Lezione 28 – Struttura dati di Albero ("alberi1")

- Strutture di dati dinamiche; dopo la lista ...
- Definizione Formale di Albero
- Rappresentazione Concreta del Tipo Albero
- Concetto di Visita di Albero
  - Algoritmi di Visita
  - differenza tra Analisi e Visita ...
- ...

Strutture di dati ... statiche e dinamiche

TABELLA
tradizionale esempio di struttura di dati STATICA
(ma l'abbiamo realizzata anche con array riallocati dinamicamente)

REALIZZA UNA **COLLEZIONE di INFORMAZIONI**
ORGANIZZAZIONE delle informazioni
- **SEQUENZIALE**
- in **blocchi di memoria allocati sequenzialmente**

LISTA
struttura DINAMICA

REALIZZA UNA **COLLEZIONE di INFORMAZIONI**
ORGANIZZAZIONE delle informazioni
- in NODI ALLOCATI DINAMICAMENTE
- **GERARCHICA**
- **LINEARE**

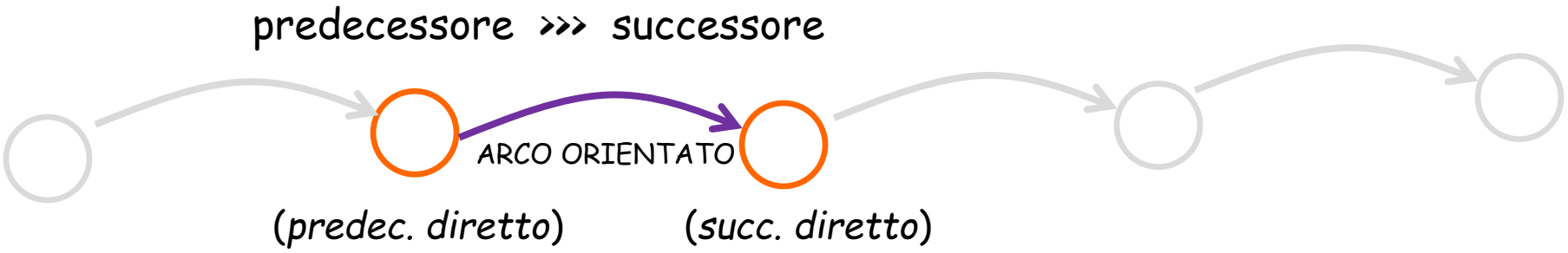
Strutture di dati ... statiche e dinamiche

TABELLA
tradizionale esempio di struttura di dati STATICA
(ma l'abbiamo realizzata anche con array riallocati dinamicamente)

REALIZZA UNA **COLLEZIONE di INFORMAZIONI**
ORGANIZZAZIONE delle informazioni
- **SEQUENZIALE**
- in **blocchi di memoria allocati sequenzialmente**

LISTA **struttura DINAMICA**

REALIZZA UNA **COLLEZIONE di INFORMAZIONI**
ORGANIZZAZIONE delle informazioni
- in NODI ALLOCATI DINAMICAMENTE
- **GERARCHICA**
- **LINEARE**

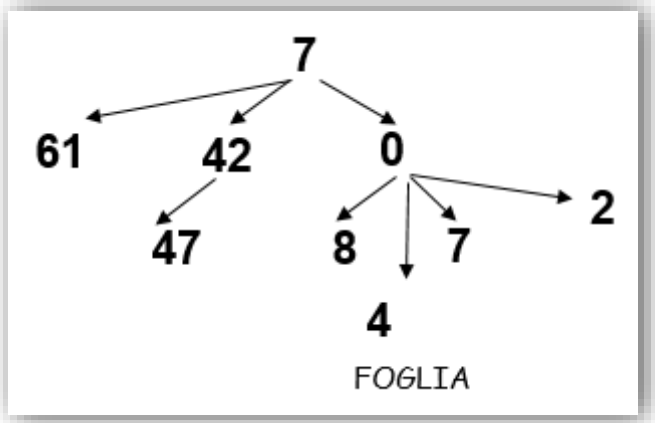


ALBERO
struttura DINAMICA
REALIZZA UNA **COLLEZIONE di INFORMAZIONI**

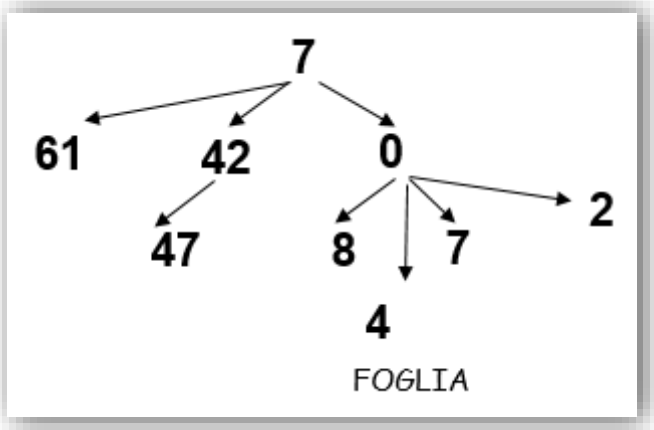
ORGANIZZAZIONE delle informazioni
- in NODI ALLOCATI DINAMICAMENTE

- **GERARCHICA**
  (predecessore >>> successore)

- **NON LINEARE**
  - per ogni NODO possibili **più successori diretti** (da 0 a N)
    (N = ARITA` dell'albero)
  - per ogni NODO possibile **1 SOLO predecessore** diretto



ALBERO
struttura DINAMICA
REALIZZA UNA **COLLEZIONE di INFORMAZIONI**



ORGANIZZAZIONE delle informazioni
- in NODI ALLOCATI DINAMICAMENTE

- **GERARCHICA**
  (predecessore >>> successore)

- **NON LINEARE**
  - per ogni NODO possibili **più successori diretti** (da 0 a N)
    (N = ARITA` dell'albero)

  - per ogni NODO possibile **1 SOLO predecessore** diretto

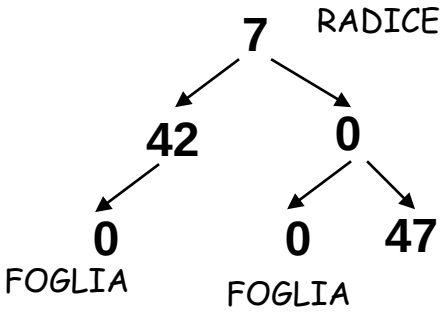
In particolare, c'è un unico "nodo iniziale", come nelle liste ...
ESISTE ED è UNICO il NODO RADICE
RADICE = NODO privo di predecessori

I "nodi finali"
possono invece essere tanti FOGLIE = NODI privi di successori

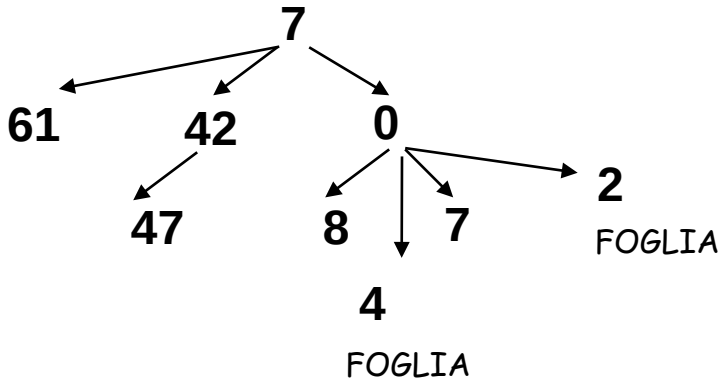
ESEMPI DI ALBERI

ALBERO struttura DINAMICA
(**NODI** in memoria) , per **COLLEZIONE di INFORMAZIONI**
ORGANIZZAZIONE delle informazioni **GERARCHICA, NON LINEARE**

albero binario (arità 2)



albero quaternario (arità 4)

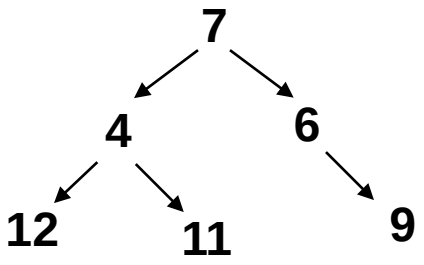


ALBERO VUOTO
= collezione vuota

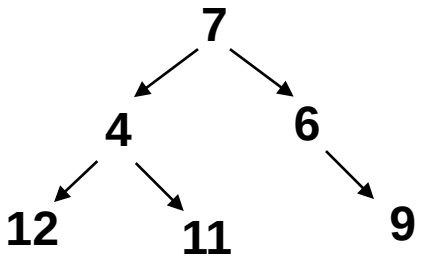
ALBERO SINGLETON = un solo nodo, che è radice e foglia

2

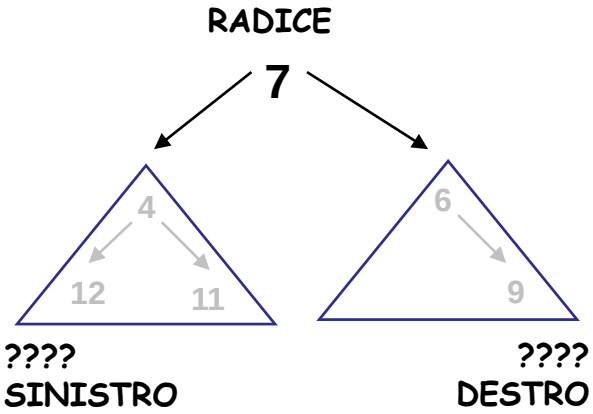
A) generico albero binario



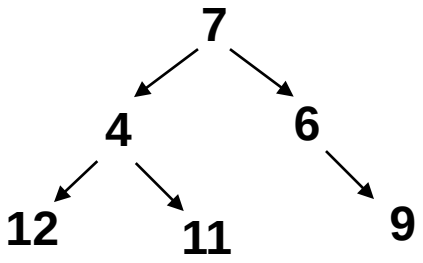
(A) generico albero binario



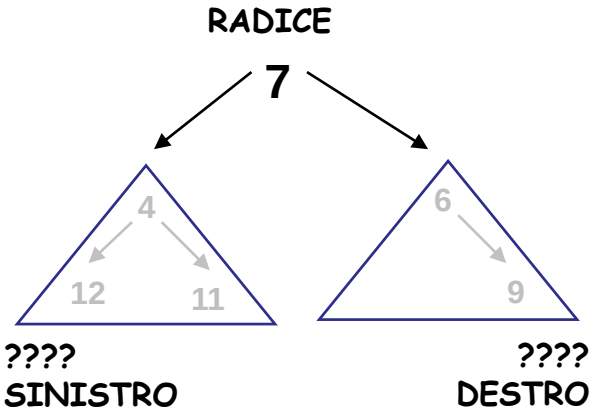
e` riconoscibile una struttura come



(A) Generico albero binario

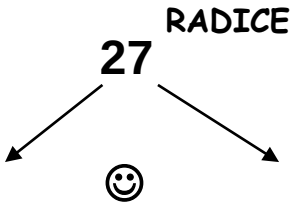


è riconoscibile una struttura come

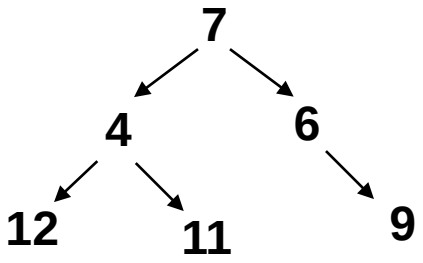


Albero VUOTO (non ha la struttura come sopra …. è un caso a se' stante)

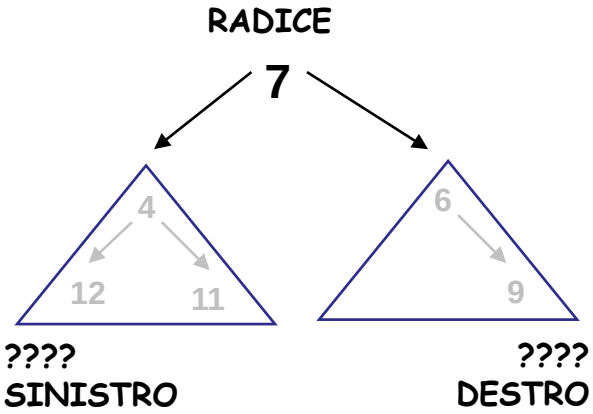
(S) Albero SINGLETON … uhm … ha la struttura vista sopra!



(A) Generico albero binario

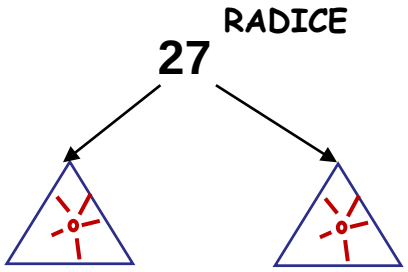


e` riconoscibile una struttura come



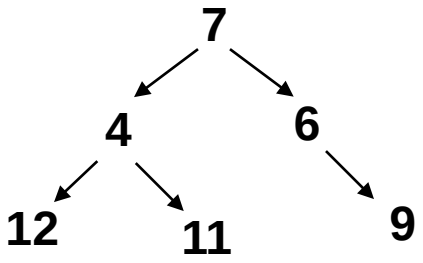
Albero VUOTO (non ha la struttura come sopra …. è un caso a se' stante)

(S) Albero SINGLETON … uhm … ha la struttura vista sopra!

