

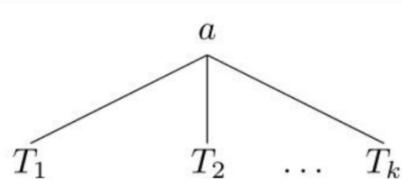
DEFINIZIONE

A : insieme delle etichette

$T(A)$: insieme di alberi su A

$a \in A \wedge T_1 \in T(A) \wedge T_2 \in T(A) \wedge \dots \wedge T_k \in T(A)$ con $k \geq 0$

$$\Rightarrow \underbrace{\{a, T_1, T_2, \dots, T_k\}}_{\text{albero}} \in T(A)$$

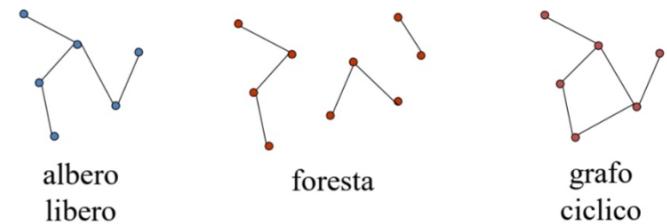


OSSERVAZIONE

- un singolo nodo con etichetta in A è un albero
- l'albero vuoto non fa parte di $T(A)$

ALBERO = GRAFO CONNESSO ACICLICO

FORESTA = insieme di alberi



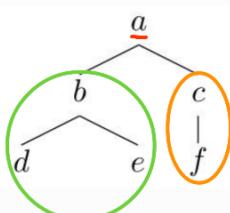
RAMO = cammino dalla radice ad una foglia

LIVELLO di un nodo = numero degli archi dal nodo alla radice

ALTEZZA dell'albero = livello del nodo di livello massimo

GRADO dell'albero = numero dei figli del nodo con più grande numero di figli

esempio:



a = radice (livello 0)

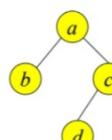
d, e, f = foglie (livello 2)

altezza = 2 grado = 2

Può essere descritto con una stringa: $\{a, \{b, \{d\}, \{e\}\}, \{c, \{f\}\}\}$

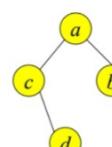
ALBERO ORDINATO \rightarrow l'ordine in cui appaiono i figli non conta

Come alberi non ordinati sono $=$, cioè uguali.



\neq

$=$



Come alberi ordinati sono \neq , cioè diversi.

ALBERI BINARI POSIZIONALI

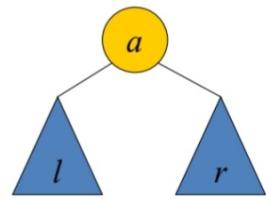
→ Alberi binari in cui l'ordine dei nodi conta

DEFINIZIONE:

$BT(A) = \text{insieme degli alberi binari etichettati in } A$

a) $\emptyset \in BT(A)$

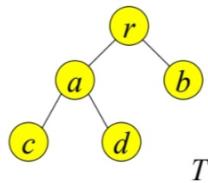
b) $a \in A, l \in BT(A), r \in BT(A) \Rightarrow \{a, l, r\} \in BT(A)$



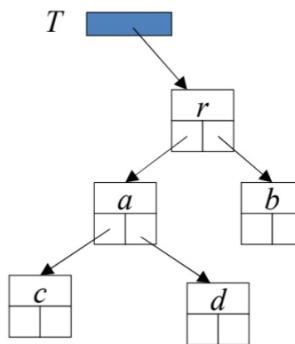
l: sottoalbero sinistro

r: sottoalbero destro

CON PUNTATORI:



key	
left	right



ALBERI K-ARI

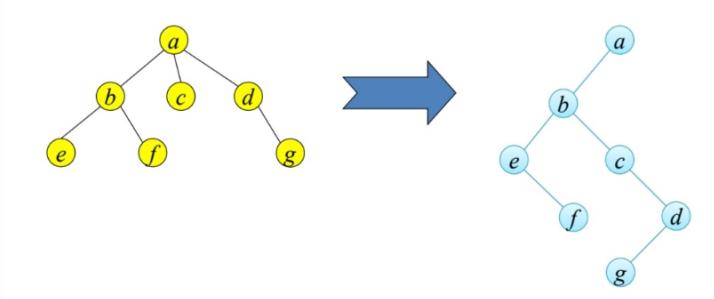
- In ogni nodo : - etichetta (key)

