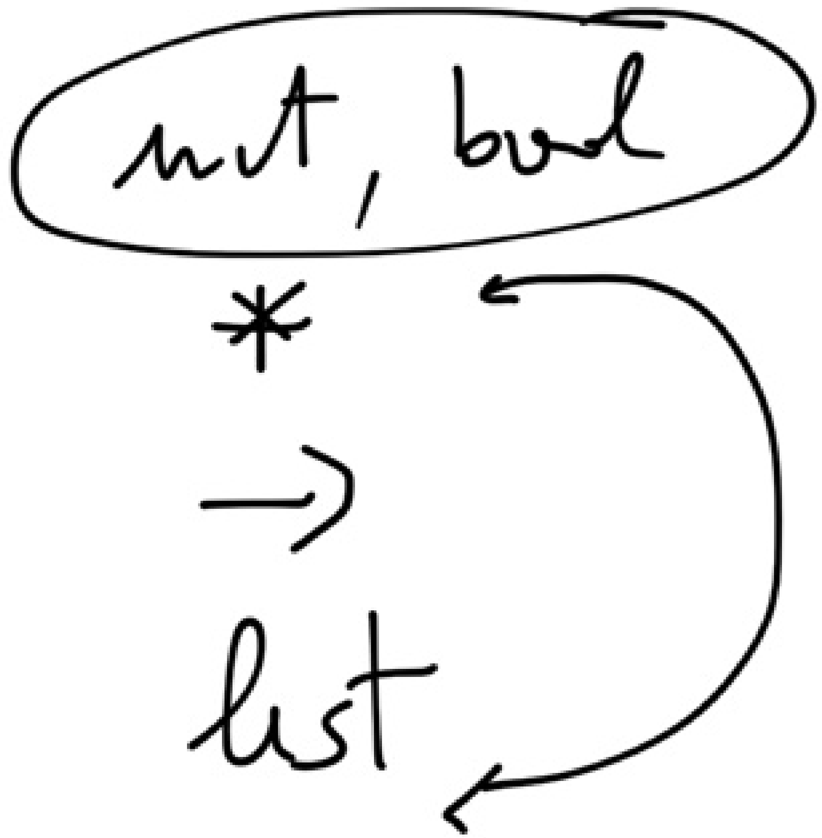
Definizioni di tipi in CAML



BAMBOO PAPER

BAMBOO PAPER

Def. di tipi <u>enumerat.</u>

in cui si enumerano i valori (nomi) dei nuovi tipi

<u>type</u> giorno = Lun | Mar | Mer | ... | Dom;

