Algoritmi e Strutture di Dati

Visite di alberi

m.patrignani

Nota di copyright

Sommario

- Visite di un albero
  - visita in postordine (postorder traversal)
  - visita in preordine (preorder traversal)
  - visita simmetrica di alberi binari (inorder traversal)
- Esercizi sulle visite di alberi

Visite di alberi

- Un albero può essere visitato ricorsivamente con due opposte discipline
  - visita in preordine (preorder traversal)
    - dopo aver processato un nodo si procede a processare i suoi figli
    - le operazioni sui nodi vengono effettuate top-down
  - visita in postordine (postorder traversal)
    - un nodo può essere processato solo quando i suoi figli sono stati processati
    - le operazioni sui nodi vengono effettuate bottom-up
- Se l'albero è binario è possibile anche una strategia intermedia
  - visita simmetrica (inorder traversal)
    - si processa prima il figlio sinistro, poi il nodo stesso, poi il figlio destro

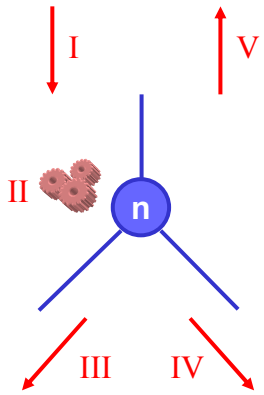
Visita in preordine

I. entro nel generico nodo *n*
- ricevo dei parametri dalla procedura eseguita sul genitore

II. eseguo la computazione su *n*
- mi avvalgo dei valori già computati sul genitore

III.e IV. lancio la procedura sul figlio sinistro e destro
- passo dei parametri alle procedure eseguite sui figli

V. esco dal nodo *n*



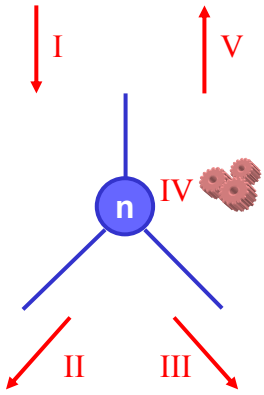
Visita in postordine

I. entro nel generico nodo *n*

II. e III: lancio la procedura sul figlio sinistro e destro
- raccolgo gli output dalle procedure lanciate sui figli

IV.eseguo la computazione su *n*
- mi avvalgo dei valori computati sui figli

V. esco dal nodo *n*
- restituisco un output alla procedura lanciata sul genitore



Visita simmetrica

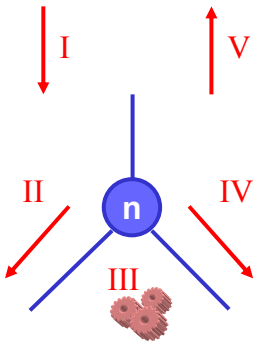
I. entro nel generico nodo *n*
   - ricevo parametri dalla procedura eseguita sul genitore

II. lancio la procedura sul figlio sinistro
   - posso passare dei parametri e ricevere un output

III. eseguo la computazione su *n*
   - posso avvalermi dei parametri passati dal genitore
   - posso avvalermi del valore computato sul solo figlio sinistro

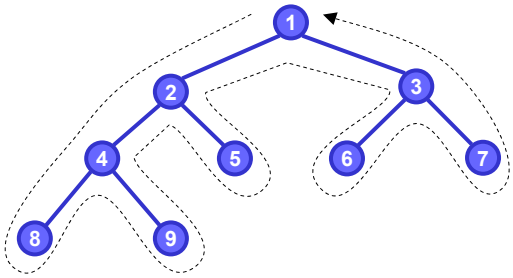
IV. lancio la procedura sul figlio destro
   - posso passare dei parametri e ricevere un output

V. esco dal nodo *n*
   - posso resitituire un output alla procedura lanciata sul genitore



Esercizi sugli alberi binari

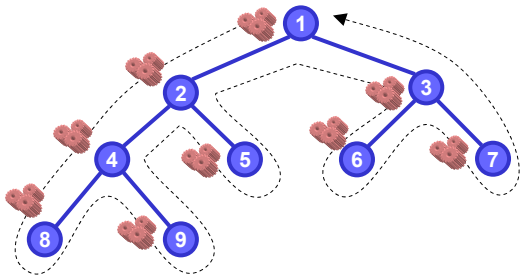
1. Scrivi la sequenza con cui i nodi vengono processati da una visita in preordine/postordine/simmetrica di questo albero binario
   - qual è la complessità asintotica delle tre visite?



