Implementazione di alberi di Go

Violetta Lonati

Università degli studi di Milano
Dipartimento di Informatica

Laboratorio di algoritmi e strutture dati
Corso di laurea in Informatica

Alberi

- Un albero è una collezione non vuota di:
  - nodi con nome e informazioni contenute;
  - lati che collegano due nodi tra loro.
- Un cammino è una sequenza di nodi collegati da lati. La proprietà fondamentale degli alberi è che *esiste esattamente un cammino da un nodo ad una qualsiasi altro nodo* (altrimenti è un grafo).
- Negli alberi con radice si sceglie un nodo particolare come radice, che di solito è rappresentato in alto. Allora si usano espressioni come sopra, sotto, foglia, nodo interno, padre, figlio, antenato, discendente, ...
- Un sottoalbero è definito scegliendo un nodo interno e comprende tale nodo e tutti i suoi discendenti
- Nel caso degli alberi ordinati, i figli hanno un ordine (figlio destro, sinistro...)
- Definizione ricorsiva di albero (con radice): un albero è una foglia o una radice connessa ad un insieme di alberi.

Alberi binari

- Sono alberi (con radice) ordinati dove ogni nodo ha al più 2 figli (destro/sinistro)
- Definizione ricorsiva: un albero binario è una foglia oppure una radice connessa ad un albero binario destro e ad un albero binario sinistro.
- Proprietà numeriche:
  - un albero binario con N nodi ha $N - 1$ lati
  - un albero binario con N nodi ha altezza *circa* $log_2 N$

Rappresentazione di alberi binari in memoria

Sono strutture non monodimensionali (a differenza delle liste).
Ogni nodo può essere rappresentato come una struttura con un campo chiamato `item` (con le informazioni contenute nel nodo) e due link (al figlio destro e al figlio sinistro).

```
type treeNode struct {
  left  *treeNode
  right *treeNode
  item  int
}

type tree struct {
  root *treeNode
}
```

`item` può essere ad esempio `int` o un qualunque altro tipo, a seconda del genere di informazione contenuta nei nodi dell'albero.

Rappresentazione di alberi binari in memoria - continua

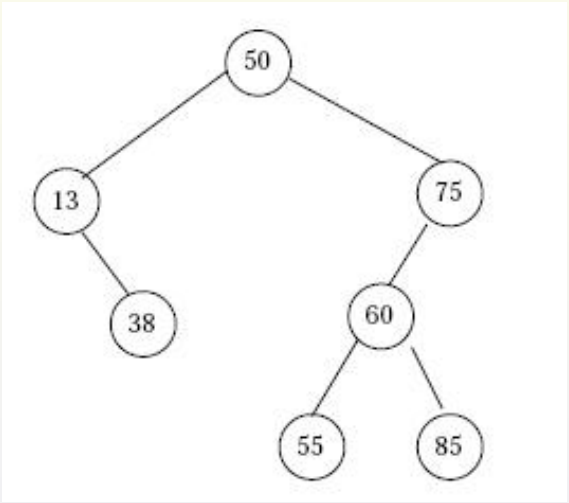
Se `n` è una variabile di tipo `treeNode` e punta ad un certo nodo, allora

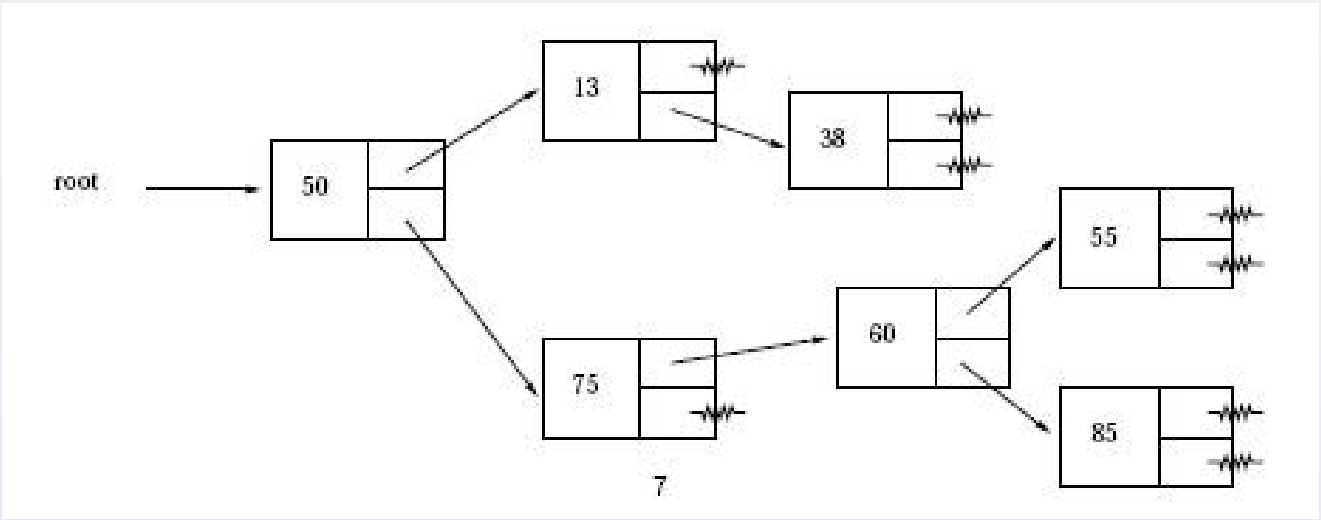
- `n.left` punta al suo figlio sinistro;
- l'assegnamento `n = n.right` fa in modo che `n` si sposti sul figlio destro.