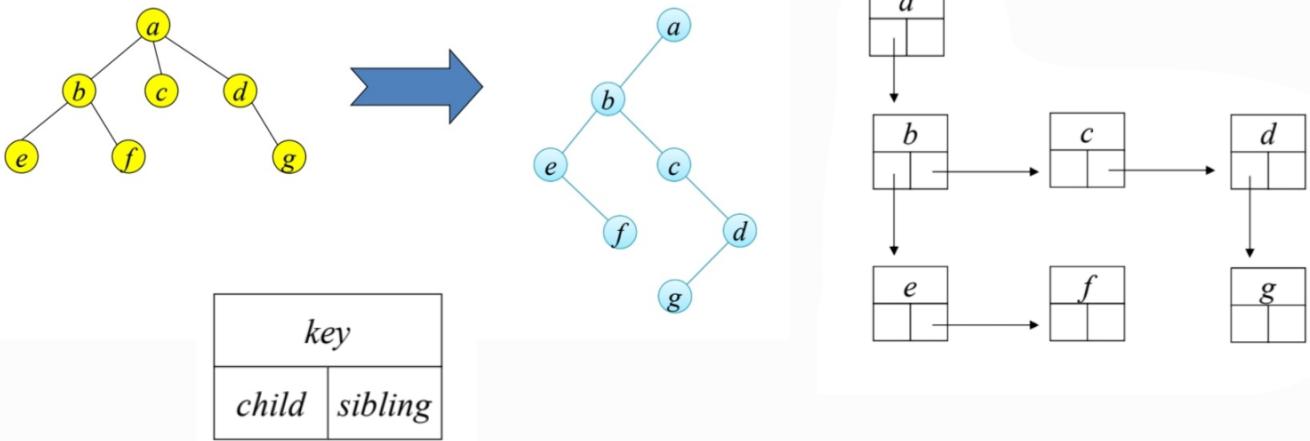
- K puntatori

deciso a priori

oppure : - etichetta (key)

- lista di puntatori → meno memoria occupata





CARDINALITA' = numero di nodi dell'albero

2-TREE-CARD(2-Tree T)

```

if  $T = \text{nil}$  then
    return 0
else
     $l \leftarrow \text{2-TREE-CARD}(T.\text{left})$ 
     $r \leftarrow \text{2-TREE-CARD}(T.\text{right})$ 
    return  $l + r + 1$ 
end if
```

k -TREE-CARD(k -Tree T)

```

if  $T = \text{nil}$  then
    return 0
else
     $card \leftarrow 1$ 
     $C \leftarrow T.\text{child}$ 
    while  $C \neq \text{nil}$  do
         $card \leftarrow card + k\text{-TREE-CARD}(C)$ 
         $C \leftarrow C.\text{sibling}$ 
    end while
    return  $card$ 
end if
```

ALTEZZA

2-TREE-HIGHT(2-Tree T) \triangleright pre: T non è vuoto

if $T.\text{left} = \text{nil}$ and $T.\text{right} = \text{nil}$ **then**

return 0 $\triangleright T$ ha un solo nodo

else

$hl, hr \leftarrow 0$

if $T.\text{left} \neq \text{nil}$ **then**

$hl \leftarrow \text{2-TREE-HIGHT}(T.\text{left})$

end if

if $T.\text{right} \neq \text{nil}$ **then**

$hr \leftarrow \text{2-TREE-HIGHT}(T.\text{right})$

end if

return $1 + \max\{hl, hr\}$

end if

k TREE-HIGHT(T)

\triangleright pre: T non è vuoto

if $T.\text{child} = \text{nil}$ **then**

return 0 $\triangleright T$ ha un solo nodo

else

$h \leftarrow 0$

$C \leftarrow T.\text{child}$

while $C \neq \text{nil}$ **do**

$h \leftarrow \max\{h, k\text{-TREE-HIGHT}(C)\}$

$C \leftarrow C.\text{sibling}$

end while

return $h + 1$

end if

VISITE

Visita (completa) : ispezione dei nodi dell'albero in cui ciascun nodo viene visitato esattamente una volta.

VISITA IN PROFONDITÀ

- Scende lungo un ramo fino ad una foglia;
- Poi, risalendo, ricomincia a scendere appena trova nodi non visitati.

TREE-DFS(k -Tree T)

```
visita  $T.key$ 
 $C \leftarrow T.child$ 
while  $C \neq nil$  do
    TREE-DFS( $C$ )
     $C \leftarrow S.sibling$ 
end while
```

- Scendendo lungo un ramo, posso memorizzare in una PILA i nodi da cui riprendere la discesa

TREE-DFS-STACK(k -Tree T)

▷ pre: T non è vuoto

```
 $S \leftarrow$  pila vuota
Push( $S, T$ )
while  $S \neq$  la pila vuota do
     $T' \leftarrow Pop(S)$ 
    visita  $T'.key$ 
    for all  $C$  figlio di  $T'$  do
        Push( $S, C$ )
    end for
end while
```

VISITA IN AMPIEZZA

- visita l'albero livello per livello
- si può ottenere con una COOA

TREE-BFS(k -Tree T)

▷ pre: T non è vuoto

```
 $Q \leftarrow$  coda vuota
Enqueue( $Q, T$ )
while  $Q \neq$  la coda vuota do
     $T' \leftarrow Dequeue(Q)$ 
    visita  $T'.key$ 
    for all  $C$  figlio di  $T'$  do
        Enqueue( $Q, C$ )
    end for
end while
```

⇒ Il tipo di visita dipende dalla struttura di appoggio usata

- Tutti: $O(n)$