algoritmi e strutture di dati

alberi radicati

m.patrignani

nota di copyright

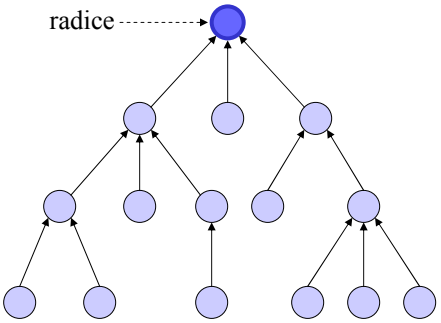
- queste slides sono protette dalle leggi sul copyright
- il titolo ed il copyright relativi alle slides (inclusi, ma non limitatamente, immagini, foto, animazioni, video, audio, musica e testo) sono di proprietà degli autori indicati sulla prima pagina
- le slides possono essere riprodotte ed utilizzate liberamente, non a fini di lucro, da università e scuole pubbliche e da istituti pubblici di ricerca
- ogni altro uso o riproduzione è vietata, se non esplicitamente autorizzata per iscritto, a priori, da parte degli autori
- gli autori non si assumono nessuna responsabilità per il contenuto delle slides, che sono comunque soggette a cambiamento
- questa nota di copyright non deve essere mai rimossa e deve essere riportata anche in casi di uso parziale

sommario

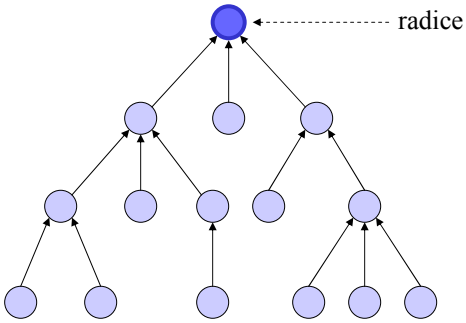
- alberi radicati
  - definizione e uso
- strutture di dati per rappresentare alberi
  - alberi binari, alberi di grado arbitrario
- visite di un albero
  - visita in postordine (postorder traversal)
  - visita in preordine (preorder traversal)
  - visita simmetrica di alberi binari (inorder traversal)
- esercizi sugli alberi

definizione di albero radicato (rooted tree)

- un *albero radicato* è un insieme di nodi, su cui è definita una relazione binaria "*x* è figlio di *y*" (oppure "*y* è genitore di *x*") tale che:
  1. ogni nodo ha un solo genitore, con l'eccezione della radice che non ha genitori
  2. c'è un cammino diretto da ogni nodo alla radice
     - l'albero, cioè, è connesso



esempio di albero radicato

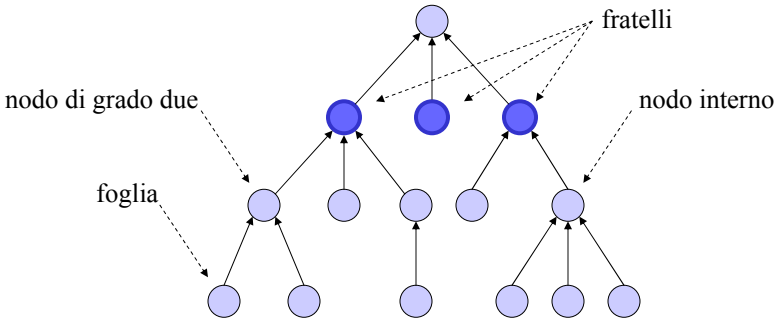


- un albero può essere costruito a partire dalla radice aggiungendo ogni volta un nodo *x* come figlio di un nodo *y* già esistente
  - ciò giustifica il fatto che, se l'albero ha *n* nodi, allora ci sono *n*-1 relazioni genitore/figlio