def. di tipo

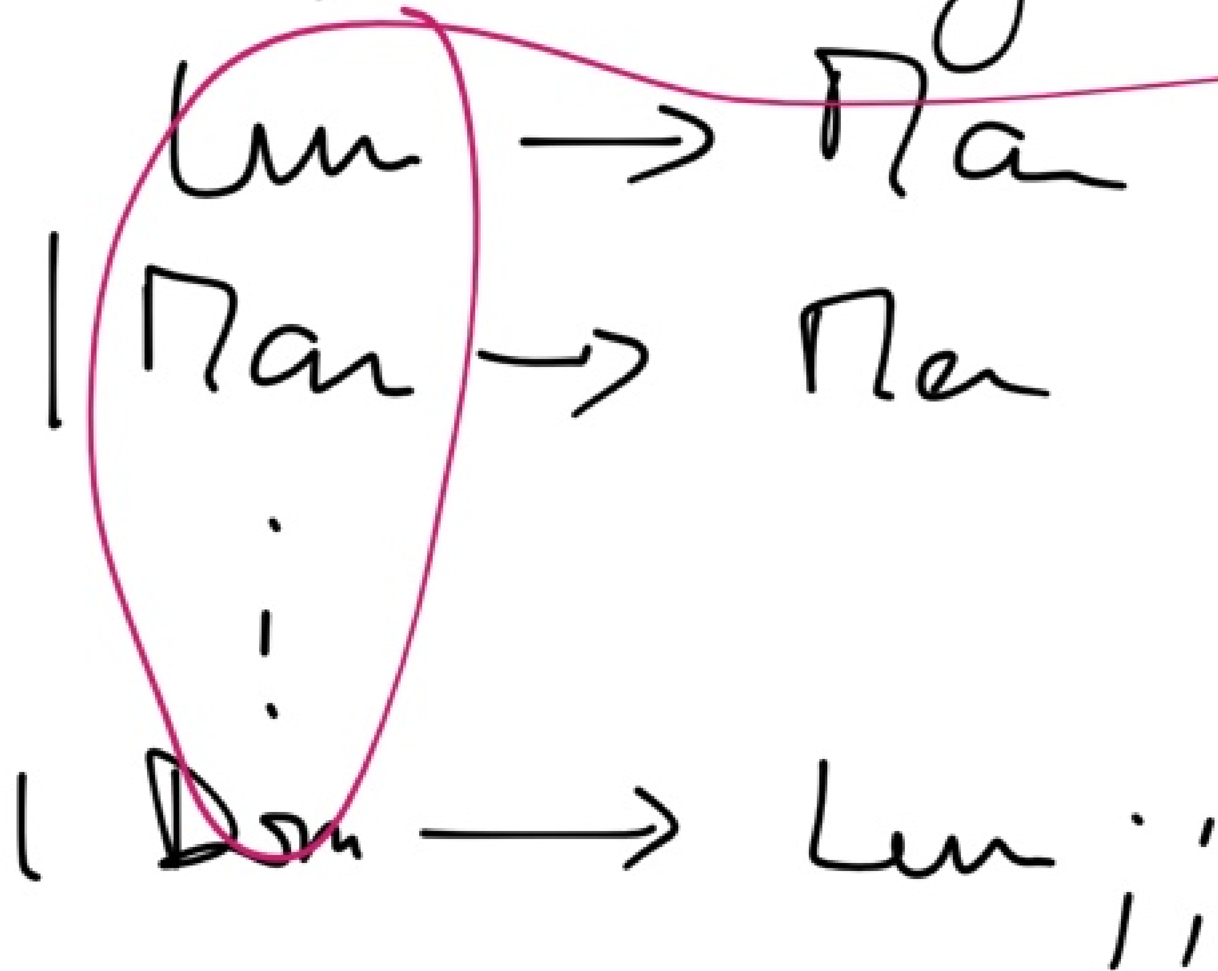
valori enumerati

```
# Lun ;;
- : giorno = Lun

# Lun < Gio ;;
- : bool = true
```

BAMBOO PAPER

14/nov/2013



```
let giornosucc g = match g with
  Lun -> Mar
| Mar -> Mer
  :
| Dom -> Lun ;;

giornosucc : giorno -> giorno
    = <fun>
```

BAMBOO PAPER

2013_11_14

14/nov/2013

```
let feriale g =
      g < Sab ;;

feriale : giorno -> bool = <fun>
```

Tipi definiti sulle base di altri

2013_11_14

BAMBOO PAPER

~~type t = int | bool ;;~~

~~5 ha tipo t~~

~~true ha tipo t~~

type t = Tn of int | Tb of bool ;;

Tn 5 : t | Tb true : t

BAMBOO PAPER

```
type t = Tn of int | Tb of bool;;
type t defined

# Tn 5;;
- : t = Tn 5

# Tn;;
- : int -> t = <fun>
# Tb;;
- : bool -> t = <fun>
```

BAMBOO PAPER

tipi polimorfi

```
type 'a pt = Pt of 'a;;

# Pt 5;;
- : int pt = Pt 5

# Pt;;
- : 'a -> 'a pt
```


BAMBOO PAPER

```
# Pt (3, true);;
- : int * bool pt = Pt (3, true)

# Pt [3; 4];;
- : int list pt = Pt [3; 4]

type ('a, 'b) ptt = Ptt
                of ('a, 'b);;
```

BAMBOO PAPER

```
Ptt (3, true);;
- : (int, bool) ptt = ...

Pt (3, true);;
- : int * bool pt
```

BAMBOO PAPER

tipi ricorsivi

```
# type nat = Zero |
             Succ of nat ;;

# Zero ;;
- : nat = Zero

# Succ ;;
- : nat -> nat ;;
```

BAMBOO PAPER

14/nov/2013

Succ(Succ Zero);;
- : nat = Succ (Succ Zero)

