Titolo nota 22/10/2015

RICEVIMENTO

OGGI POMERIGGIO 14:30

LUNEDÌ POMERIGGIO 16:00

MARTEDÌ MATTINA 11:00

UFFICIO MANCARELLA - DIP. DI INFORMATICA

ALBERI BINARI

Titolo nota 22/10/2015

```
type 'a btree = Void | Node of 'a * 'a btree * 'a btree

member : 'a -> 'a btree -> bool

let rec member x bt = match (x, bt) with
    (z, Node (z, _, _)) -> true |
    - - - -
```

non si può: le variabili usate nei pattern possono occorrere una sola volta

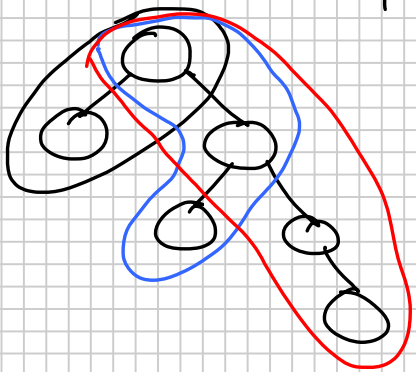
```
let rec member x bt = match bt with
      Void -> false
    | Node (z, _, _) when x = z -> true
    | Node (z, lt, rt) when x <> z -> (member x lt) or (member x rt) ;;
member : 'a -> 'a btree -> bool = <fun>
```

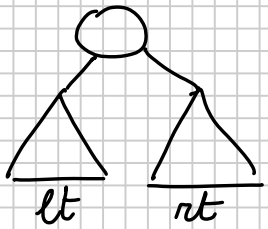
→ if (member x lt) then true else (member x rt)

PROFONDITA' di UN ALBERO BINARIO

Titolo nota 08/10/2015

è la lunghezza del cammino più lungo dalla radice ad una foglia



- l'albero vuoto ha profondità $\emptyset$
- l'albero costituito da una sola foglia ha prof. 1
- l'albero 

ha profondità

$1 + \max(\text{prof. lt}, \text{prof. rt})$

```
let rec prof bt = (*) match bt with
    Void -> ∅ |
  Node (_, lt, rt) -> 1 + max (prof lt) (prof rt)
```

```
let max x y = if x>y then x
              else y
in
```

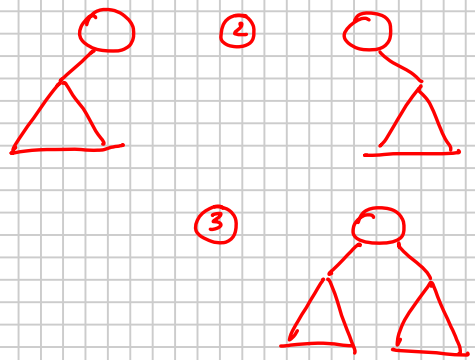
prof : 'a btree ⟶ int = <fun>

RICERCA del VALORE MASSIMO in un ALBERO (non definita se l'albero è vuoto)

Titolo nota 08/10/2015

```
let rec massimo bt = match bt with
    Node (x, Void, Void) -> x
①  | Node (x, lt, Void) when lt <> Void -> let m = massimo lt
                                           in if x > m then x else m
②  | Node (x, Void, rt) when rt <> Void -> let m = massimo rt in
                                           if x > m then x else m
③  | Node (x, lt, rt) when lt <> Void & rt <> Void ->
                let m1 = massimo lt in
                let m2 = massimo rt in
                let m = if m1 > m2 then m1 else m2 in
                    if x > m then x else m ;;
```

massimo : 'a btree -> 'a = <fun>

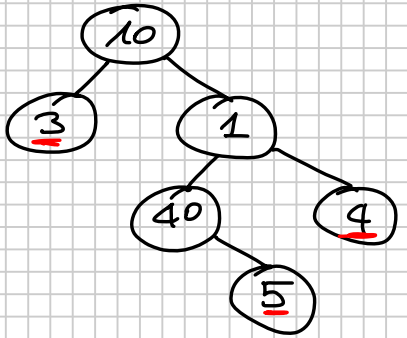


VALORE MASSIMO NELLE FOGLIE di UN ALBERO BINARIO

Titolo nota 08/10/2015

```
let rec maxfoglia bt = match bt with
    Node (x, Void, Void) ->  x
  | Node (_, lt, Void) when lt <> Void -> maxfoglia lt
  | Node (_, Void, rt) when rt <> Void -> maxfoglia rt
  | Node (_, lt, rt) when lt <> Void & rt <> Void ->
                          let m1 = maxfoglia lt in
                          let m2 = maxfoglia rt in
                            if m1>m2 then m1 else m2 ;;

maxfoglia : 'a btree -> 'a = <fun>
```