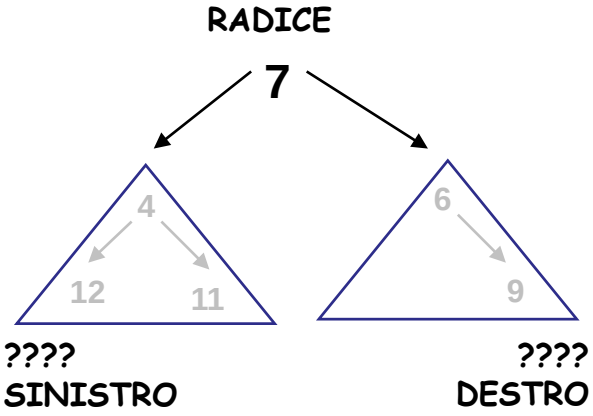


ALBERI (BINARI) - classificazione

(A) Generico albero binario

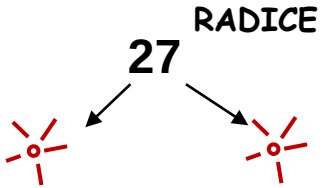


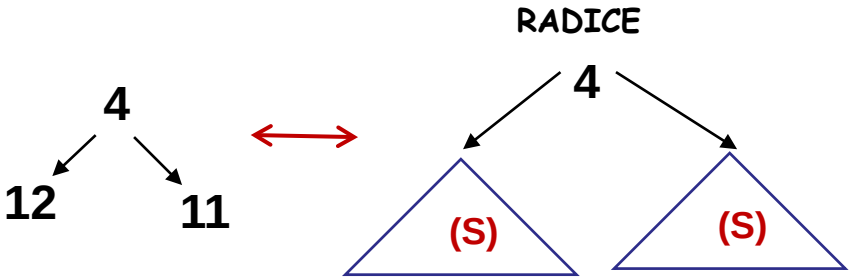
e` riconoscibile una struttura come

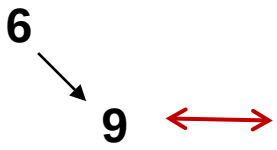


Albero VUOTO (non ha la struttura come sopra …. è un caso a se' stante)

(S) Albero SINGLETON … uhm … ha la struttura vista sopra!



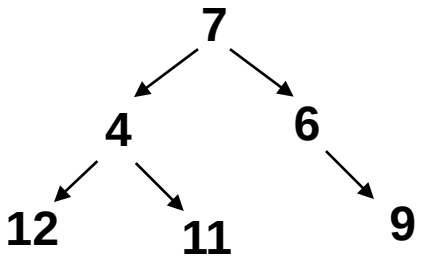




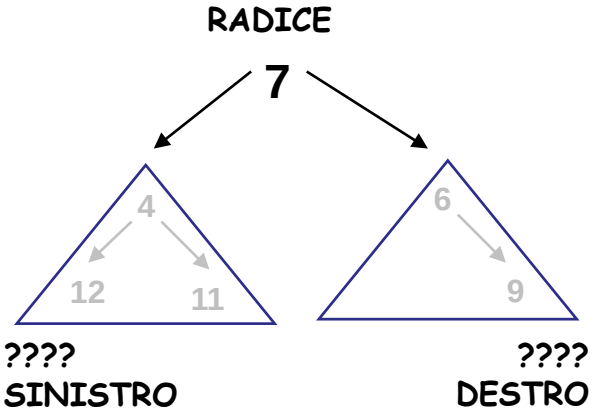
☺

ALBERI (BINARI) - classificazione

(A) Generico albero binario

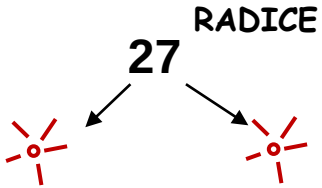


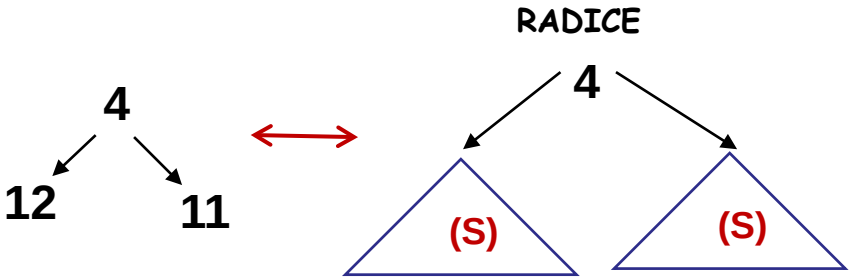
e` riconoscibile una struttura come

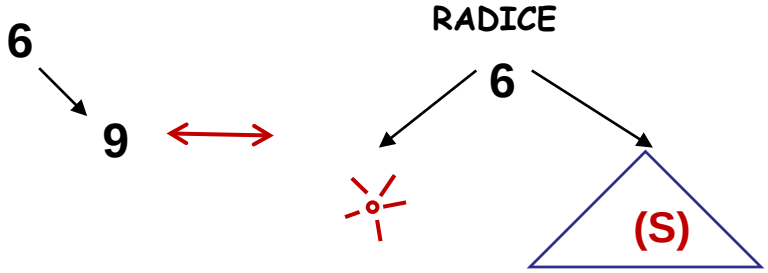


Albero VUOTO (non ha la struttura come sopra …. è un caso a se' stante)

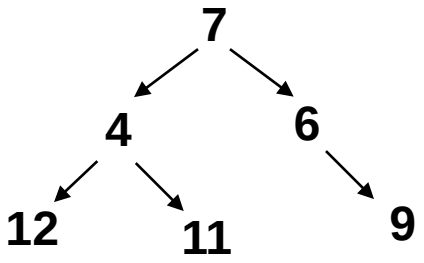
(S) Albero SINGLETON … uhm … ha la struttura vista sopra!



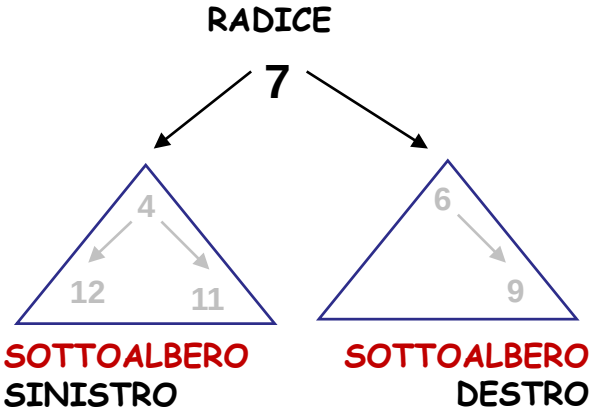




(A) Generico albero binario



e` riconoscibile una struttura come

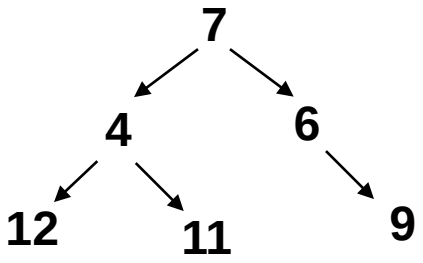


Un albero (collezione di NODI)
non vuoto
ha una RADICE e due sottoalberi (sottoinsiemi disgiunti di nodi dell'albero, che sono a loro volta organizzati come ALBERI)

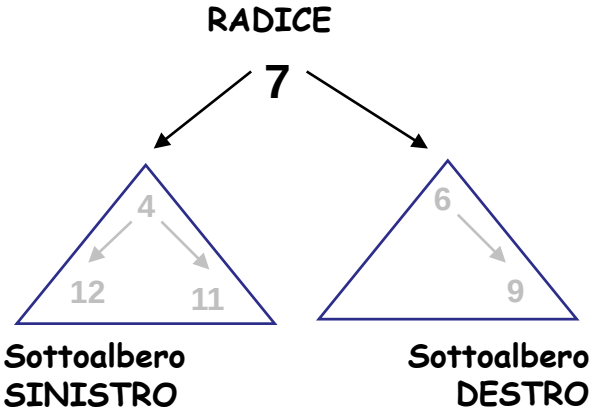
$$
\text{ALB} = \begin{cases} \text{VUOTO} \\ \text{NON VUOTO} \begin{cases} \text{RADICE} \\ \text{SIN} \quad \text{(SOTTOALBERO SINISTRO)} \\ \text{DES} \quad \text{(SOTTOALBERO DESTRO)} \end{cases} \end{cases}
$$

ALBERI (BINARI) - classificazione

(A) Generico albero binario

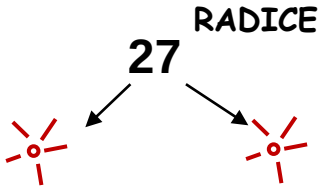


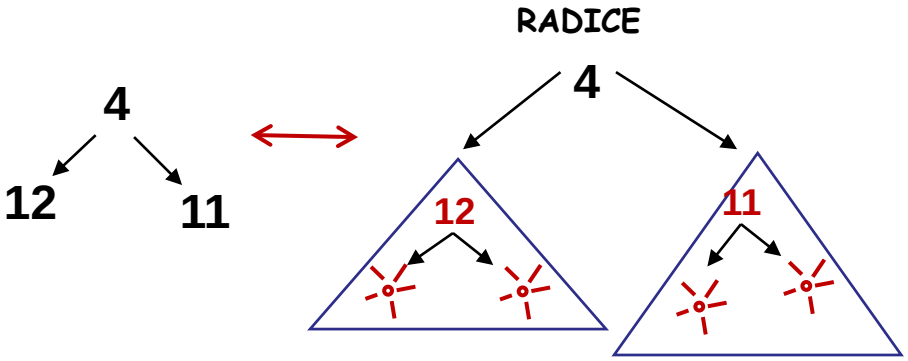
e` riconoscibile una struttura come

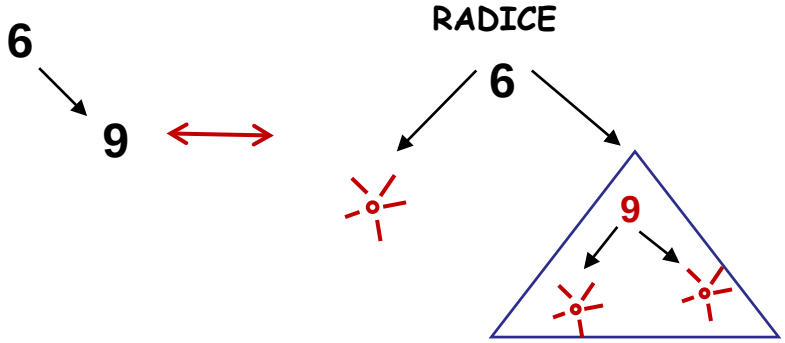


Albero VUOTO (non ha la struttura come sopra …. è un caso a se' stante)

(S) Albero SINGLETON … uhm … ha la struttura vista sopra!







Dalla definizione intrinsecamente ricorsiva della struttura di albero, discende la possibilità di definire algoritmi ricorsivi su tali strutture.

$$\text{ALB} = \begin{cases} \text{VUOTO} \\ \text{NON VUOTO} \begin{cases} \text{RADICE} \\ \text{SIN} \quad \text{(SOTTOALBERO SINISTRO)} \\ \text{DES} \quad \text{(SOTTOALBERO DESTRO)} \end{cases} \end{cases}$$

Una funzione F_{OP}, che deve eseguire sui nodi di un albero ALB l'operazione OP, può essere definita come segue

$$F_{OP}(\text{ALB}) = \begin{cases} \text{se ALB VUOTO} & \text{NULLA} \\ \text{sennò, ci sono} \begin{cases} \text{RADICE} \\ \text{SIN} \\ \text{DES} \end{cases} & \text{e facciamo} \begin{cases} - \text{ OP su RADICE} \\ - \ F_{OP}(\text{SIN}) \\ - \ F_{OP}(\text{DES}) \end{cases} \end{cases}$$

?PRIMA SIN e POI DES

oppure

PRIMA DES e POI SIN?

la natura non lineare della struttura crea delle alternative, che di solito sono risolte in base alle abitudini …

precedenza a SIN

$F_{OP}(ALB)$ =

- se ALB VUOTO → NULLA
- senno`, ci sono →
  - RADICE → - OP su RADICE
  - SIN → - $F_{OP}(SIN)$
  - DES → - $F_{OP}(DES)$

precedenza a DES

$F_{OP}(ALB)$ =

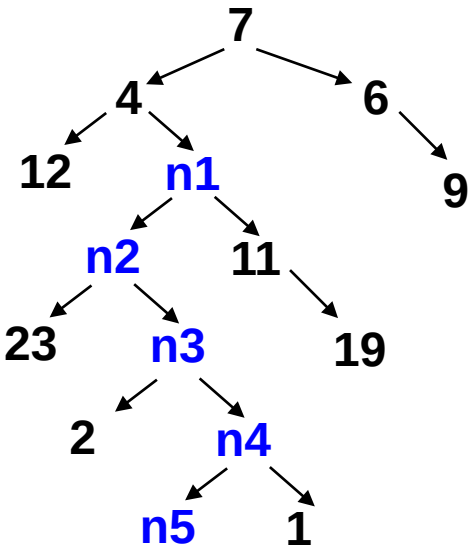
- se ALB VUOTO → NULLA
- senno`, ci sono →
  - RADICE → - OP su RADICE
  - SIN → - $F_{OP}(DES)$
  - DES → - $F_{OP}(SIN)$

Come reperire i dati conservati in un albero

Un cammino (n_1 , n_k)
di lunghezza (k-1)
è una sequenza di nodi

$n_1, n_2, ..., n_k$

in cui, per ogni i in [1,k-1]
è n_i pred $n_{(i+1)}$



Come reperire i dati conservati in un albero

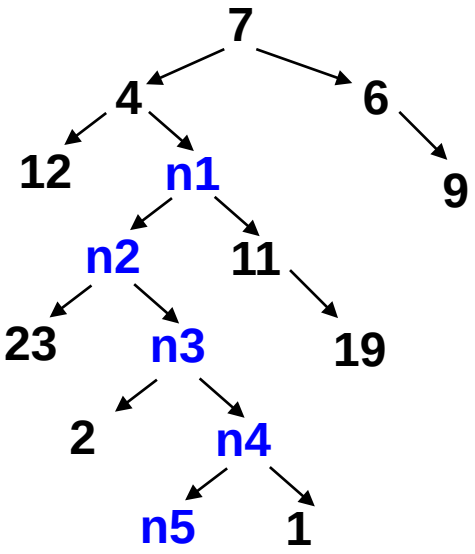
Se si cerca un dato in particolare (es. 99), bisogna "visitare" l'albero alla sua ricerca, cioè scoprire un CAMMINO che porti dalla radice al nodo dove c'è l'informazione cercata

In generale, un **cammino** è un percorso di nodi, che unisce un nodo di partenza ad un nodo di arrivo, passando da un nodo all'altro secondo la relazione predecessore-successore.

