Corso di Introduzione agli algoritmi
Prof.ssa Tiziana Calamoneri

Strutture dati fondamentali: Visite di alberi

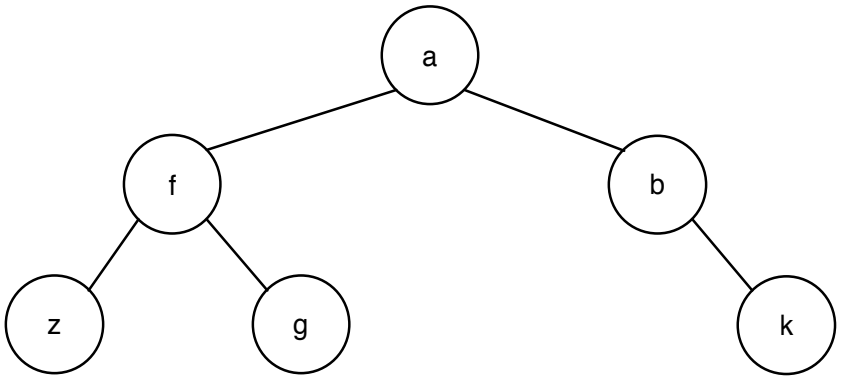
Visite di alberi binari (1)

- Un'operazione basilare sugli alberi è l'accesso a tutti i suoi nodi, uno dopo l'altro, al fine di poter effettuare una specifica operazione (che dipende ovviamente dal problema posto) su ciascun nodo.
- Tale operazione sulle liste si effettua con una semplice iterazione, ma sugli alberi la situazione è più complessa dato che la loro struttura è ben più articolata.
- L'accesso progressivo a tutti i nodi di un albero si chiama *visita dell'albero*.

Visite di alberi binari (2)

- Facendo riferimento all'ordine col quale si accede ai nodi dell'albero, è evidente che esiste più di una possibilità.
- Nel caso degli alberi binari, nei quali i figli di ogni nodo (e quindi i sottoalberi) sono al massimo due, e volendo comportarsi nello stesso modo su tutti i nodi, le possibili decisioni in merito a questa scelta danno luogo a tre diverse visite:
- *visita in preordine (preorder)*: il nodo è visitato prima di proseguire la visita nei suoi sottoalberi;
- *visita inordine (inorder)*: il nodo è visitato dopo la visita del sottoalbero sinistro e prima di quella del sottoalbero destro;
- *visita postordine (postorder)*: il nodo è visitato dopo entrambe le visite dei sottoalberi.

Visite di alberi binari (3)



visita in preordine: **a f z g b k**
visita in inordine: **z f g a b k**
visita in postordine: **z g f k b a**

Visite di alberi binari (4)

Supponiamo che l'albero sia memorizzato tramite record e puntatori e che l'albero sia dato tramite il puntatore p alla sua radice:

```
Funzione Visita_preordine (p: puntatore all'albero)
    if (p ≠ NULL)
        accesso al nodo e operazioni conseguenti
        Visita_preordine (left[p])
        Visita_preordine (right[p])
    return
```

Le altre visite sono analoghe.

L'unica differenza fra i tre casi è la posizione della pseudo-istruzione relativa all'accesso al nodo per effettuarvi le operazioni desiderate.

Costo computazionale delle visite

E', ovviamente, la stessa per tutte e tre.

Esso varia al variare della struttura dati utilizzata per memorizzare l'albero. Nel caso di memorizzazione tramite record e puntatori, detto *k* il numero di nodi del sottoalbero sinistro della radice, l'equazione di ricorrenza è:

$$T(n) = T(k) + T(n - k - 1) + \Theta(1)$$

$$T(1) = \Theta(1)$$

Non siamo in grado di risolvere questa equazione con gli strumenti dati, quindi dobbiamo valutare separatamente i diversi casi.

Costo computazionale delle visite – caso migliore

Si verifica quando l'albero è completo, poiché la suddivisione tra le due chiamate ricorsive è il più equa possibile.

In tal caso, se h è il numero dei suoi livelli, si ha $n = 2^{h+1} - 1$ ed entrambi i sottoalberi di ogni nodo sono completi. L'equazione quindi diviene:

$$T(n) = 2T(n/2) + \Theta(1)$$

che ricade nel caso 1 del teorema principale, ed ha quindi soluzione:

$$T(n) = \Theta(n^{\log 2}) = \Theta(n)$$

Costo computazionale delle visite – caso peggiore

Si verifica quando k = 0 (oppure, simmetricamente, quando *n-k-1=0*), per il quale si ottiene:

$T(n) = T(n-1) + \Theta(1)$

che ha banalmente, ad esempio col metodo iterativo, la soluzione:

$$T(n) = n\,\Theta(1) = \Theta(n)$$

Costo computazionale delle visite – caso generale

Il costo computazionale nel caso generale è limitato inferiormente da quello del caso migliore, quindi è $\Omega(n)$, e superiormente da quello del caso peggiore, quindi è $O(n)$.

Di conseguenza, anch'esso è $\Theta(n)$.

Applicazioni delle visite (1)

Le visite sono estremamente utili per ispezionare l'albero e dedurne delle proprietà. A seconda delle proprietà può essere più utile una delle tre visite considerate.

Applicazioni delle visite (2)

Esempio: conteggio del numero dei nodi.

```
Funzione Calcola_n (p: puntatore all'albero)
    if (p ≠ NULL)
        num_l ← Calcola_n (left[p])
        num_r ← Calcola_n (right[p])
        num ← num_l+num_r+1 /* accesso al nodo
    return num
```

N.B. segue la filosofia della visita in postordine!

Applicazioni delle visite (3)

Esempio: ricerca in un albero.

```
Funzione Cerca (p: puntatore all'albero, k)
    if (p ≠ NULL)
        if info[p]=k return TRUE
        else if Cerca(left[p],k)=TRUE
                return TRUE
            else return Cerca(right[p],k)
```

N.B. segue la filosofia della visita in preordine!

Si può sfruttare la compattezza del C per riscrivere la funzione:

```
Funzione Cerca_2 (p: puntatore all'albero, k)
    if (p == NULL) return FALSE
    return (info[p]==k || Cerca(left[p],k) || Cerca(right[p],k))
```

Esercizi

- Scrivere lo pseudocodice ITERATIVO della visita in preordine.
- Calcolare il costo computazionale delle visite quando l'albero venga memorizzato tramite vettore dei padri o tramite rappresentazione posizionale.

\