5

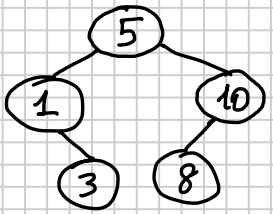
ALBERI BINARI di RICERCA

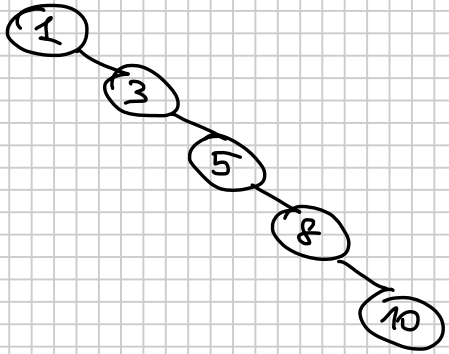
Titolo nota 08/10/2015

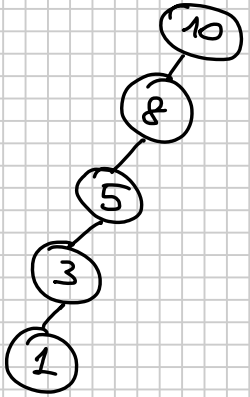
Sono alberi binari che soddisfano le seguenti proprietà

- tutti i valori che compaiono nei nodi del sottoalbero destro sono MAGGIORI del valore che compare nella radice
- tutti i valori che compaiono nei nodi del sottoalbero sinistro sono MINORI del valore che compare nella radice
- il sottoalbero sinistro e il sottoalbero destro sono alberi binari di ricerca

[3; 10; 5; 8; 1]







RICERCA di UN VALORE in UN ALBERO BINARIO di RICERCA

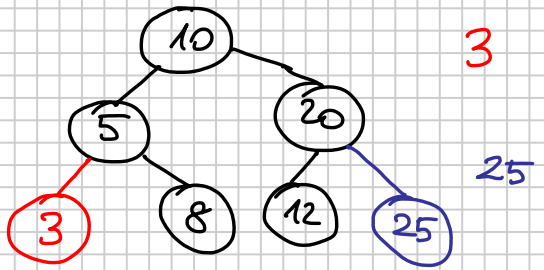
Titolo nota 08/10/2015

```
let rec memberabr x sbt = match sbt with
        Void      ->    false
  | Node (z, _, _) when x = z -> true
  | Node (z, lt, rt) when x > z -> memberabr x rt
  | Node (z, lt, rt) when x < z -> memberabr x lt ;;
```

COSTRUZIONE di UN ALBERO BINARIO di RICERCA

Titolo nota 08/10/2015

INSERIMENTO di UN VALORE in UN ALBERO BINARIO di RICERCA




```
let rec ins x sbt = match sbt with
    Void -> Node (x, Void, Void)
  | Node (z, lt, rt) when x <= z -> Node (z, ins x lt, rt)
  | Node (z, lt, rt) when x > z  -> Node (z, lt, ins x rt)
```

ins : 'b -> 'a btree -> 'b btree = <fun>

ins : 'a -> 'a btree -> 'a btree = <fun>

Titolo nota 08/10/2015

ins 2 Node (10, Node (5, Void, Void), Node (20, Void, Void))

lt rt

= { 2° pattern di ins, z = 10, 2 <= 10 }

Node (10, ins 2 Node (5, Void, Void), Node (20, Void, Void))