Un cammino (n_1 , n_k)
di lunghezza (k-1)
è una sequenza di nodi

$$n_1, n_2, \ldots, n_k$$

in cui, per ogni i in [1,k-1]
è n_i pred $n_{(i+1)}$

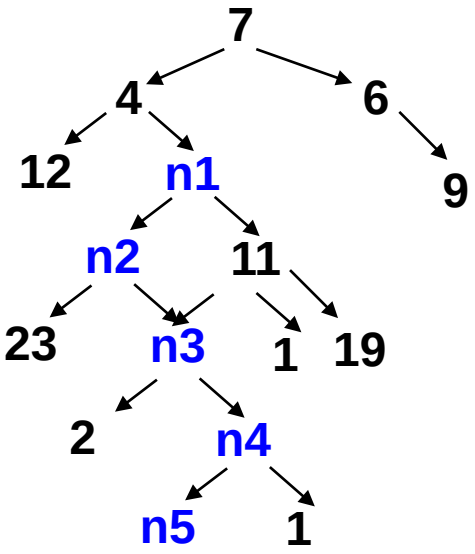


Come reperire i dati conservati in un albero

Se si cerca un dato in particolare (es. 99), bisogna "visitare" l'albero alla sua ricerca, cioè scoprire un CAMMINO che porti dalla radice al nodo dove c'è l'informazione cercata

In generale, un **cammino** è un percorso di nodi, che unisce un nodo di partenza ad un nodo di arrivo, passando da un nodo all'altro secondo la relazione predecessore-successore.

Un cammino (n_1 , n_k)
di lunghezza (k-1)
è una sequenza di nodi
n_1, n_2, ..., n_k
in cui, per ogni i in [1,k-1]
è n_i pred $n_{(i+1)}$



Proprietà in un albero

1) dati due nodi qualsiasi, n, m, dell'albero ALB, un cammino che li collega potrebbe non esistere (Esempio, 6 e 19 …)

2) dati due nodi n,m di un albero ALB,
se esiste, un cammino (n,m) è unico.

3) per ogni nodo, n, dell'albero ALB, esiste ed è unico il cammino (RADICE, n)

Livello di un nodo nell'albero

Il livello di un nodo, n, è la lunghezza del cammino (RADICE, n).

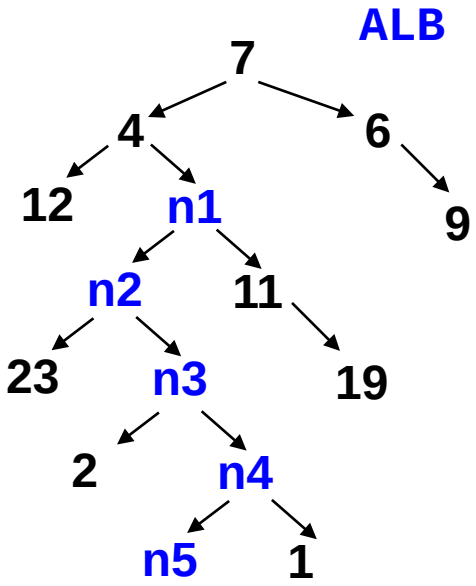
`livello(nodo, ALB)`

Ad esempio,

livello(12, ALB) = 2

livello(n5, ALB) = 6

livello(19, ALB) = 4



In generale,

livello (succ(n), ALB) = 1 + livello (n, ALB)

es.

livello(19, ALB) = 1 + livello(11, ALB)

livello(RADICE,ALB) = 0

Livello di un nodo nell'albero, e Profondità

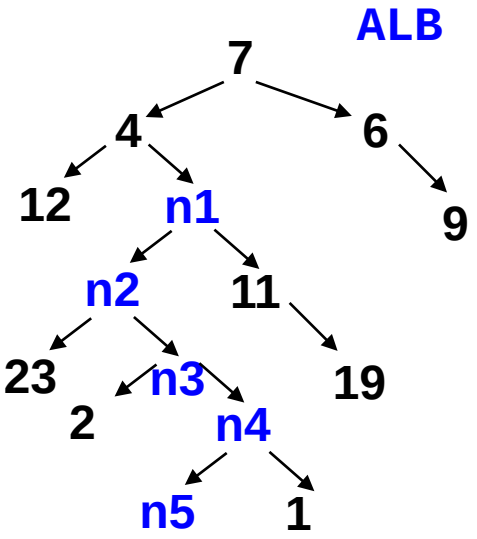
Il livello di un nodo, n, è la lunghezza del cammino (RADICE, n).

`livello(nodo, ALB)`

Ad esempio,
livello(12, ALB) = 2 livello(n5, ALB) = 6 livello(19, ALB) = 4

livello (succ(n), ALB) = 1 + livello (n, ALB)

livello(RADICE,ALB) = 0



Profondità dell'albero

Si tratta del livello massimo raggiunto dai nodi foglia

Ad esempio:

livello(12,ALB) = 2 livello(9,ALB) = 2

livello(19,ALB) = 4 livello(23,ALB) = 4 livello(2,ALB) = 5

livello(n5,ALB) = 6 livello(1,ALB) = 6 ...

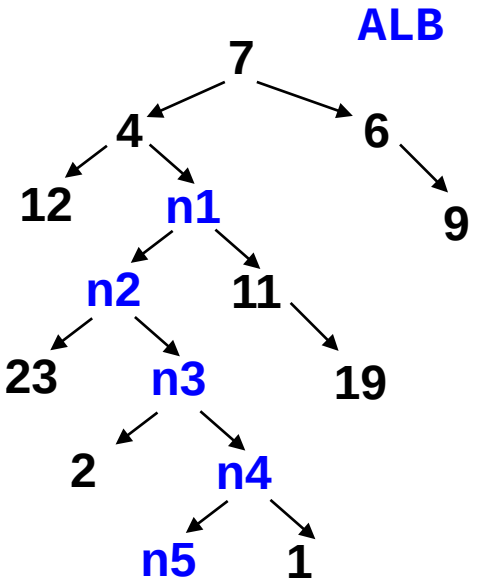
quindi la profondità di ALB è 6

Calcolo del livello di un nodo in un albero

Il livello di un nodo, n, si calcola (ha senso) solo per alberi non vuoti

$$LIV(N,ALB) = \begin{cases} \text{se } N=\text{RADICE} & 0 \\ \text{senno`, con M predecessore di N} & 1 + LIV(M, ALB) \end{cases}$$

(ALB NON VUOTO)



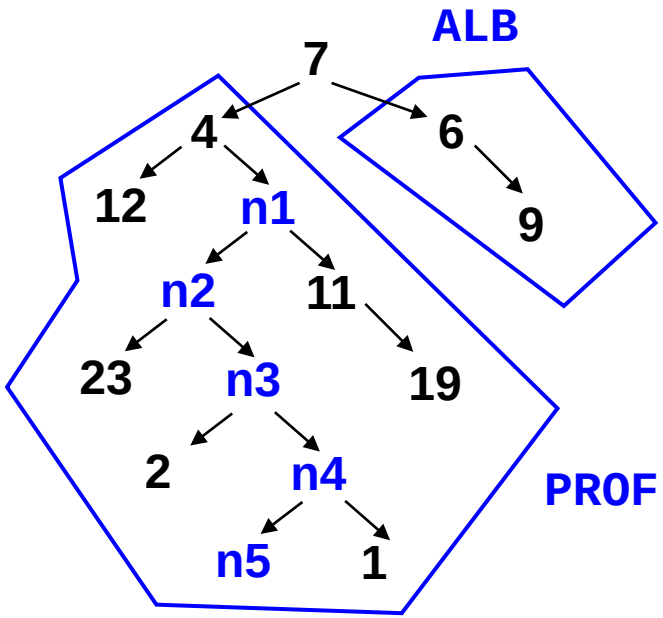
Calcolo della Profondità di un albero

La profondità dell'albero VUOTO ci da` qualche istruttivo problema; ma il principio di calcolo è il medesimo usato per il livello di un nodo …

La profondità di ALB dipende dalla profondità dei sottoalberi

Nell'esempio, il SIN di ALB ha profondità 5, il DES ha profondità 1, e quindi la profondità di ALB è 6

prof(ALB) = 1 + max (prof(SIN), prof(DES))

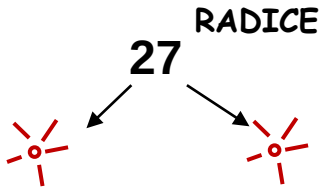


$$PROF(ALB) = \begin{cases} \text{ALB NON VUOTO} & 1 + \max(PROF(SIN), PROF(DES)) \\ \text{se ALB VUOTO} & ?? \end{cases}$$

Calcolo della profondita` di un albero

... dunque ...

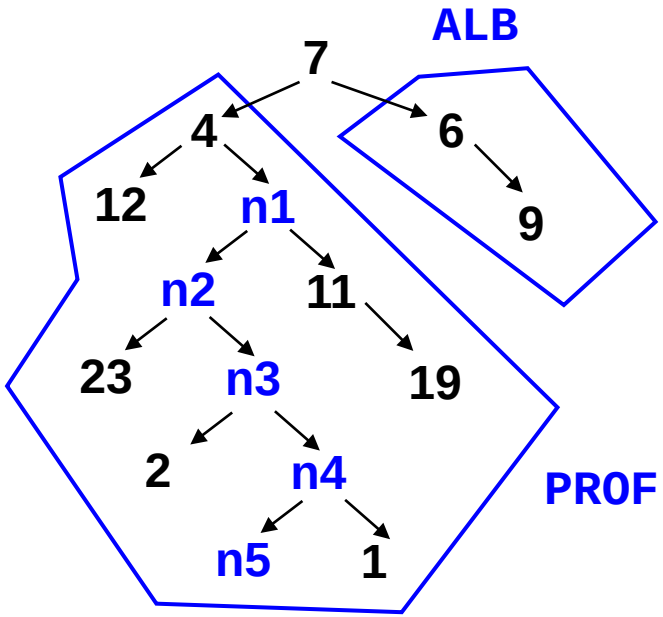
sappiamo che PROF(RADICE) = 0

e quindi PROF (

) = 0 = 1 + PROF()

perciò perché si mantenga valida l'espressione 1+max(…)

deve essere PROF() = ☺

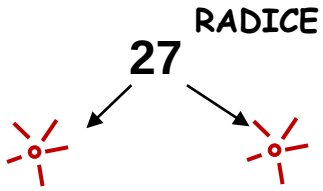


$$PROF(ALB) = \begin{cases} \text{ALB NON VUOTO} & 1 + \max(PROF(SIN), PROF(DES)) \\ \text{se ALB VUOTO} & ☺ \end{cases}$$

Calcolo della profondita` di un albero

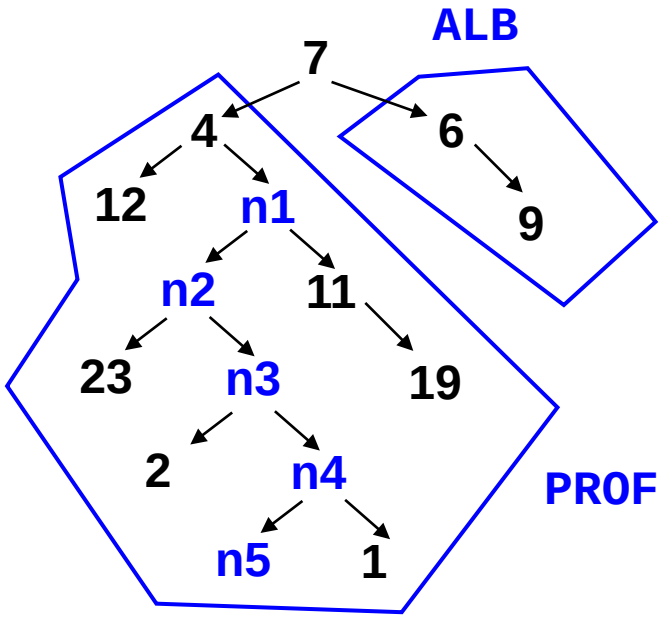
... dunque ...

sappiamo che PROF(RADICE) = 0

e quindi PROF (

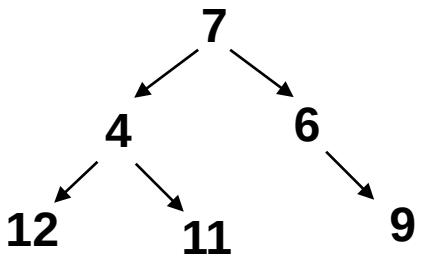
) = 0 = 1 + PROF()