NB: questa rappresentazione è comoda per attraversare l'albero dalla radice verso le foglie ma non viceversa: si potrebbe aggiungere un ulteriore campo `up *treeNode` (come per le liste bidirezionali, concatenate doppie)

Esempio: albero contenente interi

Nel seguente esempio, costruiamo manualmente un albero di interi, quindi `Item` è definito come `int`.





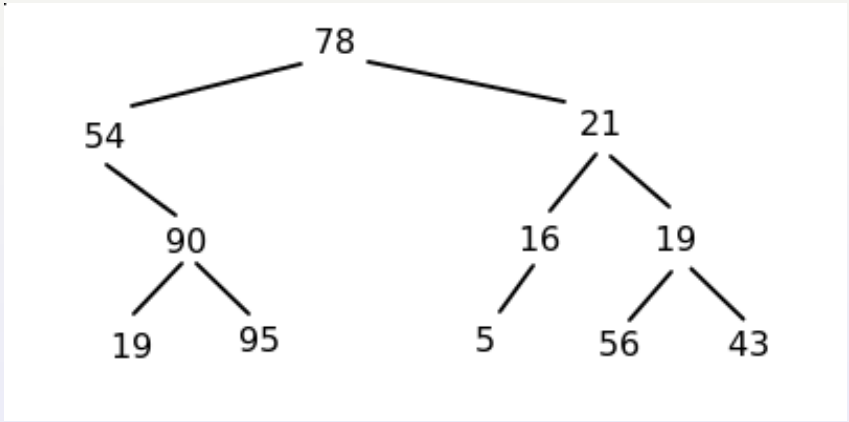
Esempi

```
func newNode(val int) *treeNode {
  return &treeNode{nil, nil, val}
}
```

```
t := &tree{nil}
t.root = &treeNode{nil, nil, 78}
t.root.left = newNode(54)
t.root.right = newNode(21)

t.root.left.right = newNode(90)
t.root.left.right.left = newNode(19)
t.root.left.right.right = newNode(95)

t.root.right.left = newNode(16)
t.root.right.left.left = newNode(5)

t.root.right.right = newNode(19)
t.root.right.right.left = newNode(56)
t.root.right.right.right = newNode(43)
```

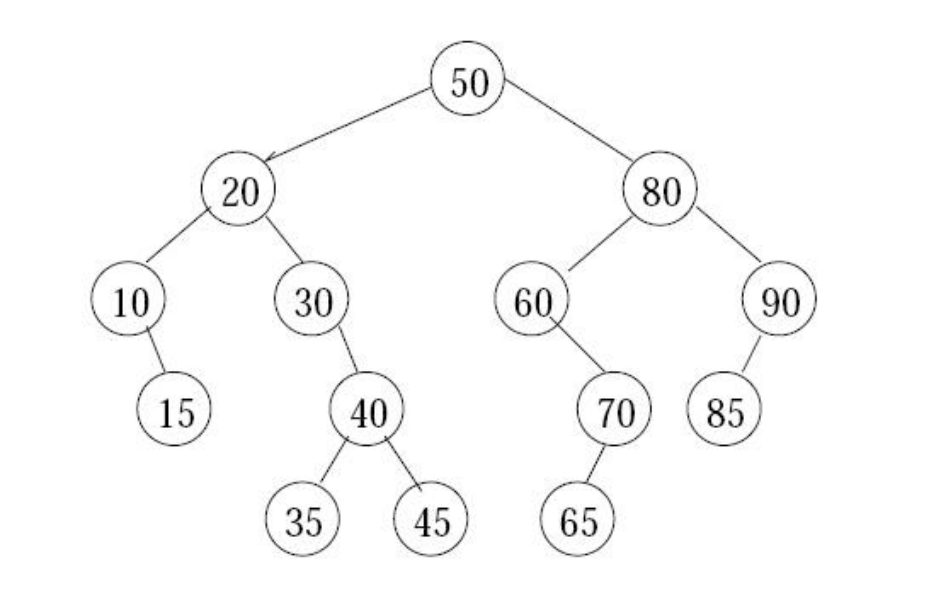
Esercizio: stampa di alberi *a sommario*

Scrivete quindi una funzione che stampi un albero binario nella rappresentazione usata nei sommari dei libri, oppure in un file browser, come nel seguente esempio:




```
*78
   *54
      *
      *90
         *19
         *95
   *21
      *16
         *5
         *
      *19
         *56
         *43
```

Attraversamento di alberi



Preorder: 50 20 10 15 30 40 35 45 80 60 70 65 90 85
Inorder: 10 15 20 30 35 40 45 50 60 65 70 80 85 90
Postorder: 15 10 35 45 40 30 20 65 70 60 85 90 80 50

Attraversamento di alberi

Attraversamento in ordine simmetrico (inorder): prima il sottoalbero di sinistra, poi la radice, infine il sottoalbero di destra:

```
func printTree(t *tree) {
  inorder(t.root)
  fmt.Println()
}

func inorder(node *treeNode) {
  if node == nil {
    return
  }
  inorder(node.left)
  fmt.Print(node.val)
  inorder(node.right)
}
```

Attraversamento di alberi

Attraversamento in ordine anticipato (preorder): prima la radice, poi il sottoalbero di sinistra, infine il sottoalbero di destra:

```
    func preorder(node *treeNode) {
    if node == nil {
      return
    }
    fmt.Print(node.val)
    preorder(node.left)
    preorder(node.right)
  }
```

Attraversamento di alberi

Attraversamento in ordine differito (postorder): prima il sottoalbero di sinistra, poi il sottoalbero di destra, infine la radice:

```
    func postorder(node *treeNode) {
    if node == nil {
      return
    }
    postorder(node.left)
    postorder(node.right)
    fmt.Print(node.val)
  }
```