```
figli (Tree P, Node v)
    e = creaLista()
    iter = v.left-child
    while iter ≠ NULL
        inserisci iter in e
        iter = iter.right_sibling
    return e
```

$T(n) = \Theta(\text{grado}(v))$

14.11.13

ALGORITMI DI VISITA DEGLI ALBERI

```
visitaGenerica (Node r)
    S = {r}
    while S ≠ ∅
        estrai un nodo u da S
        visita il nodo u
        S = S ∪ {figli di u}
```

PROTOTIPO DI ALGORITMO DI VISITA. Con "visita" si intende "esegue qualche operazione"

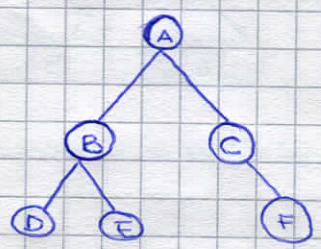
Ogni volta estraggo un nodo e prendo i figli; S rappresenta i nodi non ancora visitati

TEOREMA. L'algoritmo di visita applicato alla radice di un albero con n nodi termina in $O(n)$ iterazioni. {ovvero: lo spazio usato è $O(n)$}

DIMOSTRAZIONE.
Ipotesi: estrazioni e inserimento avvengano in tempo costante $[\Theta(1)]$

Ogni nodo verrà inserito ed estratto da S UNA SOLA VOLTA perché in un albero non si può tornare da un nodo a partire dai suoi figli procedendo di figlio in figlio.
Quindi le iterazioni del ciclo while saranno al più $O(n)$ [⇒quanti sono i nodi dell'albero]

Poiché ogni nodo compare al più una volta in S, lo spazio richiesto è proprio $O(n)$.



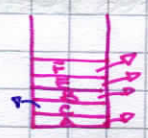
Un algoritmo che si basa sulla struttura descritta ha costo lineare.

TIPI DI VISITA

(1) DFS (Depth-First Search): Scendo nell'albero finché non ho visitato tutto il sottoalbero sinistro e poi faccio lo stesso col destro.

```
visitaGenerica(Node r)
    Stack S
    push(S, r)
    while not stack_empty(S)
        u = pop(S)
        if u ≠ NULL
            visita nodo u
            push(S, u.right)
            push(S, u.left)
```

Uso una pila. Inserisco prima il destro, poi il sinistro perché, dato che inserisco e tolgo, se facessi il contrario non riuscirei a visitare l'albero in profondità



ORDINE DI VISITA: ABDECF

VERSIONE RICORSIVA DI DFS

```
VisitaDFS(Node u)
    if u ≠ NULL
        visita il nodo u
        VisitaDFS(u.left)
        VisitaDFS(u.right)
```

TEOREMA. Se x è la radice di un sottoalbero di n nodi, la chiamata di VisitaDFS(x) richiede il tempo $\Theta(n)$

DIMOSTRAZIONE
Per dimostrare che qualcosa è $\Theta(n)$, devo dimostrare che è sia $\Omega(n)$ sia $O(n)$.

(1) Visto che DEVE visitare tutti i nodi del sottoalbero radicato in x

$$T(n) = \Omega(n) \quad \text{[limite inferiore]}$$

(2)

$$\circledast \left[T(n) = \begin{cases} c > 0 & n = 0 \\ T(k) + T(n-k-1) + d & n > 0 \end{cases} \right.$$

con d costante che rappresenta tutto quello che l'algoritmo fa eccetto la chiamata ricorsiva.

Si suppone che la funzione VisitaDFS sia chiamata per un nodo x per il quale il sottoalbero sinistro ha k nodi e il sottoalbero destro ha $(n-k-1)$ nodi.

Per dimostrare che $T(n) = O(n)$ non posso usare il teorema master: uso il metodo di sostituz.

$$T(n) \leq \underbrace{(c+d)n + c}_{\text{devo dimostrare che questa funz. limita superiormente } T(n)}$$

dimostro per induzione sul n. Caso base: $n = 0$

$$T(0) \leq (0+d)0 + c$$
$$\hookrightarrow T(0) \leq 0 + c$$
$$\hookrightarrow T(0) \leq c \qquad \text{Abbiamo detto in } \circledast \text{ che } T(0) = c$$
$$\hookrightarrow c \leq c \ ? \quad \text{sì}$$

Utilizzo l'induzione completa. Per $n > 0$

$$T(n) \leq T(k) + T(n-k-1) + d \qquad \text{Per } \circledast \text{ con } k < n$$

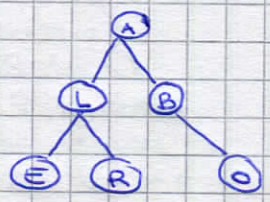
Per ipotesi induttiva

$$T(n) \leq ((c+d)k + c) + ((c+d)(n-k-1) + c) + d$$
$$\hookrightarrow (c+d)(k + n - k - 1) + 2c + d =$$
$$= (c+d)n - \cancel{d} - \cancel{d} + 2c + \cancel{d} =$$
$$= (c+d)n + c \qquad \# \underline{\text{OK}}$$

IN BASE ALL'OPERAZIONE DI VISITA DEL NODO SI OTTIENE:

(1) VISITA IN PREORDINE: si visita prima la radice poi si effettuano le chiamate ricorsive sul figlio sx o dx
(2) VISITA SIMMETRICA: si effettua prima la chiamata ricorsiva sul figlio sx, poi si visita la radice, poi si effettua la chiamata ricorsiva sul figlio destro
(3) VISITA IN POSTORDINE: si effettuano prima le chiamate ricorsive sui figli sx e dx e infine si visita la radice

ESEMPIO.



(1) Preordine: ALERBO
(2) Simmetria: ELRABO
(3) Postordine: ELRBOA

VISITA IN AMPIEZZA (BREATH-FIRST SEARCH, BFS)

```
Visita BFS(Node r)
  Queue Q
  enqueue(Q, r)
  while not queue_empty(Q)
        u = dequeue(Q)
        if u ≠ NULL
            visita ie nodo u
            enqueue(Q, u.left)
            enqueue(Q, u.right)
```

Utilizzo la coda perché mi permette di mantenere i livelli senza perderli

VISITA PER LIVELLI: ALBERO
CALCOLO DELL'ALTEZZA DI UN ALBERO

```
height(Node u)
    if u == NULL
      return -1
    else
      return 1+ max(height(u.left), height(u.right))
```

→ COMPLESSITÀ $T(n) = \begin{cases} c & n = \phi \\ T(k) + T(n-k-1) + d & n > \phi \end{cases}$
che è quella di prima (ovvero $\Theta(n)$)

GENERALIZZAZIONE

Albero generale
left_child right_sibling

```
heightgen(Node u)
    if u == NULL
        return -1
    else
        return max{1 + heightgen(u.left_child), heightgen(u.right_sibling)}
```

Stessa complessità di prima, $\Theta(n)$

↳ Fratelli allo stesso livello