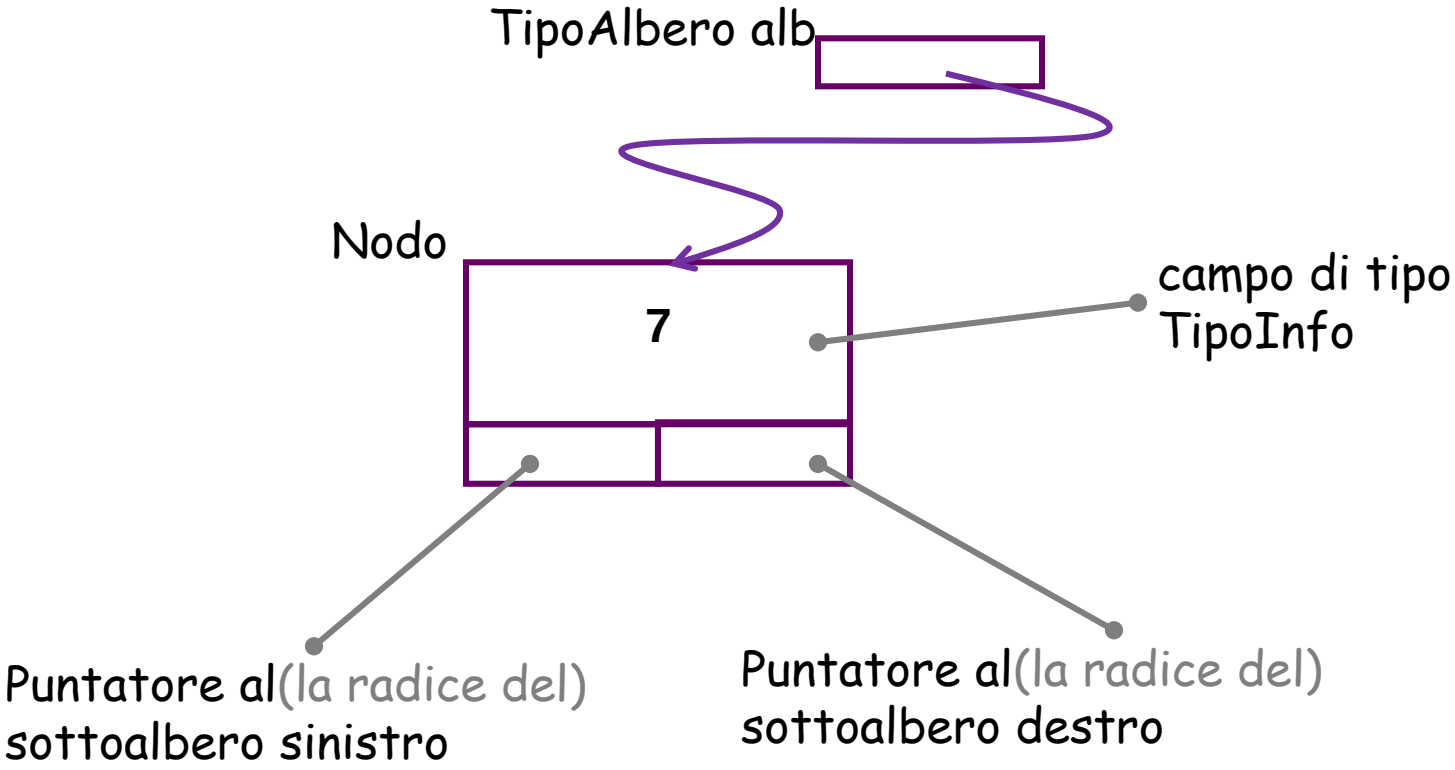
perciò perché si mantenga valida l'espressione 1+max(...)
deve essere PROF() = -1



$$PROF(ALB) = \begin{cases} \text{ALB NON VUOTO} & 1 + \max(PROF(SIN), PROF(DES)) \\ \text{se ALB VUOTO} & -1 \end{cases}$$

Rappresentazione concreta di albero in C



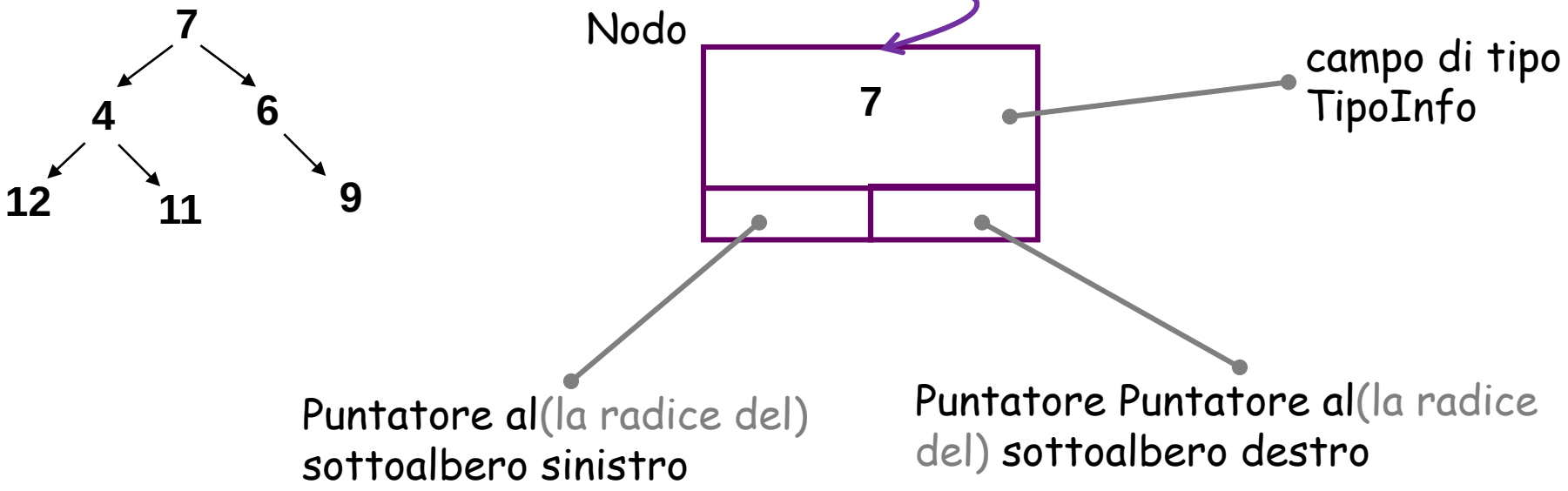


Rappresentazione concreta di albero in C

I nodi che compongono l'albero sono gestiti in modo molto simile a quel che avviene per le liste collegate

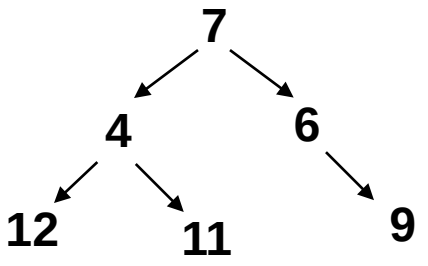
solo che ogni nodo dell'albero binario, oltre a contenere l'informazione contiene anche DUE puntatori, ai sottoalberi.

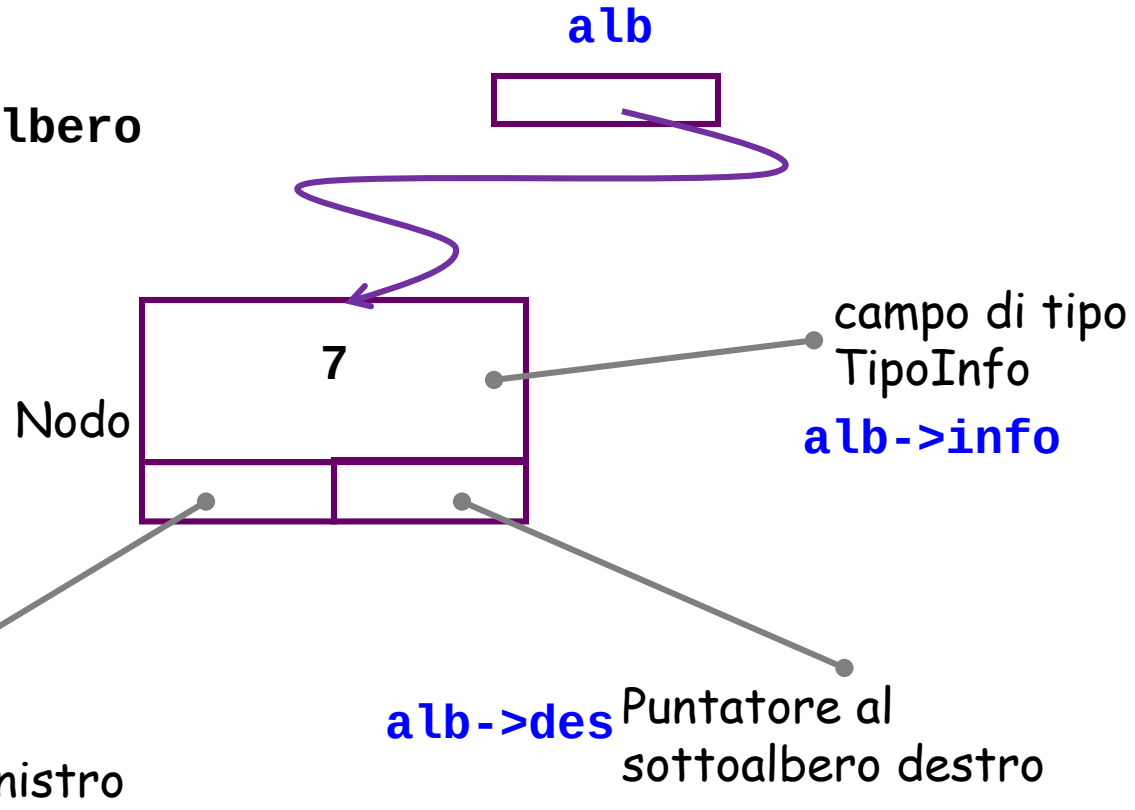
Una variabile di tipo TipoAlbero, è un puntatore, che punta alla radice dell'albero.



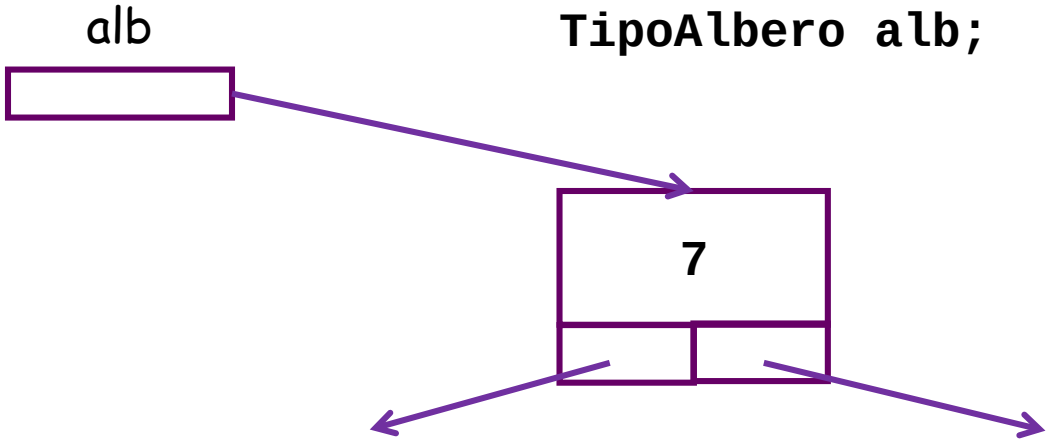
Rappresentazione concreta di albero in C

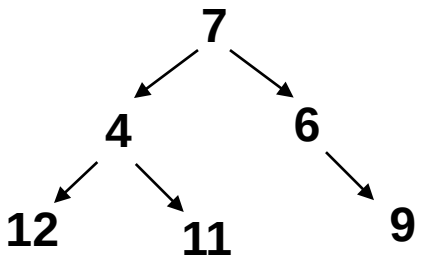
```
typedef …
        TipoInfoAlbero

struct structureAlbero {
        TipoInfoAlbero info;
        struct structureAlbero *sin;
        struct structureAlbero *des;
};

typedef struct structureAlbero
        TipoNodoAlb;

typedef TipoNodoAlb *
        TipoAlbero;

TipoAlbero alb;
```





Rappresentazione concreta di albero in C

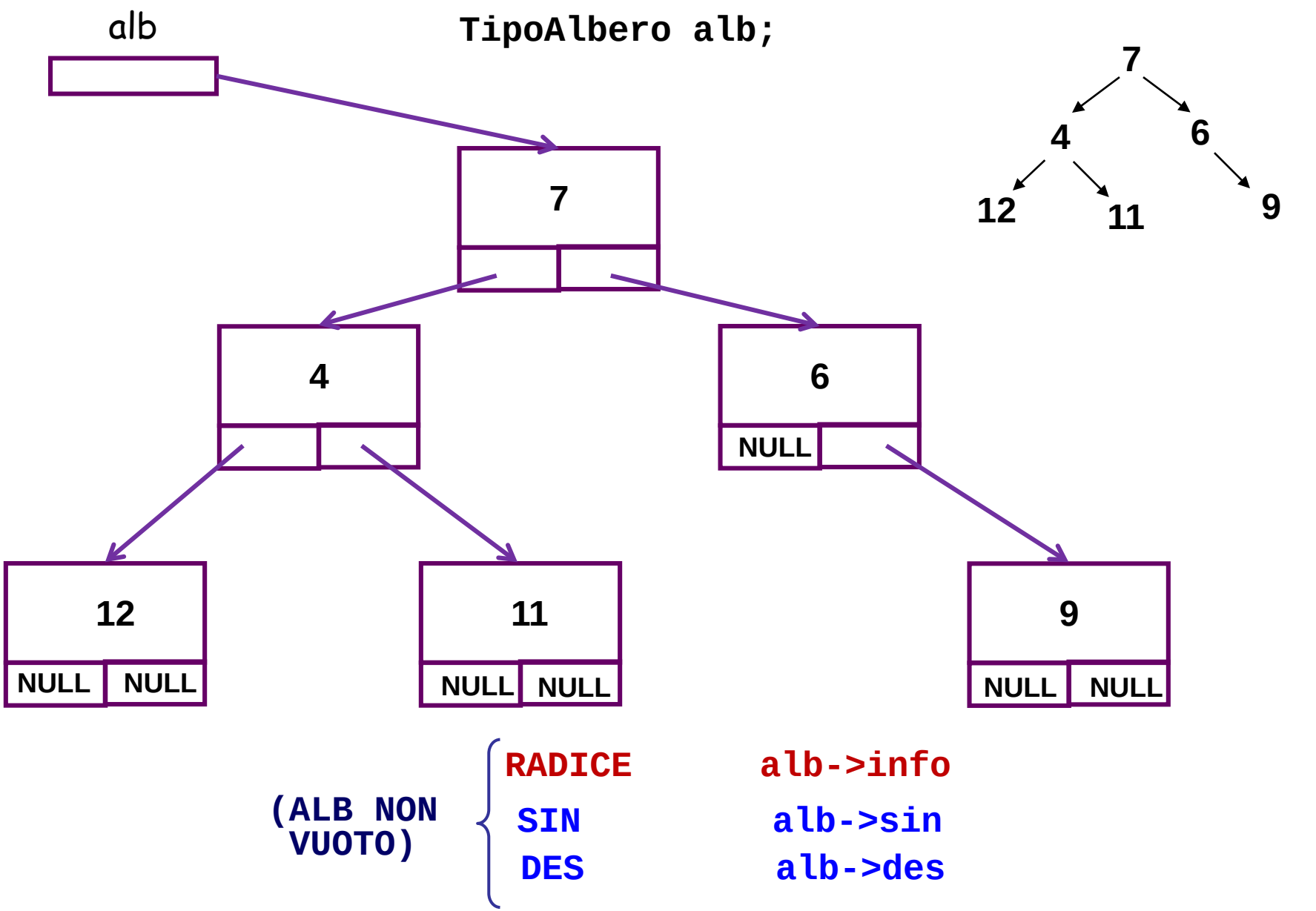




(ALB NON VUOTO)

