Lezione 9 – Alberi binari: visite ed esercizi

Informatica

4 Maggio 2016

Operazioni sugli alberi: visita

La *visita* di un albero esplora tutti i nodi ed esegue una qualche operazione su di essi. Per esempio, ne stampa l'etichetta:

1. stampa l'etichetta della radice e poi stampa le *etichette del sottoalbero sinistro* seguite da quelle del *sottoalbero destro*
2. la visita si implementa con una funzione *ricorsiva*

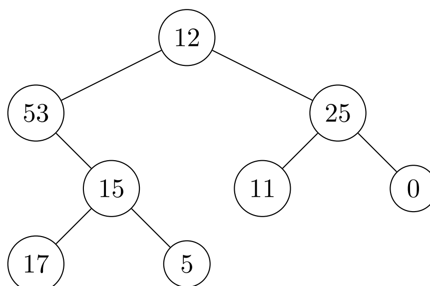
Questo tipo di visita è detto *prefissa*. Ne esistono altri due tipi:

- visita *infissa*: stampa prima il sottoalbero sinistro, poi la radice e infine il sottoalbero destro
- visita *postfissa*: stampa il sottoalbero sinistro, poi quello destro ed infine la radice

```
def prefix_visit(t):
    if not is_empty(t):
        print(label(t))
        prefix_visit(first_child(t))
        prefix_visit(second_child(t))

def infix_visit(t):
    if not is_empty(t):
        infix_visit(first_child(t))
        print(label(t))
        infix_visit(second_child(t))

def postfix_visit(t):
    if not is_empty(t):
        postfix_visit(first_child(t))
        postfix_visit(second_child(t))
        print(label(t))
```

Visita: esempi

Dato l'albero binario



qual'è il risultato delle tre visite?

```
t = bin(12,
        bin(53, tree(), bin(15, leaf(17), leaf(5))),
        bin(25, leaf(11), leaf(0)))
prefix_visit(t)
infix_visit(t)
postfix_visit(t)
```

La visita prefissa produce

```
12    53    15    17     5    25    11     0
```

La visita infissa produce

```
53    17    15     5    12    11    25     0
```

La visita postfissa produce

```
17     5    15    53    11     0    25    12
```

Visita: cercare e contare i nodi

1. Scrivere una funzione `search(n, a)` che ritorna `True` se il nodo `n` è presente nell'albero `a`, e `False` altrimenti.
2. Scrivere una funzione `size(a)` che, dato un albero binario `a`, ritorna la sua *dimensione*, ossia il numero di nodi che compongono l'albero.
3. Scrivere una funzione `print_leafs(a)` che stampa le etichette di tutte le foglie di un albero, seguendo l'ordine da sinistra a destra.

Per cercare un nodo in un albero si deve effettuare una *visita prefissa:*

- se l'albero è vuoto ritorno `False`
- se la radice è uguale al nodo `n` allora posso terminare la ricerca e ritornare `True`
- altrimenti, cerco `n` nel sottoalbero sinistro. In caso di successo posso ritornare `True`
- In caso di fallimento cerco `n` nel sottoalbero destro. In caso di successo posso ritornare `True`
- Se anche l'ultima ricerca fallisce, ritorno `False`.

```
def search(n, a):
    if is_empty(a):
        return False
    if label(a) == n:
        return True
    if search(n, first_child(a)):
        return True
    if search(n, second_child(a)):
        return True
    return False
```

oppure, in alternativa:

```
def search(n, a):
    if is_empty(a):
        return False
    if label(a) == n:
        return True
    return search(n, first_child(a)) or search(n, second_child(a))
```

La soluzione del secondo esercizio si basa sulla definizione ricorsiva della dimensione di un albero, ed effettua una *visita postfissa*:

- L'albero vuoto ha dimensione `0`
- La dimensione di un albero non vuoto si ottiene sommando le dimensioni dei sottoalberi e aggiungendo uno.

```
def size(a):
    if is_empty(a):
        return 0
    else:
        sx = size(first_child(a))
        dx = size(second_child(a))
        return sx + dx + 1
```

Per stampare le foglie seguendo l'ordine da sinistra a destra si effettua una visita prefissa che controlla se l'albero corrente è una foglia: in caso positivo la stampa, altrimenti prosegue con la visita dei sottoalberi.

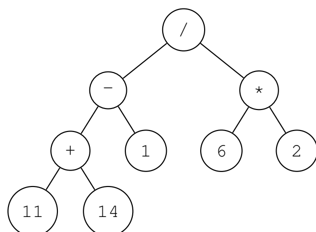
```
def is_leaf(a):
     if is_empty(a):
         return False
     return is_empty(first_child(a)) and is_empty(second_child(a))

def print_leafs(a):
     print_aux(a)
     print()

def print_aux(a):
     if not is_empty(a):
         if is_leaf(a):
             print(label(a), end=" ")
         else:
             print_aux(first_child(a))
             print_aux(second_child(a))
```

Visite: alberi ed espressioni

Con gli alberi binari si rappresentano *espressioni aritmetiche*:

- I nodi interni sono etichettati con gli operatori `+, -, *, /`
- Le foglie sono etichettate con operandi di tipo `int`

Esempio: `((11 + 14) - 1) / (6 * 2)`



- la radice è etichettata con *l'operatore principale*
- i sottoalberi rappresentano le due *sottoespressioni*

Scrivere una funzione che stampa l'espressione *in forma infissa* e una che valuta l'espressione.

```
def expr(t):
    if not is_empty(t):
        op = label(t)
        if type(op) == int:
            return op
        fc = expr(first_child(t))
        if fc == None:
            return None
        sc = expr(second_child(t))
        if sc == None:
            return None
        if op == '+':
            return fc +sc
        elif op == '-':
            return fc - sc
        elif op == '*':
            return fc * sc
        elif op == '/':
            if sc != 0:
                return fc / sc
            else:
                return None
        else:
            return None
    else:
        return None

def print_expr(t):
    print_expr_aux(t)
    print()

def print_expr_aux(t):
    if not is_empty(t):
        if is_leaf(t):
            print(label(t), end="")
        else:
            print("(", end="")
            print_expr_aux(first_child(t))
            print(label(t),end="")
            print_expr_aux(second_child(t))
            print(")", end="")

print(" --- ESPRESSIONI --- ")
e = bin('/', bin('-', bin('+', leaf(11), leaf(14)), leaf(1)),
        bin('*', leaf(6), leaf(2)))
print_expr(e)
print(expr(e))
e = bin('/', bin('-', bin('+', leaf(11), leaf(14)), leaf(1)),
        bin('/', leaf(6), leaf(0)))
```

```
print_expr(e)
print(expr(e))
e = bin('/', bin('-', bin('+', leaf(11), leaf(14)), leaf(1)),
        bin('*', leaf(6), leaf('+')))
print_expr(e)
print(expr(e))
```