Succ(Succ(Succ(Succ Zero)))
- : nat = . - - -

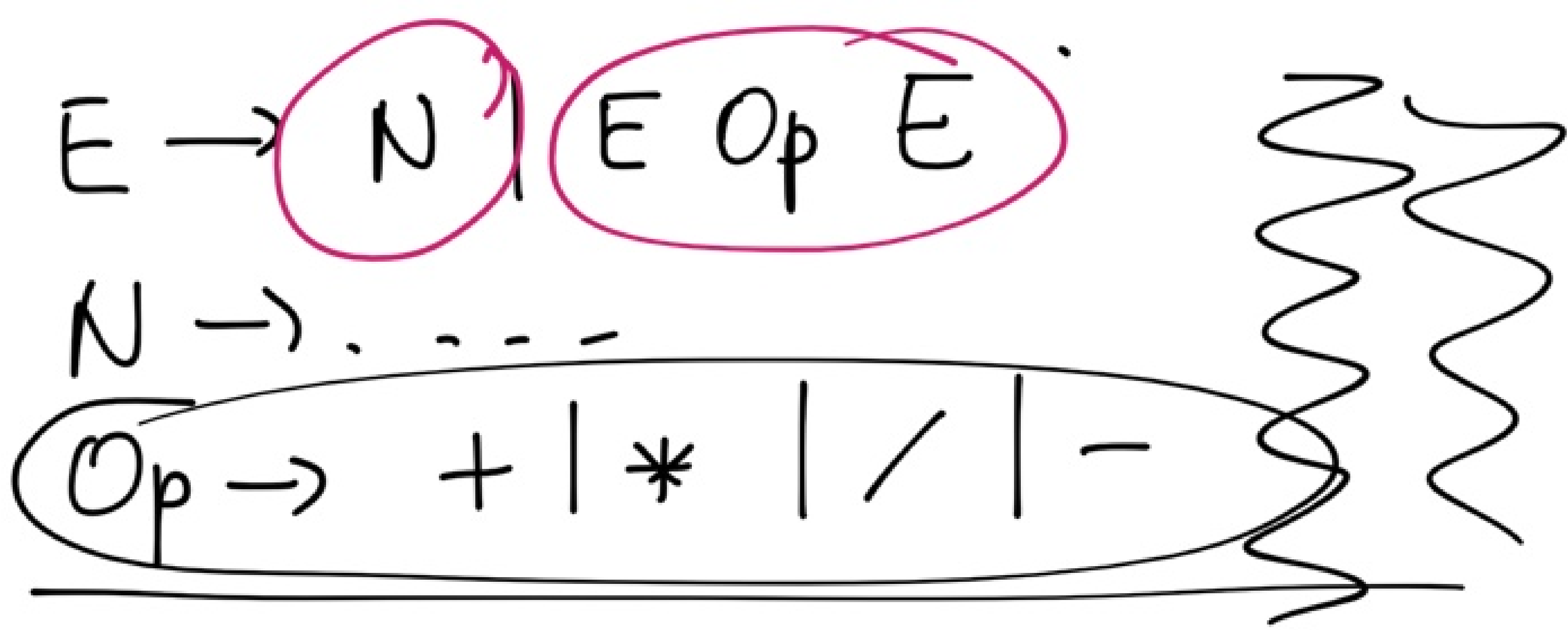
BAMBOO PAPER

2013_11_14

BAMBOO PAPER

```
let rec nat2int n =
    match n with
    Zero -> 0
  | Succ m ->
      1 + nat2int m ;;

nat2int : nat -> int = <fun>
```



$5 * 45 + 23$

In CAML vogliamo rappresentare il linguaggio delle espressioni e vogliamo calcolarne il valore.

BAMBOO PAPER

```
type op = Mul | Add | Div | Sub ;;
type aexp = Num of int |
    Exp of aexp * op * aexp ;;
```

```
Mul ;;
- : op = Mul

Num 5 ;;
- : aexp = Num 5
```

```
Num ;;
- : int -> aexp
    = <fun>

Exp ;;
- : aexp * op * aexp
    -> aexp = <fun>
```

BAMBOO PAPER

BAMBOO PAPER

3 + 5 * 6

Exp (Exp (Num 3, Sum, Num 5), Mul, Num 6)

```
let rec eval e = match e with
    Num n -> n
  | Exp (e1, o, e2) ->
      (evalop o) (eval e1)
                 (eval e2) ;;
eval : aexp -> int = <fun>
```

BAMBOO PAPER

```
let evalop o = match o with
    Mul -> (prefix *)
  | Add -> (prefix +)
  | Div -> (prefix /)
  | Sub -> (prefix -) ;;

evalop : op -> int -> int -> int
       = <fun>
```