- RADICE è alb->info
- SIN è alb->sin
- DES è alb->des

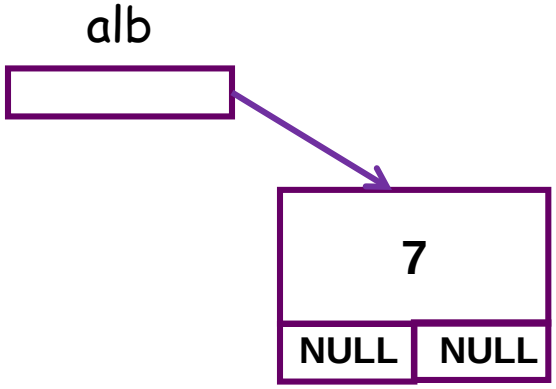
Rappresentazione concreta di albero in C



Rappresentazione concreta di albero in C


```
TipoAlbero alb;
```

albero vuoto




```
TipoAlbero alb;
```

albero singleton

Funzione per il calcolo della profondita` di un albero

$$PROF(ALB) = \begin{cases} 1 + \max(PROF(SIN), PROF(DES)) & \text{ALB NON VUOTO} \\ -1 & \text{se ALB VUOTO} \end{cases}$$

```
int prof(TipoAlbero alb) {
  int s, d; /* per le profondita` dei due sottoalberi */

  if (!alb) return -1;

  s = prof(alb->sin);
  d = prof(alb->des);

  if(s>d) return (1 + s);
  else return(1+d);
}
```

Funzione per il calcolo della profondita` di un albero

$$PROF(ALB) = \begin{cases} 1 + max(PROF(SIN), PROF(DES)) & \text{ALB NON VUOTO} \\ -1 & \text{se ALB VUOTO} \end{cases}$$

```
int prof(TipoAlbero alb) {
  int s, d; /* per le profondita` dei due sottoalberi */

  if (!alb) return -1;

  s = prof(alb->sin);
  d = prof(alb->des;

  if(s>d) return (1 + s);
  else return(1+d);
}
```

Alternativa

```
int prof(TipoAlbero alb) {
  int s, d;
  if (!alb) return -1;

  s = prof(alb->sin);
  d = prof(alb->des);

  return (s>d ? 1+s : 1+d);
}
```

Funzione per il calcolo della profondita` di un albero

$$PROF(ALB) = \begin{cases} 1 + \max(PROF(SIN), PROF(DES)) & \text{ALB NON VUOTO} \\ -1 & \text{se ALB VUOTO} \end{cases}$$

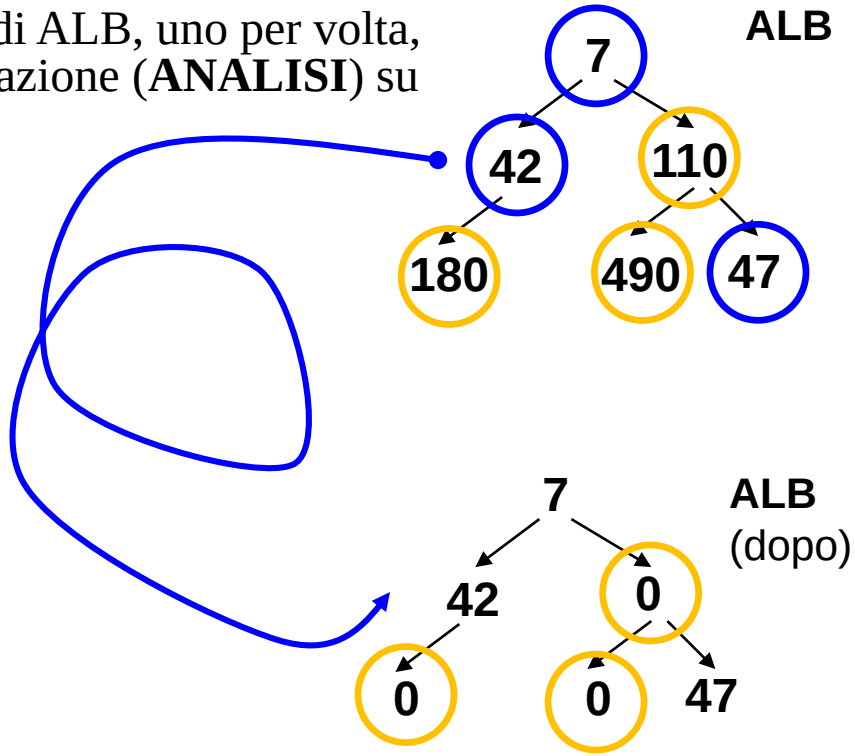
```
int prof(TipoAlbero alb) {
  int s, d; /* per le profondita` dei due sottoalberi */

  if (!alb) return -1;

  s = prof(alb->sin);
  d = prof(alb->des;

  if(s>d) return (1 + s);
  else return(1+d);
}
```

Alternativa 2

```
int prof(TipoAlbero alb) {
  int s, d;
  if (!alb) return -1;
  else return(
            s=prof(alb->sin)
                  >
            d=prof(alb->des)
         ?
            ++s; ++d)
}
```

CONCETTO di VISITA di un ALBERO

La **VISITA** e` l'elencazione / scorrimento dei nodi di ALB, uno per volta, senza perderne nessuno, fatta per eseguire una operazione (**ANALISI**) su ciascun nodo



ESEMPIO

`AzzeraSeGrande(ALB)`

funzione che riceve un albero ALB e ne azzera i nodi maggiori di 100.

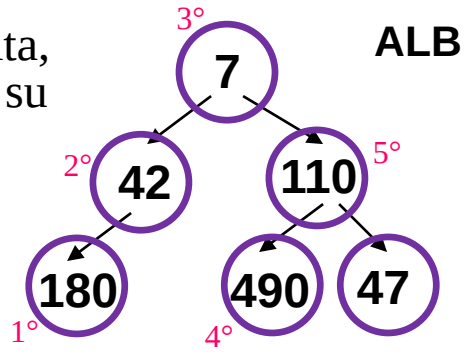
VISITA = incontrare i nodi uno per volta
ANALISI = check: se >100 azzerare

La sequenza di visita dipende dal modo in cui procediamo (cioe` dall'*algoritmo di visita* che usiamo).
Ricordiamoci della natura NON LINEARE della struttura di albero: non c'e` una sequenza univoca dei nodi ...). Questa e` la sequenza che abbiamo seguito:

?sequenza di check = 7 - 42 - 180(0) - 110(0) - 490(0) - 47

CONCETTO di VISITA di un ALBERO

La **VISITA** e` l'elencazione / scorrimento dei nodi di ALB, uno per volta, senza perderne nessuno, fatta per eseguire una operazione (**ANALISI**) su ciascun nodo



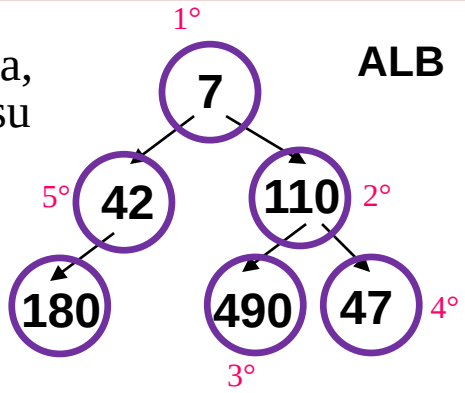
La sequenza di visita dipende dal modo in cui procediamo (cioe` dall'*algoritmo di visita* che usiamo).
Ricordiamoci della natura NON LINEARE della struttura di albero: non c'e` una sequenza univoca dei nodi ...). Ognuna delle successive e` una sequenza di visita legittima.

?sequenza di check = 7 - 42 - 180(0) - 110(0) - 490(0) - 47

?sequenza di check = 180 - 42 - 490 - 47 - 110 - 7
?sequenza di check = 7 - 110 - 490 - 47 - 42 - 180 ←
?sequenza di check = 180 - 42 - 7 - 490 - 110 - 47
?sequenza di check = 490 - 47 - 110 - 180 - 42 - 7
?sequenza di check = 7 - 42 - 110 - 180 - 490 - 47

CONCETTO di VISITA di un ALBERO

La **VISITA** e` l'elencazione / scorrimento dei nodi di ALB, uno per volta, senza perderne nessuno, fatta per eseguire una operazione (**ANALISI**) su ciascun nodo



La sequenza di visita dipende dal modo in cui procediamo (cioe` dall'*algoritmo di visita* che usiamo).
Ricordiamoci della natura NON LINEARE della struttura di albero: non c'e` una sequenza univoca dei nodi ...). Ognuna delle successive e` una sequenza di visita legittima.

?sequenza di check = 7 - 42 - 180(0) - 110(0) - 490(0) - 47

?sequenza di check = 180 - 42 - 490 - 47 - 110 - 7 ←
?sequenza di check = 7 - 110 - 490 - 47 - 42 - 180
?sequenza di check = 180 - 42 - 7 - 490 - 110 - 47
?sequenza di check = 490 - 47 - 110 - 180 - 42 - 7
?sequenza di check = 7 - 42 - 110 - 180 - 490 - 47

AzzeraSeGrande(): funzionamento logico

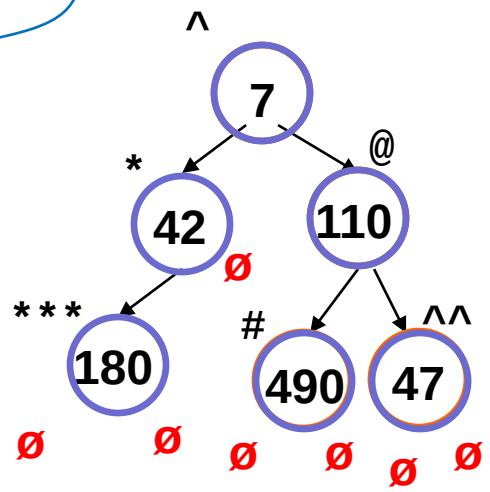
AZZERASG(ALB)=

- **VUOTO** ---
- **NON VUOTO**
  - **RAD** - se RAD>100 azzera
  - **SIN** - AZZERASG SIN
  - **DES** - AZZERASG DES

ANALISI RAD

VISITA sottoALBERI

ALB




```
AZZERASG (ALB)
    - RAD rimane 7
    - AZZERASG (*)
        - RAD (42) rimane 42
        - AZZERASG (***)
            - RAD (180) azzerata
            - AZZERASG (ø)
            - AZZERASG (ø)
        - AZZERASG (ø) [DES di 42]
    - AZZERASG (@)
        - RAD (110) azzerata
        - AZZERASG (#)
            - RAD (490) azzerato
            - AZZERASG (ø) [SIN 490]
            - AZZERASG (ø)
            ...
```

```
...
    - AZZERASG (#)
        - RAD (490) azzerato
        - AZZERASG (ø) [SIN 490]
        - AZZERASG (ø)
    - AZZERASG (^^) [DES di 110]
        - RAD (47) rimane 47
        - AZZERASG (ø) [SIN 47]
        - AZZERASG (ø)
```

void azzeraSeGrande() : implementazione e funzionamento con i RDA

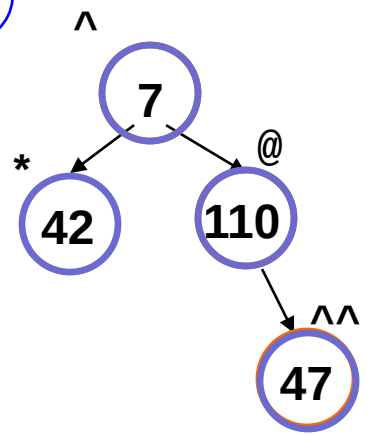
```
void azzeraSeGrande (TipoAlb alb) {
```

AZZERASG(ALB)=

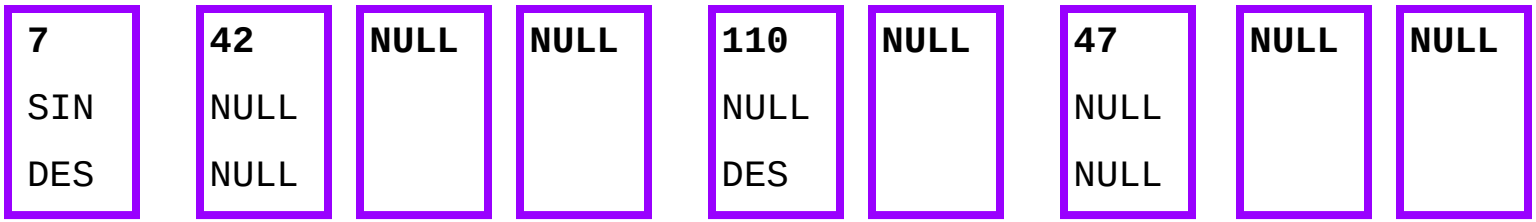
VUOTO	---
NON VUOTO	
RAD	- se RAD>100 azzera
SIN	- AZZERASG SIN
DES	- AZZERASG DES