numerose applicazioni usano alberi

- i rapporti di ereditarietà determinano alberi
  - alberi genealogici o filogenetici
  - ereditarietà di classi nella programmazione ad oggetti
- i rapporti gerarchici sono alberi
  - gerarchie organizzative, di controllo, di responsabilità
- i rapporti di contenimento formano alberi
  - la classificazione scientifica degli organismi (tassonomie)
  - le directory del filesystem
  - i cammini minimi da una sorgente a tutti i nodi di una rete
- la struttura sintattica di una frase è un'albero
  - alberi sintattici
- …

alberi: definizioni

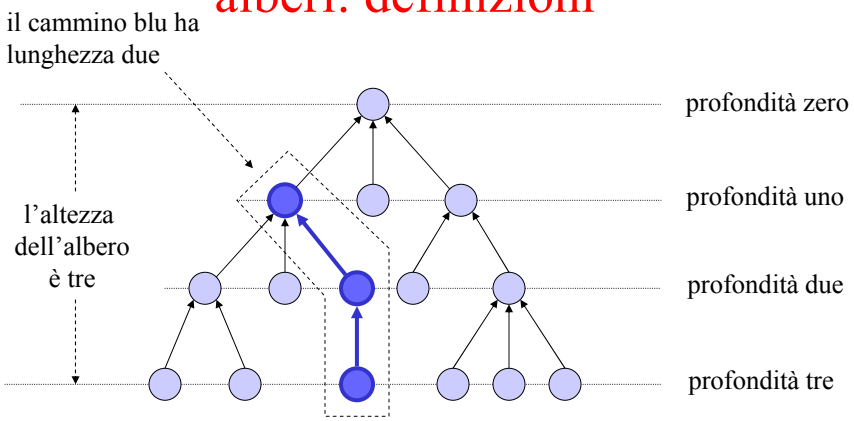


- due nodi che hanno lo stesso genitore si dicono *fratelli*
- il numero di figli di un nodo è il suo *grado*
- i nodi di grado zero sono *foglie*
- un nodo non foglia è detto *nodo interno*

tipi di alberi

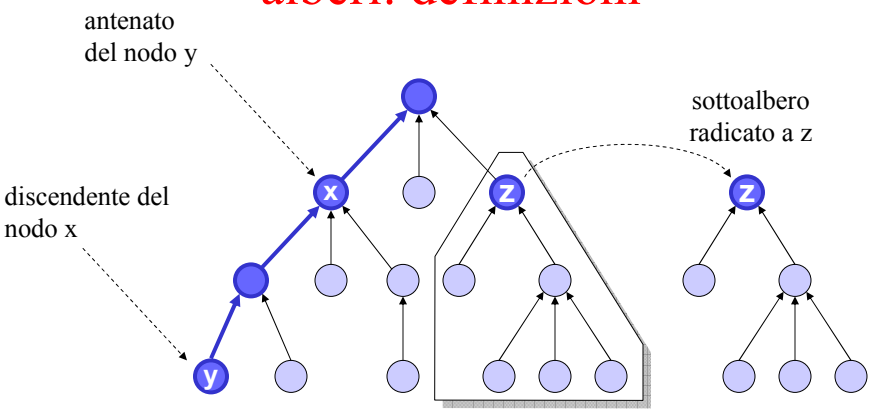
- alberi binari
  - ogni nodo può avere solamente un figlio sinistro e un figlio destro
    - l'ordine dei figli è generalmente significativo
    - si distingue tra avere il solo figlio sinistro e avere il solo figlio destro
- alberi di grado arbitrario
  - non è noto a priori il numero massimo dei figli di un nodo
    - l'ordine dei figli generalmente non è significativo

alberi: definizioni



- una sequenza di nodi tali che uno è il genitore del successivo è detta *cammino*
  - il cammino percorre gli archi alla rovescia rispetto alla figura qui sopra
- il numero degli archi di un cammino è la sua *lunghezza*
- la *profondità* di un nodo è la lunghezza del cammino dal nodo alla radice
- profondità del nodo più profondo è *l'altezza* dell'albero