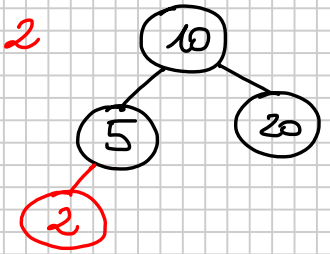
lt rt

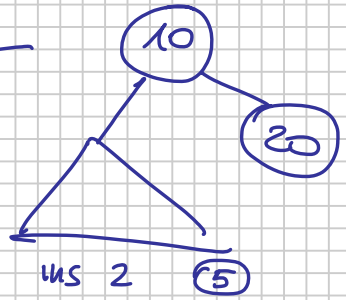
= { 2° pattern, z = 5, 2 <= 5 }

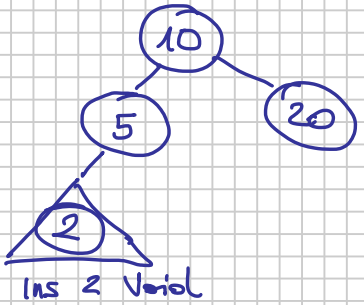
Node (10, Node (5, ins 2 Void, Void), Node (20, Void, Void))

= { 1° pattern }

Node (10, Node (5, Node (2, Void, Void), Void), Node (20, Void, Void))







```
let rec buildsbt l = match l with
    []      ->  Void
  | x :: xs ->  ins  (buildsbt xs) ;;
```

ins: 'a → 'a btree → 'a btree

buildsbt : 'b list → 'b btree = <fun>

Titolo nota 08/10/2015

buildsbt [10; 20; 3]

= { 2° pattern, x = 10, xs = [20; 3] }
ins 10 (buildsbt [20;3])

= { 2° pattern, x = 20, xs = [3] }
ins 10 (ins 20 (buildsbt [3]))

= { 2° pattern, x = 3, xs = [] }
ins 10 (ins 20 (ins 3 (buildsbt [])))

= { 1° pattern }
ins 10 (ins 20 (ins 3 Void))

= ins 10 (ins 20 Node (3, Void, Void))

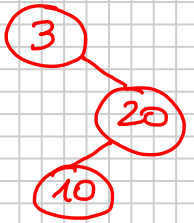
= ins 10 (Node (3, Void, ins 20 Void))

= ins 10 (Node (3, Void, Node (20, Void, Void)))

= Node (3, Void, ins 10 Node (20, Void, Void))

= Node (3, Void, Node (20, ins 10 Void, Void))

= Node (3, Void, Node (20, Node (10, Void, Void), Void))



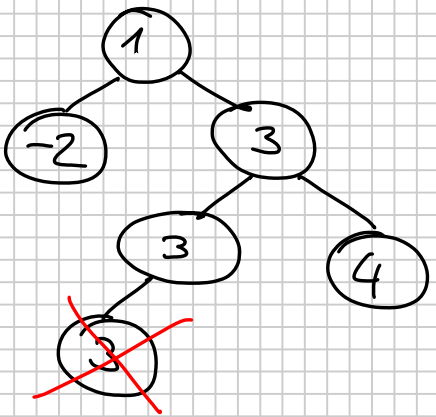
CANCELLAZIONE di UN NODO da UN ALBERO BINARIO

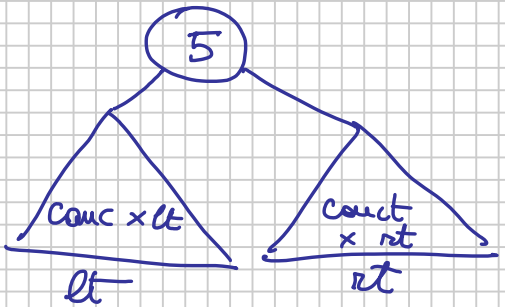
Titolo nota 08/10/2015

Cancelliamo tutte le foglie che contengono un valore dato

conc 3

```
let rec conc x bt = match bt with
    Void  ->  Void
| Node (z, Void, Void)  when x = z  ->  Void
| Node (z, Void, Void)  when x <> z  ->  Node (z, Void, Void)
| Node (z, lt, rt)  when lt <> Void or rt <> Void
        ->  Node (z, conc x lt, conc x rt)
```