ANALISI RAD

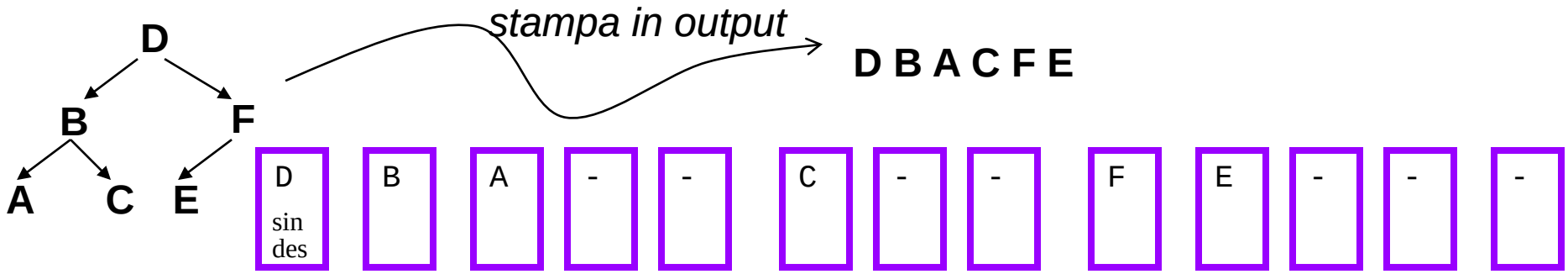
VISITA sottoALBERI




```
   if (alb) {
      if (alb->info>100)
         alb->info = 0;

      azzeraSeGrande(alb->sin);
      azzeraSeGrande(alb->des); }
return;
}
```



VISITA di un ALBERO - caso della stampa dei nodi 1/3




```
void stampaAlb1 (TipoAlb alb) {
```

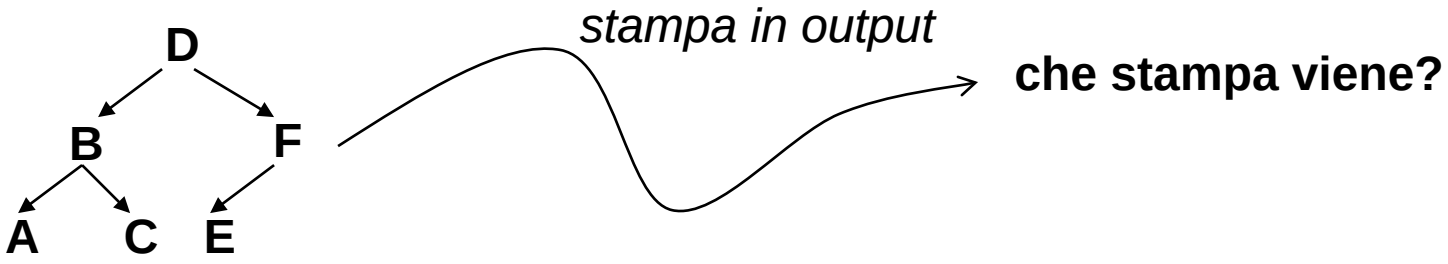
=	VUOTO	---
	else	
	RAD	1. stampa info (RAD)
	SIN	2. stampaAlb1 SIN
	DES	3. stampaAlb1 DES

```
   if (alb) {
      printf(" %c ", alb-> info);
      stampaAlb1(alb->sin);
      stampaAlb1(alb->des);
return;
}
```

NB. APPROCCIO ***precedenza alla radice e precedenza a sinistra ...***

- *prima* ***ANALISI RADICE***
- POI **VISITA** SOTTOALB SIN
- POI **VISITA** SOTTOALB DES

VISITA di un ALBERO - caso della stampa dei nodi 2/3

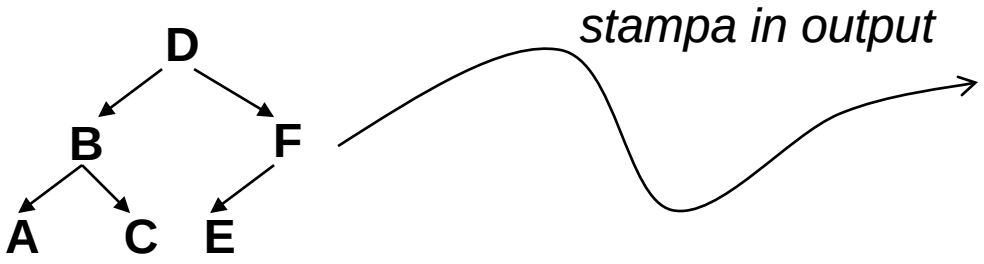



```
void stampaAlb2 (TipoAlb alb) {

                        VUOTO       ---

                    =   else
                        RAD         1. stampaAlb2 SIN
                        SIN         2. stampaAlb2 DES
                        DES         3. stampa info (RAD)

   if (alb) {
      stampaAlb2(alb->sin);
      stampaAlb2(alb->des);
      printf(" %c ", alb-> info;
return;
}
```

***NB. APPROCCIO**: precedenza ai sottoalberi e precedenza a sinistra ...*

- prima **VISITA** SOTTOALB SIN
- POI **VISITA** SOTTOALB DES
- *POI **ANALISI RADICE***

VISITA di un ALBERO - caso della stampa dei nodi 2/3



che stampa viene?

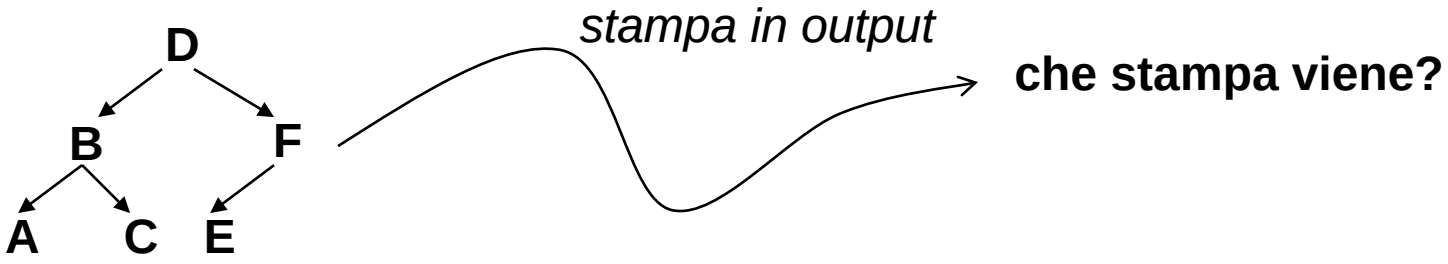
A C B E F D

```
void stampaAlb2 (TipoAlb alb) {

                    { VUOTO      ---
                    {
                    { else
                  = { RAD        1. stampaAlb2 SIN
                    { SIN        2. stampaAlb2 DES
                    { DES        3. stampa info (RAD)

   if (alb) {
      stampaAlb2(alb->sin);
      stampaAlb2(alb->des);
      printf(" %c ", alb-> info;
return;
}
```

***NB. APPROCCIO*: precedenza ai sottoalberi e precedenza a sinistra ...**

- prima **VISITA** SOTTOALB SIN
- POI **VISITA** SOTTOALB DES
- *POI **ANALISI RADICE***

VISITA di un ALBERO - caso della stampa dei nodi 3/3

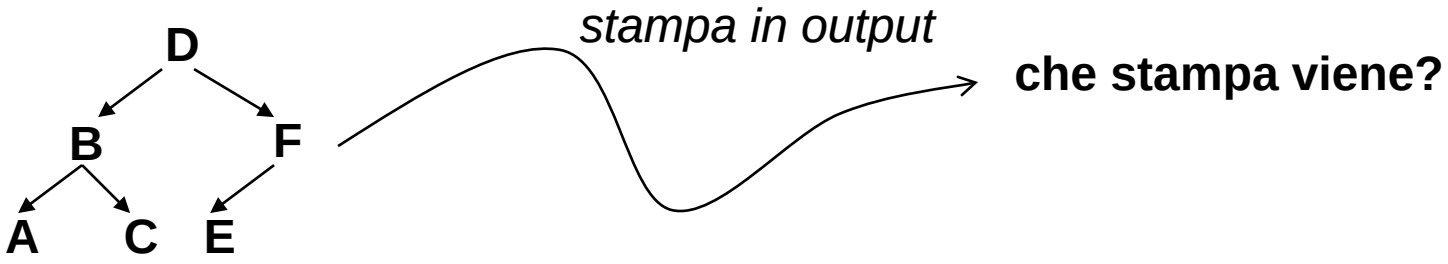



```
void stampaAlb3 (TipoAlb alb) {

                        { VUOTO        ---
                        {
                        {
                    =   { else
                        { RAD          1. stampaAlb3 SIN
                        { SIN          2. stampa info (RAD)
                        { DES          3. stampaAlb3 DES

   if (alb) {
      stampaAlb3(alb->sin);
      printf(" %c ", alb-> info;
      stampaAlb3(alb->des);
return;
}
```

***NB. APPROCCIO**: precedenza al sottoalbero sinistro ...*

- *prima* **VISITA** SOTTOALB SIN
- POI **ANALISI RADICE**
- POI **VISITA** SOTTOALB DES

VISITA di un ALBERO - caso della stampa dei nodi 3/3

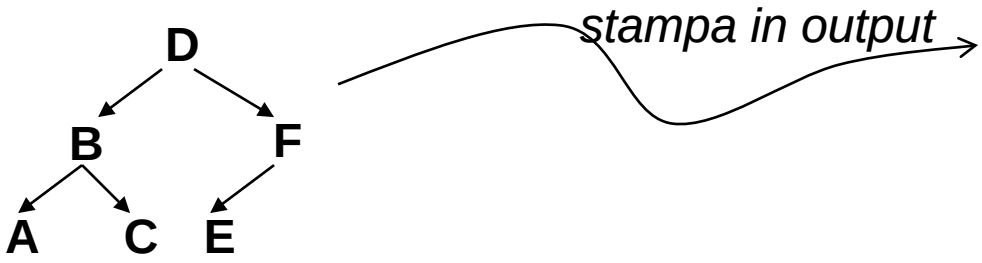



```
void stampaAlb3 (TipoAlb alb) {

                        { VUOTO        ---
                        {
                        {
                    =   { else
                        { RAD          1. stampaAlb3 SIN
                        { SIN          2. stampa info (RAD)
                        { DES          3. stampaAlb3 DES

   if (alb) {
      stampaAlb3(alb->sin);
      printf(" %c ", alb-> info;
      stampaAlb3(alb->des);
return;
}
```

***NB. APPROCCIO*: precedenza al sottoalbero sinistro ...**

- *prima* **VISITA** SOTTOALB SIN
- POI **ANALISI RADICE**
- POI **VISITA** SOTTOALB DES

VISITA di un ALBERO - visita sinistra Vs. visita destra

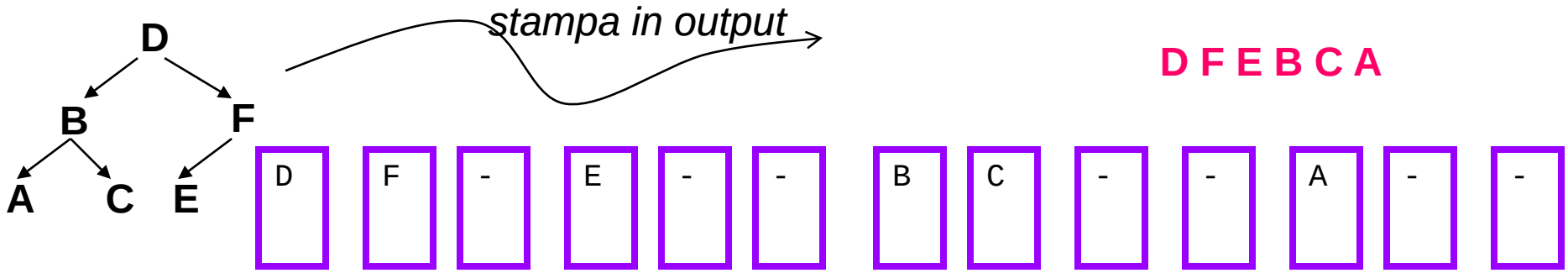



```
void stampaAlb1bis (TipoAlb alb){

                    = { VUOTO      ---
                      { else
                      { RAD        1. stampa info (RAD)
                      { SIN        2. stampaAlb1bis DES
                      { DES        3. stampaAlb1bis SIN

   if (alb) {
      printf(" %c ", alb->info;
      stampaAlb1bis(alb->des);
      stampaAlb1bis(alb->sin);
return;
}
```

NB. APPROCCIO ***precedenza alla radice e precedenza a destra ...***

- *prima* ***ANALISI RADICE***
- POI **VISITA** SOTTOALB DES
- POI **VISITA** SOTTOALB SIN

VISITA di un ALBERO - visita sinistra Vs. visita destra




```
void stampaAlb1bis (TipoAlb alb){
```

=	VUOTO	---
	else	
	RAD	1. stampa info (RAD)
	SIN	2. stampaAlb1bis DES
	DES	3. stampaAlb1bis SIN

```
   if (alb) {
      printf(" %c ", alb->info;
      stampaAlb1bis(alb->des);
      stampaAlb1bis(alb->sin);
return;
}
```

NB. APPROCCIO ***precedenza alla radice e precedenza a destra ...***

- *prima* ***ANALISI RADICE***
- POI **VISITA** SOTTOALB DES
- POI **VISITA** SOTTOALB SIN

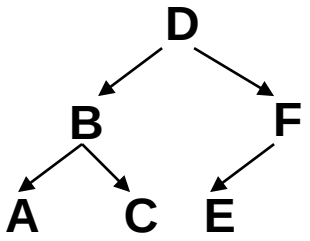
VISITA di un ALBERO – ricapitolazione main algoritmi di visita

Assumendo per default la precedenza sinistra, ecco tre algoritmi di base che realizzano la VISITA di un albero binario (le prime tre funzioni di stampa dell'albero che abbiamo visto realizzano proprio questi tre algoritmi; la quarta era una visita in "preordine destro".