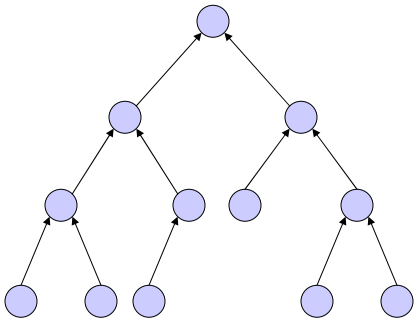
alberi: definizioni



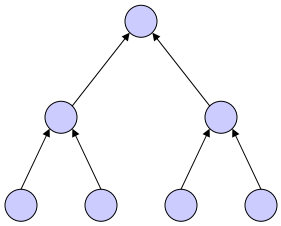
- qualunque nodo *x* sul cammino (unico) dalla *y* alla radice è un *antenato* di *y*, mentre *y* è un *discendente* di *x*;
- l'insieme costituito da un nodo *z* e da tutti i suoi discendenti è il *sottoalbero radicato a z*

alberi: definizioni

albero binario

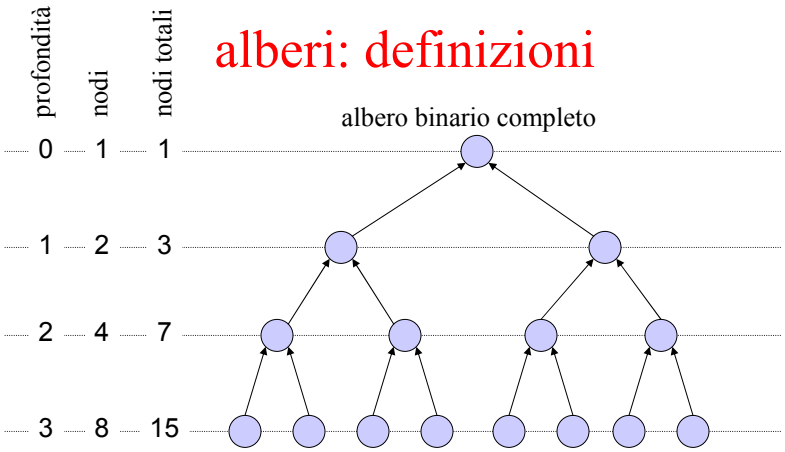


albero binario completo



- un albero *ordinato* è un albero per il quale l'ordine dei figli di ogni nodo è significativo (non possono essere permutati)
- un albero *binario* è un albero ordinato in cui i nodi hanno grado al più due
- un albero binario è *completo* se ogni livello presenta tutti i nodi possibili

alberi: definizioni



- un albero binario completo di altezza h
  - ha 2^h foglie, dunque $h = \log_2$(numero foglie)
  - ha 2^h-1 nodi interni
  - ha 2^{h+1}-1 nodi

il tipo astratto albero

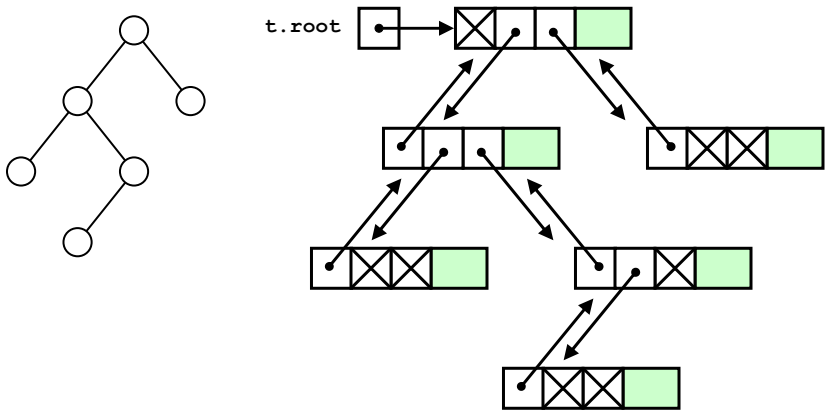
- tipo astratto albero di interi
  - domini
    - il dominio di interesse è l'insieme degli alberi di interi
    - dominio di supporto: i riferimenti R che identificano le posizioni nell'albero
    - dominio di supporto: gli interi Z = {0, 1, -1, 2, -1, …}
    - dominio di supporto: i booleani B = {true, false}
  - costanti
    - l'albero vuoto