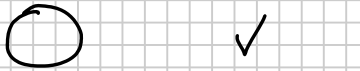




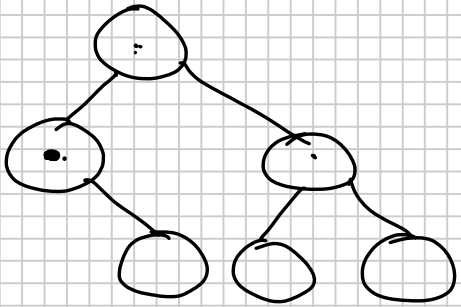
ALBERI PERFETTAMENTE BILANCIATI

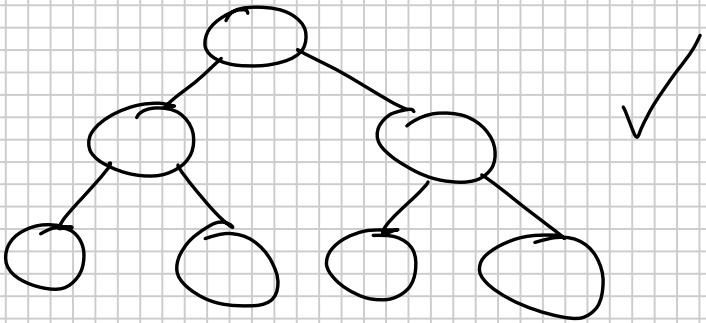
Titolo nota 08/10/2015

È un albero in cui la profondità del sottoalbero sinistro è uguale alla profondità del sottoalbero destro, e ogni sottoalbero è perfettamente bilanciato

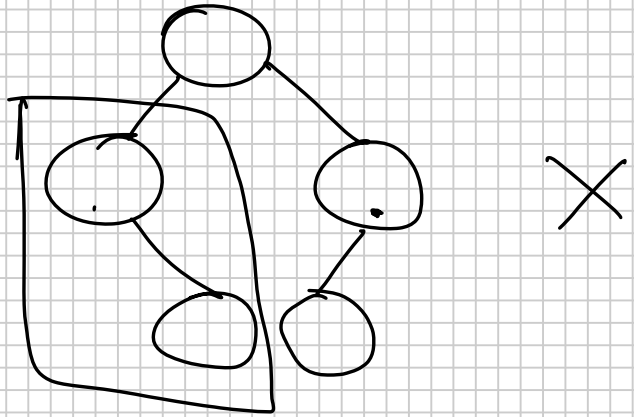


✓





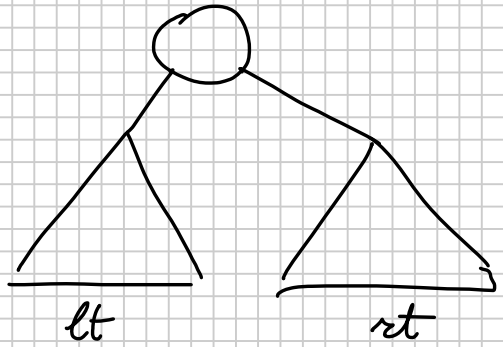
✓

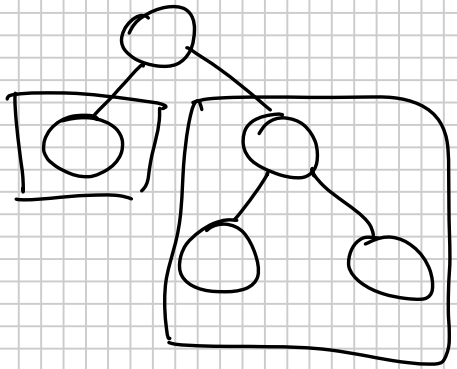


✗

Titolo nota 08/10/2015

```
let rec pbil bt =
  [ let rec pp bt =
      restituisce una coppia
        (b, n)  dove  b ∈ bool
                      n ∈ int  è la profondità di bt
  in
    let (b, n) = pp bt  in  b
```





```
let rec pp bt = match bt with
    Void  ->  (true, ∅)
| Node(_, lt, rt) -> let (b1, p1) = pp lt in
                     let (b2, p2) = pp rt in
                     if b1 & b2 then if p1 = p2 then (true, 1+p1)
                                                else (false, ∅)
                                else
                                     (false, ∅)
```