IN PREORDINE

```
se ALB NON VUOTO
1. ANALISI info (RAD)
2. VISITA SIN
3. VISITA DES
```

D B A C F E



IN POSTORDINE

```
se ALB NON VUOTO
1. VISITA SIN
2. VISITA DES
3. ANALISI info (RAD)
```

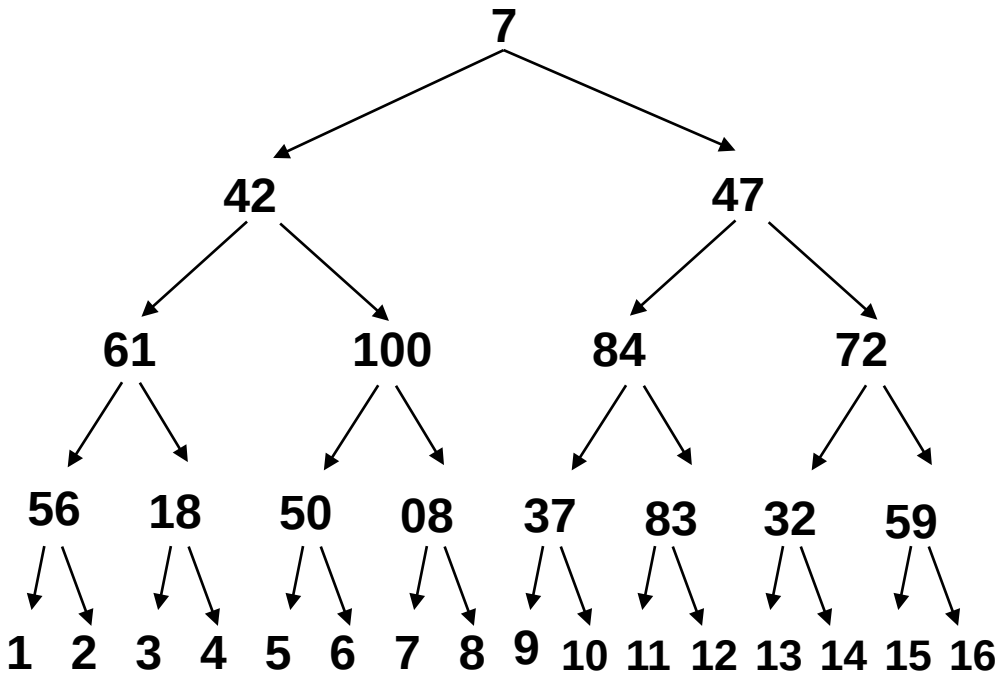
A C B E F D

SIMMETRICA

```
se ALB NON VUOTO
1. VISITA SIN
2. ANALISI info (RAD)
3. VISITA DES
```

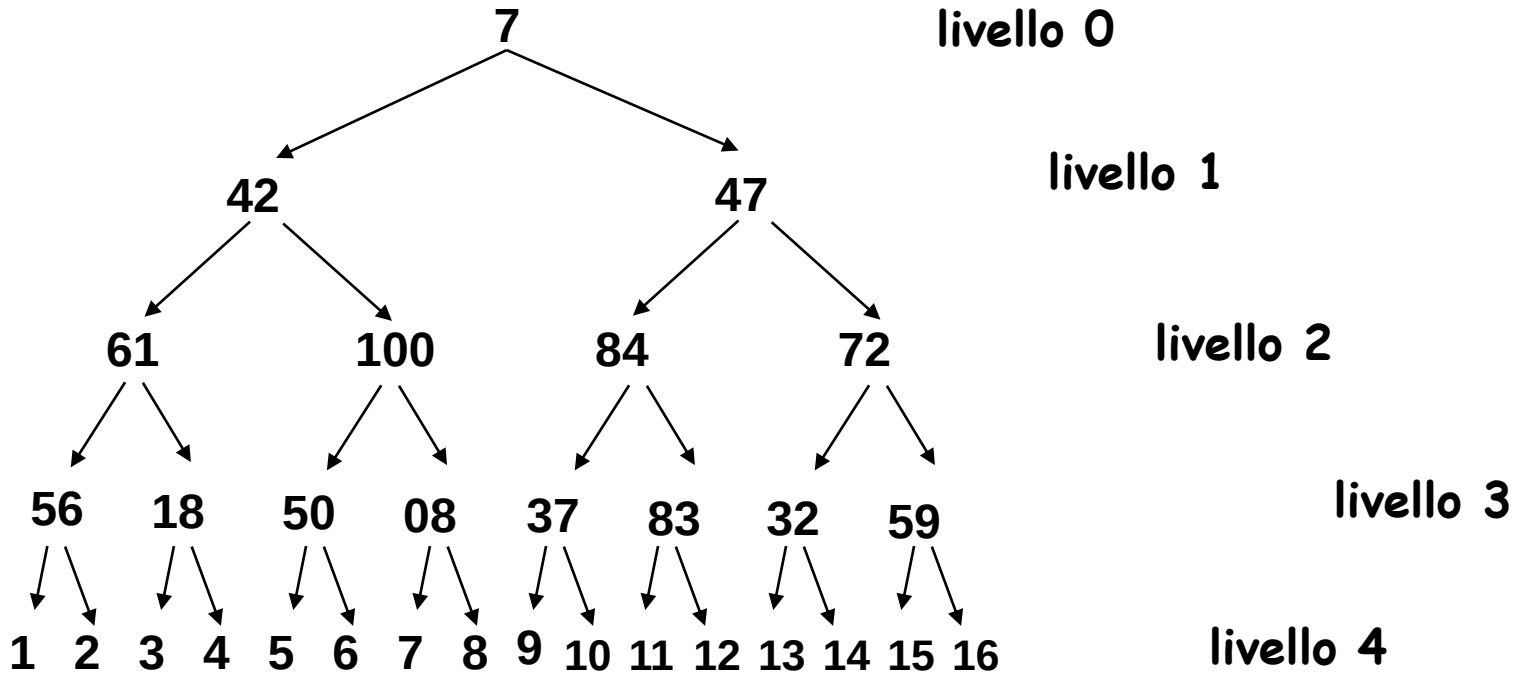
A B C D E F

albero binario completo



Ogni nodo, che non sia una foglia, ha 2 successori diretti

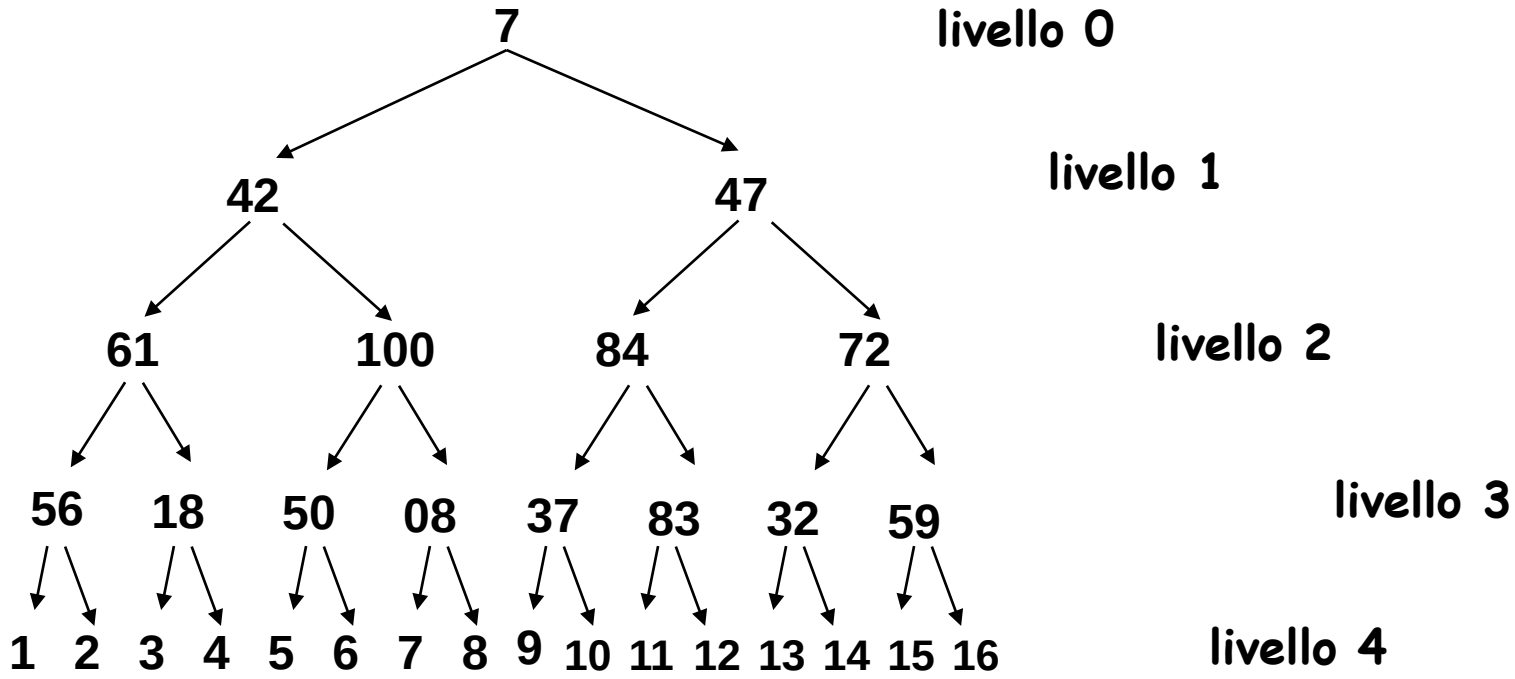
albero binario completo



Ogni nodo, che non sia una foglia, ha 2 successori diretti

- livello h: 2^h nodi
- numero nodi tot. = $2^{(prof + 1)} - 1$
- prof 4 -> 31 nodi; prof 10 -> 2047; prof 19 -> ca. 1M

albero binario complete: costo di una visita



COSTO di una VISITA per un ALBERO COMPLETO

- costo di 1 analisi = 1 coin (unico costo significativo)
- costo complessivo … costo analisi x numero di nodi
- numero nodi tot. = $2^{(prof+1)}-1$ α ($2^{(prof+1)}$)

costo proporzionale a $2^{(prof+1)}$

Ricerca esaustiva in albero binario

```
int presente (TipoInfo elem, TipoAlb alb) {
  ...
}
```

Ricerca esaustiva: si scandisce l'albero con una visita, confrontando ogni nodo con elem; se si trova un nodo uguale ad elem, si esce con 1; se si finisce di scandire l'albero senza aver trovato elem, si restituisce zero.

```
presente(ELEM, ALB))

= se ALB VUOTO                                  0
  se ALB non vuoto ci sono RADICE SIN DES
    RADICE è ELEM                               1
    RADICE <> ELEM
      se presente(ELEM,SIN)                     1
      se presente(ELEM,DES)                     1
      sennò                                     0
```

Ricerca esaustiva in albero binario

```
int presente (TipoInfo elem, TipoAlb alb) {
  ...
}
```

Ricerca esaustiva: si scandisce l'albero con una visita, confrontando ogni nodo con elem; se si trova un nodo uguale ad elem, si esce con 1; se si finisce di scandire l'albero senza aver trovato elem, si restituisce zero.

```
presente(ELEM, ALB))

= se ALB VUOTO                                  0
  sennò, ci sono RADICE
      RADICE è ELEM                             1
      RADICE <> ELEM
          se presente(ELEM,SIN)
             OR presente(ELEM,DES)              1
          senno`                                0
      SIN
      DES
```

Ricerca esaustiva in albero binario

```
int presente (TipoInfo elem, TipoAlb alb) {
  ...
}
```

Ricerca esaustiva: si scandisce l'albero con una visita, confrontando ogni nodo con elem; se si trova un nodo uguale ad elem, si esce con 1; se si finisce di scandire l'albero senza aver trovato elem, si restituisce zero.

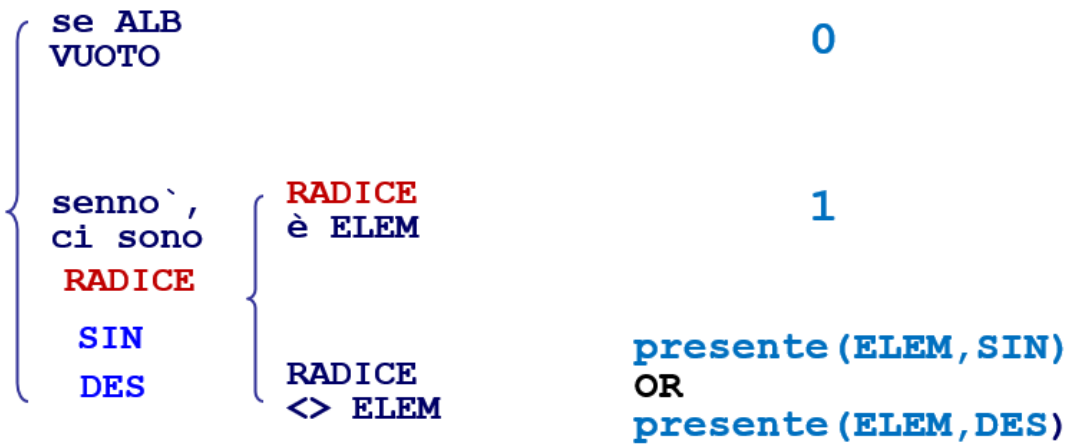
```
presente(ELEM, ALB))

= { se ALB VUOTO                                  → 0
  { senno`, ci sono RADICE
      { RADICE è ELEM                             → 1
      { SIN / DES  RADICE <> ELEM                 → presente(ELEM,SIN)
                                                    OR
                                                    presente(ELEM,DES)
```

Ricerca esaustiva in albero binario

Ricerca esaustiva

riceve elemento e albero,

restituisce
  0 (non trovato elem in albero)
  1 (trovato elem in albero)




```
int presente (TipoInfo elem, TipoAlb alb) {
 if (!alb)
    return 0;
 else
    if (uguali(elem, alb->info)
       return 1;
    else
       return (
          presente(elem, alb->sin) || presente(elem, alb->des)
       );
}
```

che visita e`?