operazioni del tipo astratto albero

- operazioni sugli alberi di interi
  - ritorna il riferimento alla radice: ROOT: $T \rightarrow R$
  - ritorna il riferimento al figlio sinistro: LEFT: $T \times R \rightarrow R$
  - ritorna il riferimento al figlio destro: RIGHT: $T \times R \rightarrow R$
  - ritorna l'intero nel nodo specificato: INFO: $T \times R \rightarrow Z$
  - verifica se un albero è vuoto: IS-EMPTY: $T \rightarrow B$
  - aggiunge un nodo come radice: ADD-ROOT: $T \times Z \rightarrow T$
  - aggiunge un nodo come figlio sinistro: ADD-LEFT: $T \times R \times Z \rightarrow T$
  - aggiunge un nodo come figlio destro: ADD-RIGHT: $T \times R \times Z \rightarrow T$
  - elimina una foglia: DELETE-LEAF: $L \times R \rightarrow L$
  - cerca un nodo: SEARCH: $T \times Z \rightarrow R$
  - svuota l'albero: EMPTY: $T \rightarrow T$
  - conta i nodi dell'albero: SIZE: $T \rightarrow Z$
  - …

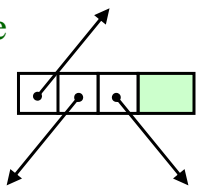
rappresentazione di alberi binari

- analogamente alle liste, gli alberi binari possono essere rappresentati mediante oggetti e riferimenti



rappresentazione di alberi binari

- un nodo dell'albero binario è un oggetto con i quattro campi
  - `parent`: riferimento al nodo genitore
  - `left`: riferimento al figlio sinistro
  - `right`: riferimento al figlio destro
  - `info`: dati satellite



- un albero binario è un oggetto con un solo campo `root` che è un riferimento al nodo radice

operazioni sugli alberi binari

- `CREATE-TREE()`
  - restituisce una struttura rappresentante l'albero vuoto
  - questa funzione rappresenta la costante
- `IS-EMPTY(t)`
  - restituisce TRUE se l'albero è vuoto
- `ROOT(t)`
  - restituisce il riferimento alla radice dell'albero (`NULL` se t è vuoto)
- `LEFT(t,n)`
  - restituisce il riferimento (può essere `NULL`) al figlio sinistro del nodo *n*
- `RIGHT(t,n)`
  - restituisce il riferimento (può essere `NULL`) al figlio destro del nodo *n*
- `INFO(t,n)`
  - restituisce le informazioni (dati satellite) memorizzate nel nodo *n*
- ...

esercizi sugli alberi binari

1. scrivi lo pseudocodice delle funzioni
   `CREATE-TREE()`
   `IS-EMPTY(t)`
   `ROOT(t)`
   `LEFT(t,n)`
   `RIGHT(t,n)`
   `INFO(t,n)`
   descritte nella slide precedente
2. scrivi lo pseudocodice della funzione `TWO_CHILDREN(n)` che ritorna `TRUE` se il nodo `n` ha due figli, `FALSE` altrimenti