   - nota: la sequenza dei nodi visitati è sempre la stessa. Ciò che cambia è il momento in cui avvengono le computazioni sul nodo

Esercizi sugli alberi binari

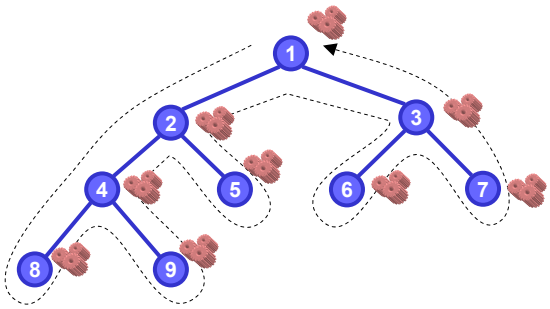
- Visita in preordine
  - appena arrivo su un nodo lo processo



  - ordine di visita: 1, 2, 4, 8, 9, 5, 3, 6, 7
  - complessità: Θ(n)

Esercizi sugli alberi binari

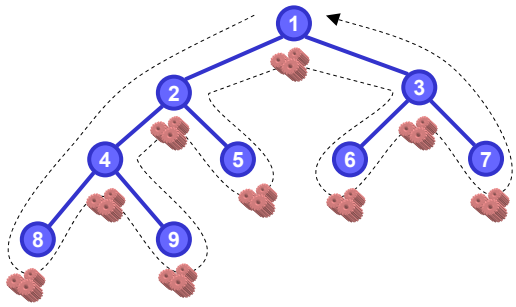
- Visita in postordine
  - processo un nodo prima di lasciarlo definitivamente



  - ordine di visita: 8, 9, 4, 5, 2, 6, 7, 3, 1
  - complessità: Θ(n)

Esercizi sugli alberi binari

- Visita in simmetrica
  - processo il nodo dopo aver processato il figlio sinistro e prima di aver processato il figlio destro



  - ordine di visita: 8, 4, 9, 2, 5, 1, 6, 3, 7
  - complessità: $\Theta(n)$

Esercizi sugli alberi binari

2. Scrivi lo pseudocodice della procedura `CERCA(t,n)` che ritorna `TRUE` se il valore `n` è presente nell'albero binario `t`
   - facendo uso di una visita in preordine
   - facendo uso di una visita in postordine
   - facendo uso di una visita simmetrica
3. Scrivi lo pseudocodice della procedura `CONTA_NODI(t)` che ritorna il numero di nodi dell'albero binario `t`
   - fai uso di una visita in postordine

Esercizi sugli alberi binari

4. Scrivi lo pseudocodice della procedura `CAMMINO(t)` che verifica se un albero binario `t` è un cammino
   - cioè se tutti i nodi hanno grado uno con l'eccezione dell'unica foglia
   - assumi che un albero vuoto sia un cammino
5. Scrivi lo pseudocodice della procedura `HEIGHT(t)` che calcola l'altezza di un albero binario `t`
   - cioè il numero di archi del cammino che va dalla radice alla foglia più profonda
   - ritorna -1 se l'albero è vuoto

CERCA(t,v) in preordine

```
CERCA(t,v)
1. return CERCA_PREORDINE(t.root,v)      ▷ innesco
```

```
CERCA_PREORDINE(n,v)
1. if n == NULL
2.     return FALSE
3. if n.info == v
4.     return TRUE
5. l = CERCA_PREORDINE(n.left)    ▷ sottoalbero sinistro
6. r = CERCA_PREORDINE(n.right)   ▷ sottoalbero destro
7. return l or r
```


CERCA(t,v) in postordine

```
CERCA(t,v)
1.  return CERCA_POSTORDINE(t.root,v)      ▷ innesco
```

```
CERCA_POSTORDINE(n,v)
1.  if n == NULL
2.      return FALSE
3.  if  CERCA_POSTORDINE(n.left)
4.      return TRUE
5.  if  CERCA_POSTORDINE(n.right)
6.      return TRUE
7.  return n.info == v
```


CERCA(t,v) con visita simmetrica

```
CERCA(t,v)
1.  return RICERCA_SIMMETRICA(t.root,v)      ▷ innesco
```

```
RICERCA_SIMMETRICA(n,v)
1.  if n == NULL
2.      return FALSE
3.  if  RICERCA_SIMMETRICA(n.left)
4.      return TRUE
5.  if  n.info == v
6.      return TRUE
7.  return RICERCA_SIMMETRICA(n.right)
```


Altri esercizi sugli alberi binari

6. Scrivi lo pseudocodice della procedura `AVERAGE(t)` che calcoli la media dei valori contenuti in un albero binario `t`
   - puoi far uso o meno di `CONTA_NODI(t)`
   - se l'albero è vuoto produci un errore
7. Scrivi lo pseudocodice della procedura `COMPLETO(t)` che verifichi se un albero binario `t` è completo
   - puoi far uso o meno della procedura `HEIGHT(t)`
   - se l'albero è vuoto ritorna `TRUE`