Ricerca esaustiva in albero binario

Ricerca esaustiva

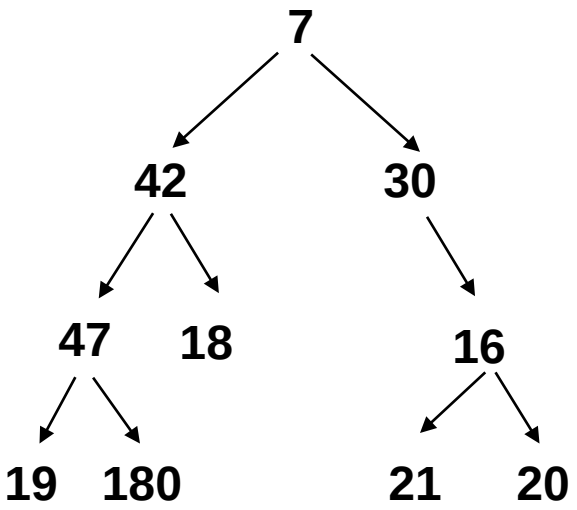
```
int presente (TipoInfo elem, TipoAlb alb) {
 if (!alb)
    return 0;
 else
    if (uguali(elem, alb->info) return 1;
    else return(presente(elem, alb->sin) || presente(elem, alb->des));
}
```

se ALB VUOTO → 0

senno`, ci sono RADICE:
- RADICE è ELEM → 1
- RADICE <> ELEM → presente(ELEM,SIN) OR presente(ELEM,DES)

quante analisi, quanti RDA per la chiamata `presente(el, albero)`

el	analisi	RDA
42		



Ricerca esaustiva in albero binario

Ricerca esaustiva

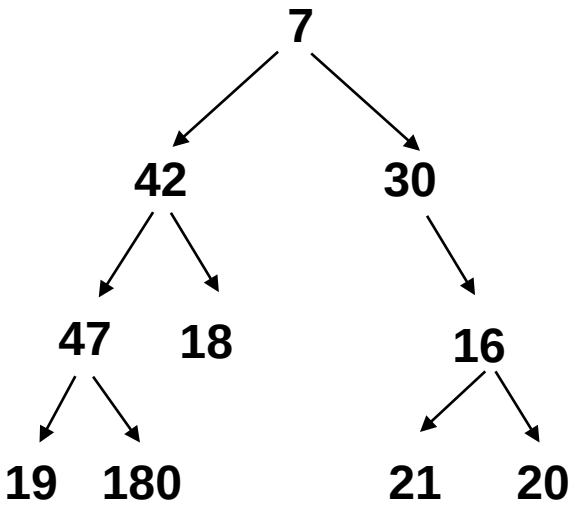
```
int presente (TipoInfo elem, TipoAlb alb) {
 if (!alb)
    return 0;
 else
    if (uguali(elem, alb->info) return 1;
    else return(presente(elem, alb->sin) || presente(elem, alb->des));
}
```

se ALB VUOTO → 0

senno`, ci sono RADICE:
- RADICE è ELEM → 1
- RADICE <> ELEM → presente(ELEM,SIN) OR presente(ELEM,DES)

quante analisi, quanti RDA per la chiamata `presente(el, albero)`

el	analisi	RDA
42	2	2
180		



Ricerca esaustiva in albero binario

Ricerca esaustiva

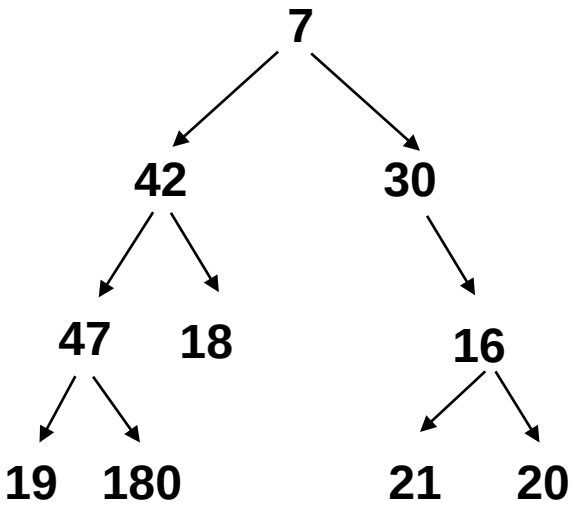
```
int presente (TipoInfo elem, TipoAlb alb) {
 if (!alb)
    return 0;
 else
    if (uguali(elem, alb->info) return 1;
    else return(presente(elem, alb->sin) || presente(elem, alb->des));
}
```

se ALB VUOTO → 0

senno', ci sono RADICE:
- RADICE è ELEM → 1
- RADICE <> ELEM → presente(ELEM,SIN) OR presente(ELEM,DES)

quante analisi, quanti RDA per la chiamata `presente(el, albero)`

el	analisi	RDA
42	2	2
180	5	7
18		



Ricerca esaustiva in albero binario

Ricerca esaustiva

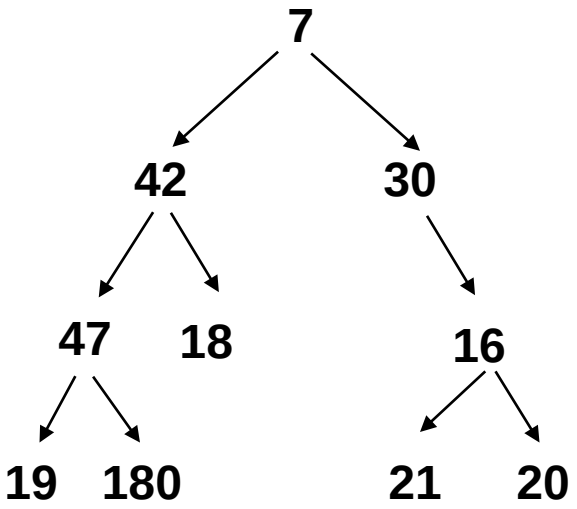
```
int presente (TipoInfo elem, TipoAlb alb) {
 if (!alb)
    return 0;
 else
    if (uguali(elem, alb->info) return 1;
    else return(presente(elem, alb->sin) || presente(elem, alb->des));
}
```

se ALB VUOTO → 0

senno`, ci sono RADICE:
- RADICE è ELEM → 1
- RADICE <> ELEM → presente(ELEM,SIN) OR presente(ELEM,DES)

quante analisi, quanti RDA per la chiamata `presente(el, albero)`

el	analisi	RDA
42	2	2
180	5	7
18	6	10
30		



Ricerca esaustiva in albero binario

Ricerca esaustiva

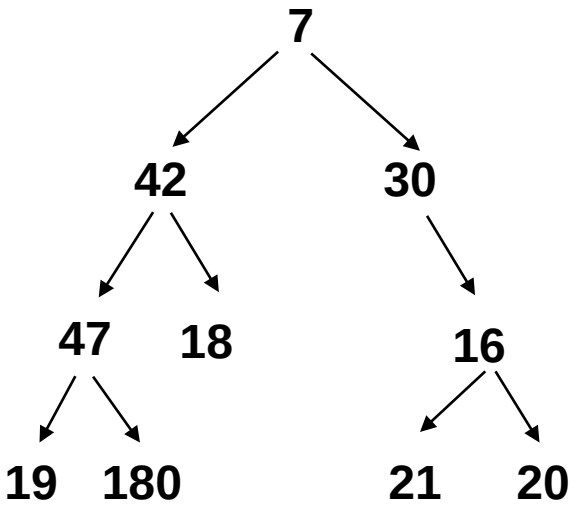
```
int presente (TipoInfo elem, TipoAlb alb) {
 if (!alb)
    return 0;
 else
    if (uguali(elem, alb->info) return 1;
    else return(presente(elem, alb->sin) || presente(elem, alb->des));
}
```

se ALB VUOTO → 0

senno`, ci sono RADICE:
- RADICE è ELEM → 1
- RADICE <> ELEM → presente(ELEM,SIN) OR presente(ELEM,DES)

quante analisi, quanti RDA per la chiamata `presente(el, albero)`

el	analisi	RDA
42	2	2
180	5	7
18	6	10
30	7	13
21		



Ricerca esaustiva in albero binario

Ricerca esaustiva

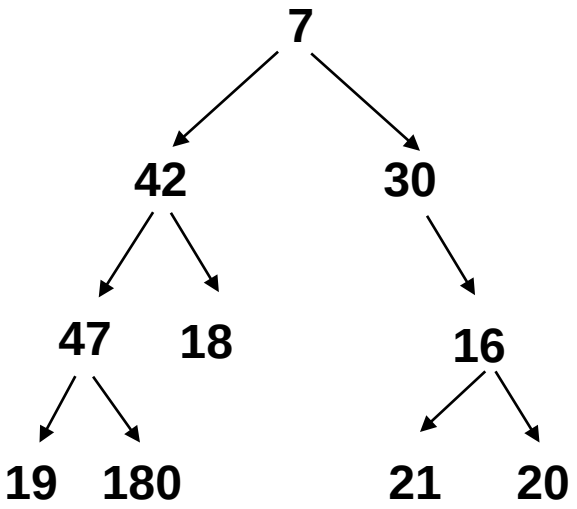
```
int presente (TipoInfo elem, TipoAlb alb) {
 if (!alb)
    return 0;
 else
    if (uguali(elem, alb->info) return 1;
    else return(presente(elem, alb->sin) || presente(elem, alb->des));
}
```

se ALB VUOTO → 0

senno`, ci sono RADICE:
- RADICE è ELEM → 1
- RADICE <> ELEM → presente(ELEM,SIN) OR presente(ELEM,DES)

quante analisi, quanti RDA per la chiamata `presente(el, albero)`

el	analisi	RDA
42	2	2
180	5	7
18	6	10
30	7	13
21	9	16
20		



Ricerca esaustiva in albero binario

Ricerca esaustiva

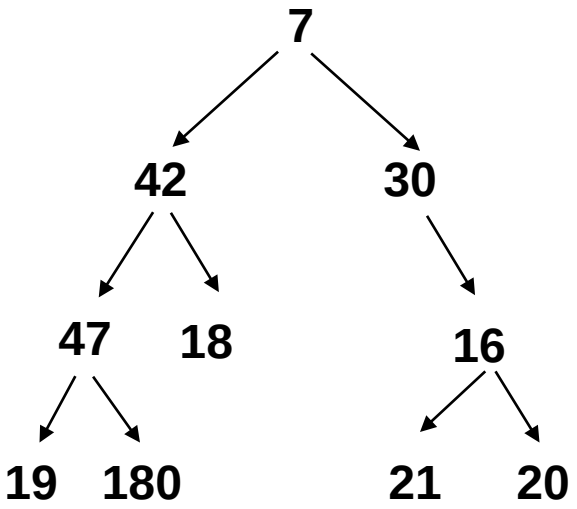
```
int presente (TipoInfo elem, TipoAlb alb) {
 if (!alb)
    return 0;
 else
    if (uguali(elem, alb->info) return 1;
    else return(presente(elem, alb->sin) || presente(elem, alb->des));
}
```

se ALB VUOTO → 0

senno`, ci sono RADICE:
- RADICE è ELEM → 1
- RADICE <> ELEM → presente(ELEM,SIN) OR presente(ELEM,DES)

quante analisi, quanti RDA per la chiamata `presente(el, albero)`

el	analisi	RDA
42	2	2
180	5	7
18	6	10
30	7	13
21	9	16
20	10	19
100		



Ricerca esaustiva in albero binario

Ricerca esaustiva

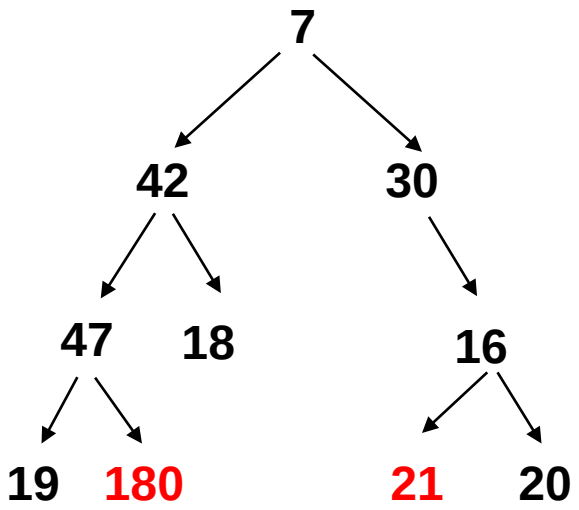
```
int presente (TipoInfo elem, TipoAlb alb) {
 if (!alb)
    return 0;
 else
    if (uguali(elem, alb->info) return 1;
    else return(presente(elem, alb->sin) || presente(elem, alb->des));
}
```

se ALB VUOTO → 0

senno`, ci sono RADICE:
- RADICE è ELEM → 1
- RADICE <> ELEM → presente(ELEM,SIN) OR presente(ELEM,DES)

quante analisi, quanti RDA per la chiamata `presente(el, albero)`

el	analisi	RDA
42	2	2
180	5	7
18	6	10
30	7	13
21	9	16
20	10	19
100	10	21



il costo della ricerca varia parecchio in base alla posizione del nodo

Ricerca esaustiva in albero binario

Ricerca esaustiva

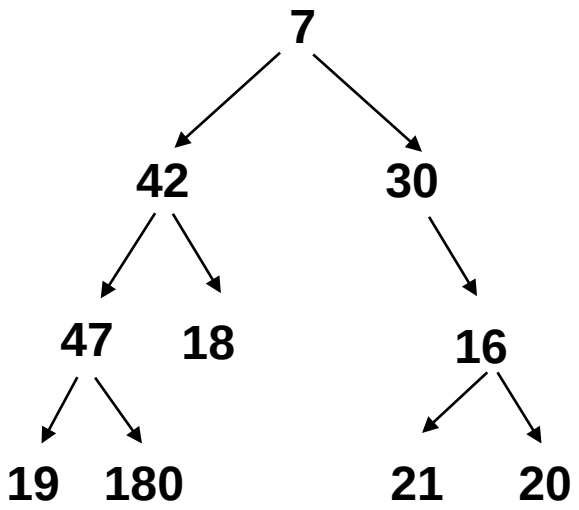
```
int presente (TipoInfo elem, TipoAlb alb) {
 if (!alb)
    return 0;
 else
    if (uguali(elem, alb->info) return 1;
    else return(presente(elem, alb->sin) || presente(elem, alb->des));
}
```

se ALB VUOTO → 0

senno`, ci sono RADICE:
- RADICE è ELEM → 1
- RADICE <> ELEM → presente(ELEM,SIN) OR presente(ELEM,DES)

quante analisi, quanti RDA per la chiamata `presente(el, albero)`

el	analisi	RDA
42	2	2
180	5	7
18	6	10
30	7	13
21	9	16
20	10	19
100	10	21



costo calcolato solo per le operazioni di analisi

α(numnodi)

[α ($2^{(prof+1)}$) per un albero completo]