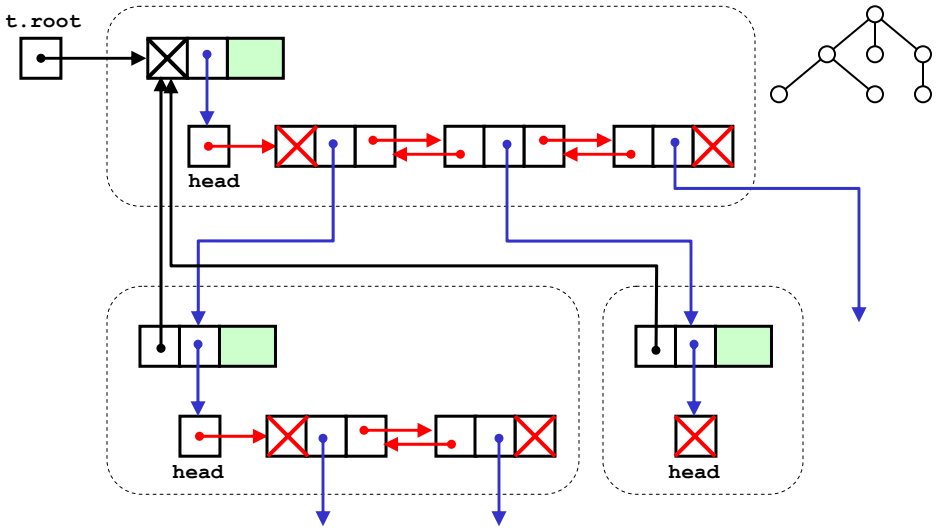
esercizi sugli alberi binari

3. scrivi lo pseudocodice della procedura `ADD-ROOT(t,z)` che aggiunga il nodo radice con valore `k` all'albero binario `t`
   - assumi che `t` sia vuoto
4. scrivi lo pseudocodice delle procedure `ADD-LEFT(t,n,z)` e `ADD-RIGHT(t,n,z)` che aggiungono il figlio sinistro e destro al nodo `n`, contenente il valore `z`
5. scrivi lo pseudocodice della funzione `ONLY LEFT(t)` che restituisce `TRUE` se tutti i nodi dell'albero binario `t` hanno solamente il figlio sinistro (o nessun figlio), `FALSE` altrimenti
   - se l'albero è vuoto restituisci `TRUE`

rappresentazione di alberi di grado arbitrario

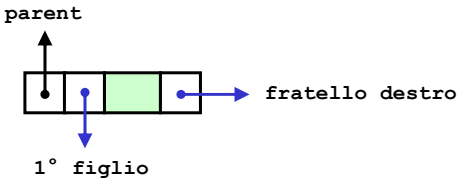
- per rappresentare alberi di grado arbitrario si possono utilizzare diverse strategie
  - uso di una lista per i figli di ogni nodo
    - poco usato perché molto prolisso
  - uso di una struttura detta "figlio-sinistro-fratello-destro"
    - più sintetico

uso di una lista per i figli di ogni nodo

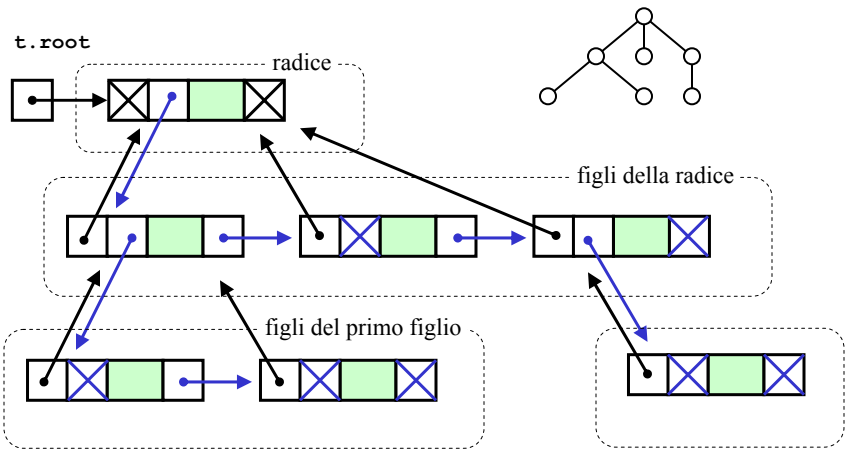


struttura "figlio-sinistro-fratello-destro"