Altri esercizi sugli alberi binari

8. Scrivi lo pseudocodice della procedura `DEALLOCA(t)` che rimuova (deallocandoli) tutti i nodi di un albero `t`
9. Scrivi lo pseudocodice della procedura `POTA(t,x)` che elimini da un albero binario il sottoalbero radicato ad un nodo `x` specificato tramite riferimento
   - puoi omettere di deallocare i nodi potati
10. Scrivi lo pseudocodice della procedura `POTA(t,h)` che poti un albero binario lasciando solamente i nodi a profondità minore di `h`
    - puoi fare uso o meno di `POTA(t,x)`

Rappresentazioni testuali di alberi binari

11. Scrivi lo pseudocodice della procedura `PARENTETICA_SIMMETRICA`(*t*) che stampi un albero binario *t* nella rappresentazione parentetica simmetrica
    - cioè nel formato:
      "(" <sottoalbero-sx> <val-radice> <sottoalbero-dx> ")"
    - esempio: `(((()2())1(()3()))`
12. Scrivi lo pseudocodice della procedura `PARENTETICA_PREORDINE`(t) che stampi un albero binario *t* nella rappresentazione parentetica in preordine
    - cioè nel formato:
      "(" <val-radice> <sottoalbero-sx> <sottoalbero-dx> ")"
    - esempio: `(1(2()())(3()()))`

Ancora sugli alberi binari

13. Scrivi lo pseudocodice della procedura `VALORE_NONNO`(*t*) che calcoli il numero di nodi dell'albero binario *t* che hanno lo stesso valore del genitore del genitore (cioè del nonno)
14. Scrivi lo pseudocodice della procedura `DUE_FIGLI`(*t*) che calcoli il numero di nodi nell'albero binario *t* che hanno esattamente due figli
15. Scrivi lo pseudocodice della procedura `QUATTRO_NIPOTI`(*t*) che calcoli il numero di nodi dell'albero binario *t* che hanno quattro nipotini

Ancora sugli alberi binari

16. Scrivi la procedura `CAMMINO(t,n)` che ritorni una lista con gli identificatori dei nodi del cammino dalla radice fino al nodo il cui riferimento è `n`
    - puoi supporre che `n` appartenga all'albero
17. Scrivi la procedura `PARENTELA(t, n1, n2)` che calcoli il grado di parentela di due nodi con riferimenti `n1` ed `n2`
    - il grado di parentela è definito come la lunghezza del cammino che unisce i due nodi
    - puoi supporre di avere a disposizione la procedura `CAMMINO(t,n)`
      - come potresti utilizzarla?

Esercizi sugli alberi di grado arbitrario

18. Scrivi lo pseudocodice della procedura `CONTA_NODI(t)` che ritorni il numero dei nodi di un albero `t` realizzato tramite una struttura di dati "figlio-sinistro-fratello-destro"
19. Scrivi la procedura `CERCA(t,k)` che ritorni il riferimento al nodo che contiene il valore `k` in un albero `t` realizzato tramite una struttura di dati "figlio-sinistro-fratello-destro"

Esercizi sugli alberi qualsiasi

20. Scrivi la procedura `BINARIO(t)` che verifica se un albero `t` realizzato tramite una struttura di dati "figlio-sinistro-fratello-destro" sia in realtà un albero binario (in cui cioè i nodi hanno grado massimo due)
21. Scrivi la procedura `GRADO_MASSIMO(t)` che ritorni il numero massimo dei figli dei nodi di un albero `t` realizzato tramite una struttura di dati "figlio-sinistro-fratello-destro"

Esercizi sulla copia di alberi

22. Scrivi lo pseudocodice della funzione `COPIA_ALBERO(t)` che accetti in input un albero binario `t` e restituisca in output una sua copia (senza modificare l'albero `t`)
23. Scrivi lo pseudocodice della funzione analoga per alberi di grado arbitrario

Soluzioni: COPIA_ALBERO (1)

```
COPIA_ALBERO(t)
    /* t2 è un nuovo albero con il solo campo t.root */
    if ( t.root == NULL)
            t2.root = NULL
    else
            /* temp è un nuovo nodo con i campi parent, left, right
               (riferimenti) e info (intero) */
            t2.root = temp
            temp.parent = NULL
            COPIA_RIC(t.root, temp)
    return t2
```


Soluzioni: COPIA_ALBERO (2)

```
COPIA_RIC(n, n2)  /* lanciato sempre su due riferimenti non NULL */
    n2.info = n.info
    if (n.left == NULL)
            n2.left = NULL
    else
            /* temp nuovo nodo con i campi parent, left, right (rif) e info (intero) */
            n2.left = temp
            temp.parent = n2
            COPIA_RIC(n.left, temp)
    if (n.right == NULL)
            n2.right = temp
    else
            /* temp nuovo nodo con i campi parent, left, right (rif) e info (intero) */
            n2.right = temp
            temp.parent = n2
            COPIA_RIC(n.right, temp)
```

