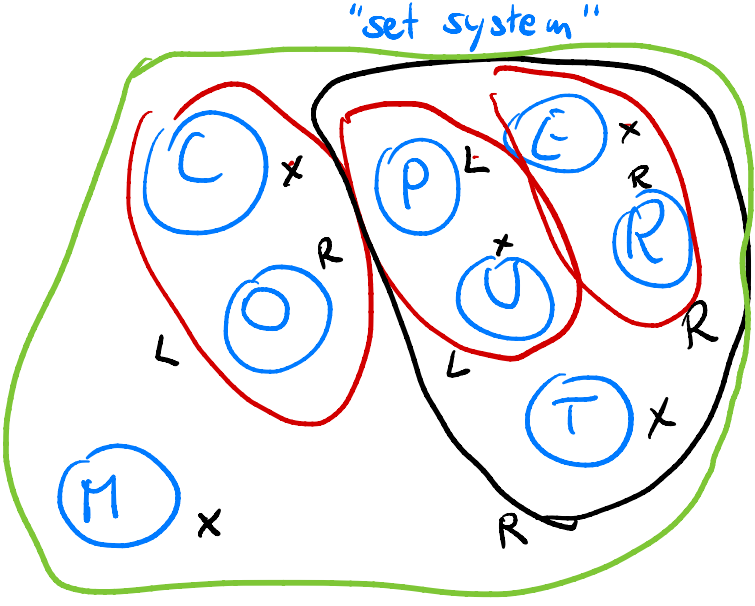




$$S = \{C, O, M, P, U, T, E, R\}$$

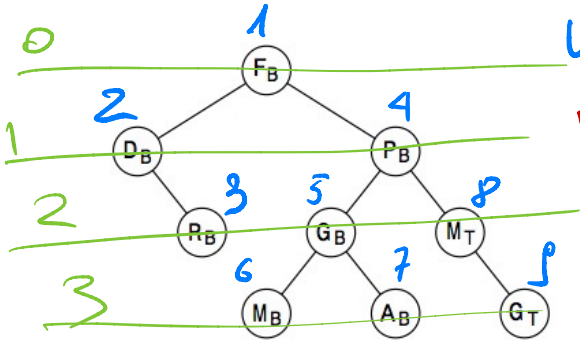
L $\quad$ X $\quad$ R

$\phi = L \times \overset{\cup}{R}$



$$\forall A, B : \begin{cases} A \cap B = \emptyset \\ A \subseteq B \\ B \subseteq A \end{cases}$$

| no sovrapposizione parziale



anticipata:

F_B D_B R_B P_B G_B M_B A_B M_T G_T

simmetrica:

D_B R_B F_B M_B G_B A_B P_B M_T G_T

posticipata:

R_B D_B M_B A_B G_B G_T M_T P_B F_B

ampiezza:

F_B D_B P_B R_B G_B M_T M_B A_B G_T

chiamate ricorsive → la vediamo

valutazioni ← con i push

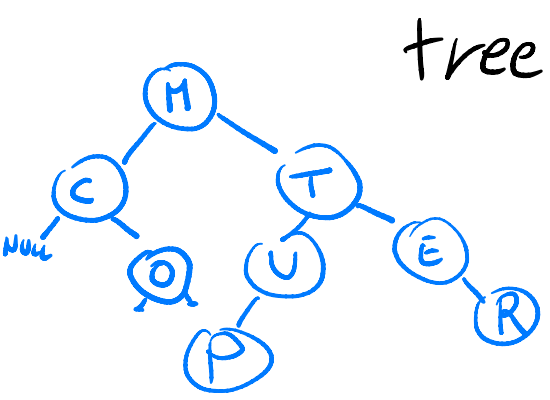
visita(u):

```
if (u ≠ NULL) {
    visita(u->left)
    cout << u->key
    visita(u->right)
}
```

vedi codice

dimensione di un albero = # nodi
taglia

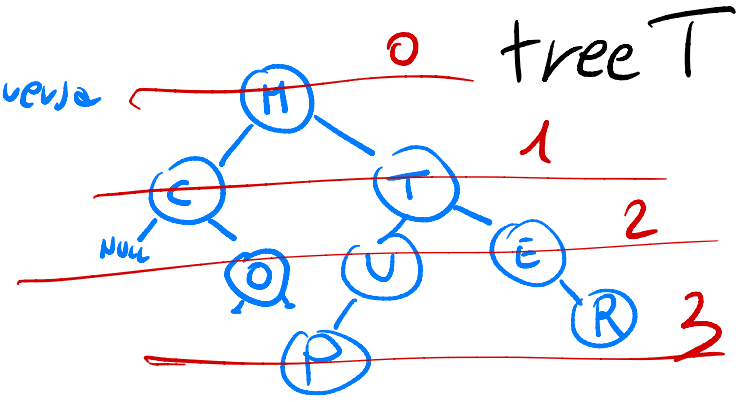
$$|T| = 8$$




```
dim(u):
  if (u = NULL) return 0
  return 1 + dim(u->left) + dim(u->right)
           radice
```

profondità: # archi minimo da attraversare per salire fino alla radice

livello i = nodi a profondità i



altezza = max profondità

```
alt(u):
if (u==NULL) return -1;
return 1 + max(alt(u->sx), alt(u->dx))
```

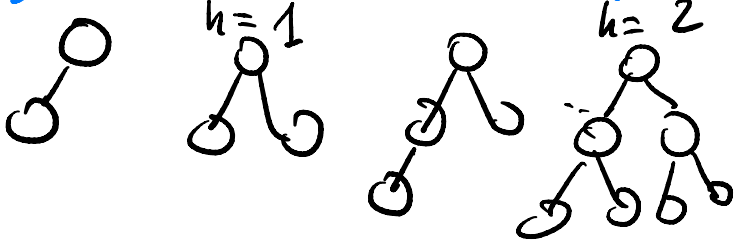
h = altezza$(T) = 3$

$\log_2(n+1) - 1 \leq h \leq n-1$

$n+1 \leq 2^{h+1}$ ⇔ $h+1 \geq \log_2(n+1)$

$2^h - 1 < n \leq 2^{h+1} - 1$

& ogni livello pieno e l'ultimo contiene il resto dei nodi



albero pieno: $2^{h+1} - 1$ nodi

```
1  Decomponibile(u):
2    IF (u == null) {
3      RETURN Decomponibile(null);
4    } ELSE {
5      risultatoSX = Decomponibile(u.sx);
6      risultatoDx = Decomponibile(u.dx);
7      RETURN Ricombina(risultatoSX, risultatoDx);
8    }
```

CASO BASE

DIVIDE et IMPERA per alberi

O(n.o di nodi)