- i nodi hanno gli usuali campi `parent`, `left`, `right` e `info`
  - i campi `parent` e `info` hanno il significato usuale
  - il campo `left` è un riferimento al figlio di sinistra (cioè al primo figlio)
  - il campo `right`, invece di essere un riferimento al figlio destro, è un riferimento al prossimo fratello



figlio-sinistro-fratello-destro



operazioni sugli alberi qualsiasi

- `CREATE-TREE()`
  - restituisce una struttura rappresentante l'albero vuoto
- `IS-EMPTY(t)`
  - restituisce `TRUE` se l'albero è vuoto
- `ROOT(t)`
  - restituisce il riferimento alla radice dell'albero (`NULL` se `t` è vuoto)
- `FIRST-CHILD(t,n)`
  - restituisce il riferimento (può essere `NULL`) al figlio sinistro del nodo `n`
- `NEXT-SIBLING(t,n)`
  - restituisce il riferimento (può essere `NULL`) al fratello destro del nodo `n`
- `INFO(t,n)`
  - restituisce l'intero memorizzato nel nodo `n`
- …

esercizi sugli alberi qualsiasi

6. scrivi lo pseudocodice della procedura `ADD-ROOT(t,z)` che aggiunga un nodo radice con valore `z` all'albero `t`
   - supponi che l'albero `t` sia vuoto
7. scrivi lo pseudocodice della procedura `ADD-SIBLING(t,n,z)` che aggiunge al nodo `n` un figlio che contiene il valore `z`

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

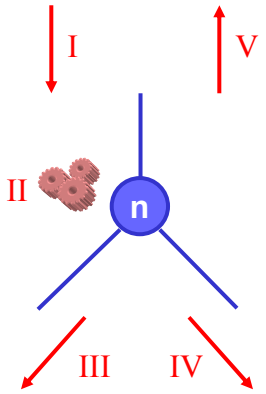
visita in preordine

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*



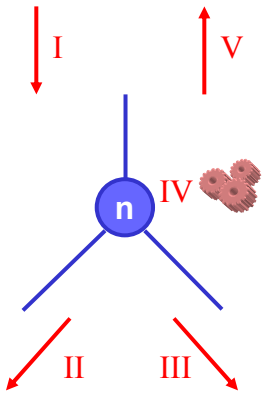
visita in postordine

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

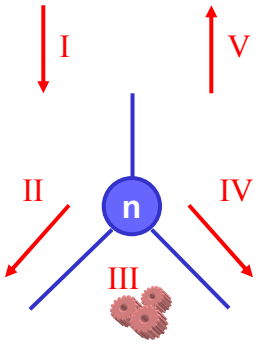
I. entro nel generico nodo *n*
   - ricevo parametri dalla procedura eseguita sul genitore

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

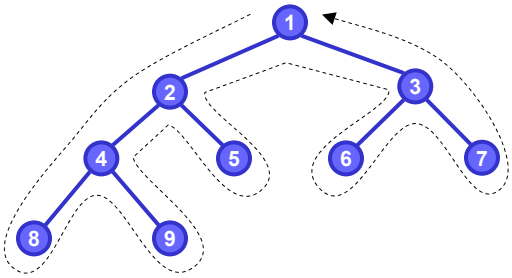
IV. lancio la procedura sul figlio destro
   - posso passare dei parametri e ricevere un output

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



esercizi sugli alberi binari

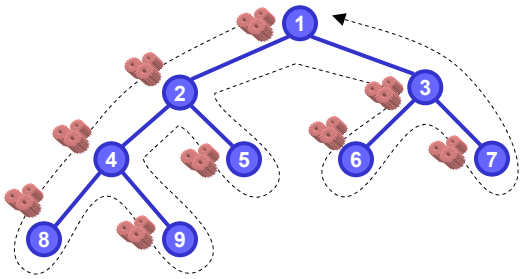
8. scrivi la sequenza con cui i nodi vengono processati da una visita in preordine/postordine/simmetrica di questo albero binario
   – qual è la complessità asintotica delle tre visite?



   – 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

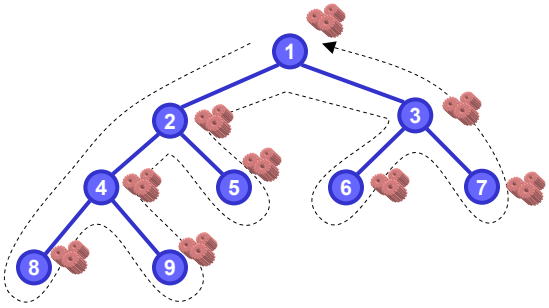
- visita in preordine
  - appena arrivo su un nodo lo processo



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

esercizi sugli alberi binari

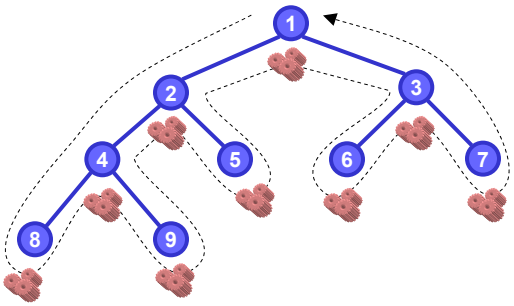
- visita in postordine
  - processo un nodo prima di lasciarlo definitivamente



  - 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

9. 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
10. 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

11. 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
12. 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

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

13. 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
14. 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

15. scrivi lo pseudocodice della procedura `DEALLOCA(t)` che rimuova (deallocandoli) tutti i nodi di un albero `t`
16. 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
17. 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

18. 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()))`
19. 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

20. 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)
21. scrivi lo pseudocodice della procedura `DUE-FIGLI`(*t*) che calcoli il numero di nodi nell'albero binario *t* che hanno esattamente due figli
22. 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

23. scrivi la procedura `CAMMINO(t,n)` che ritorni una lista con gli identificatori dei nodi del cammino dalla radice fino al nodo `n`
    - puoi supporre che `n` appartenga all'albero
24. scrivi la procedura `PARENTELA(`n_1`, `n_2`)` che calcoli il grado di parentela di due nodi n_1 ed n_2
    - 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 qualsiasi

25. 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"
26. 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

27. 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)
28. 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

29. 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`)
    - supponi per semplicità che gli indici dei nodi dell'albero siano i primi interi da zero fino al numero dei nodi-1
30. scrivi lo pseudocodice della funzione analoga per alberi di grado arbitrario

soluzioni: COPIA_ALBERO (1)

```
COPIA_ALBERO(t)
    n = CONTA_NODI(t.root)
    /* nuovonodo è un array di dimensione n, dove nuovonodo[i] è un
    riferimento al nodo del nuovo albero che rappresenta la copia del nodo
    dell'albero t con indice i */
    for i = 0 to nuovonodo.length
            /* creo un nuovo nodo nuovonodo[i] */
            nuovonodo[i].info = i
            nuovonodo[i].parent = NULL
            nuovonodo[i].left = NULL
            nuovonodo[i].right = NULL
    /* tout è un nuovo albero */
    tout.root = NULL        /* inizializzazione (non indispensabile) */
    if (t.root != NULL) tout.root = nuovonodo[t.root.info]
    COPIA(t.root, nuovonodo)
    return tout
```


soluzioni: COPIA_ALBERO (2)

```
CONTA_NODI(n)
    if (n == NULL) return 0
    return 1 + CONTA_NODI(n.right) + CONTA_NODI(n.left)

COPIA(n, nuovonodo)
    if (n == NULL) return
    if (n.parent != NULL)
            nuovonodo[n.info].parent = nuovonodo[n.parent.info]
    if (n.left != NULL)
            nuovonodo[n.info].left = nuovonodo[n.left.info]
    if (n.right != NULL)
            nuovonodo[n.info].right = nuovonodo[n.right.info]
    COPIA(n.left, nuovonodo)
    COPIA(n.right, nuovonodo)
```