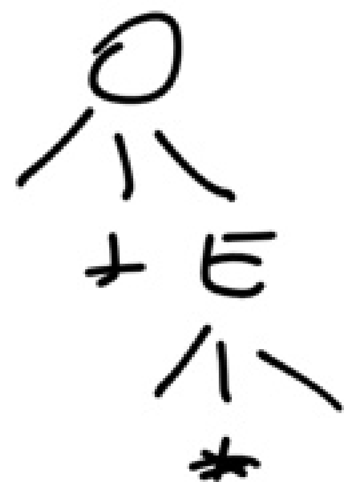
BAMBOO PAPER

3+5*6

ML

```
eval(Exp(Exp(Num 3, Add, Num 5),
          Mul, Num 6));;
- : int = 48
```

```
eval(Exp(Num 3, Add,
     Exp(Num 5, Mul, Num 6)));;
- : int = 33
```



BAMBOO PAPER

14/nov/2013

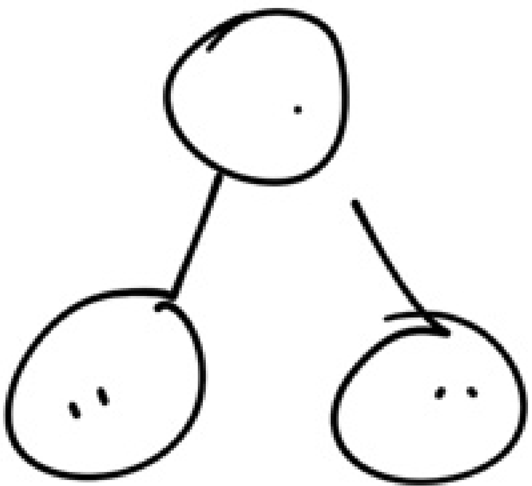
tipi polimorfi ricorsivi

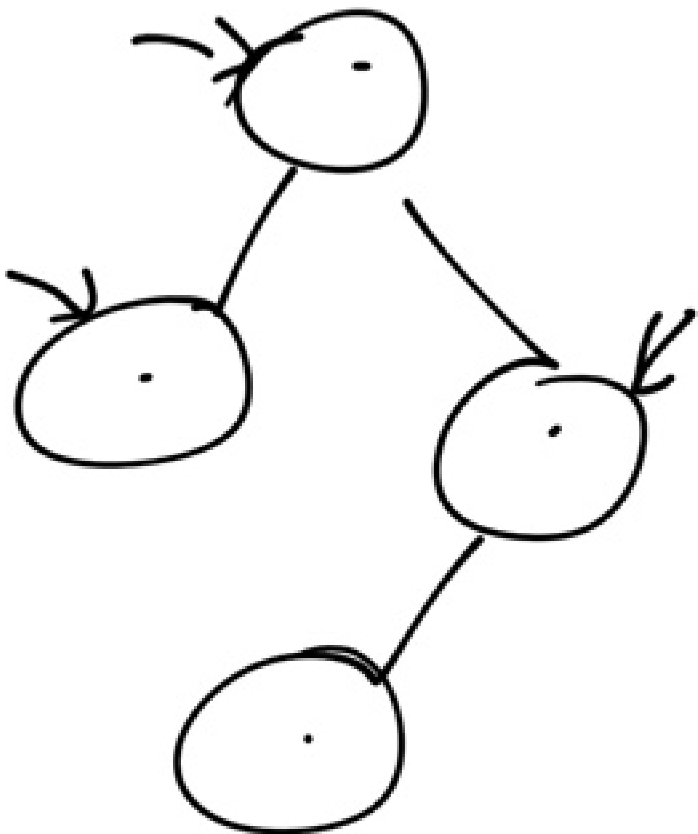
Alberi binari

- albero vuoto
- nodo con (esattamente) due sottoalberi binari (sinistro e destro)

BAMBOO PAPER

2013_11_14

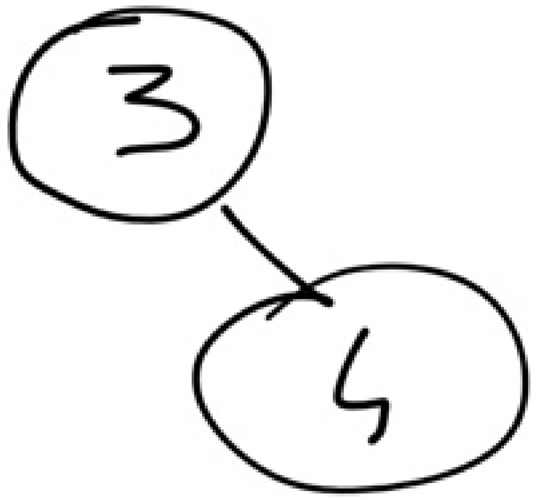




BAMBOO PAPER

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

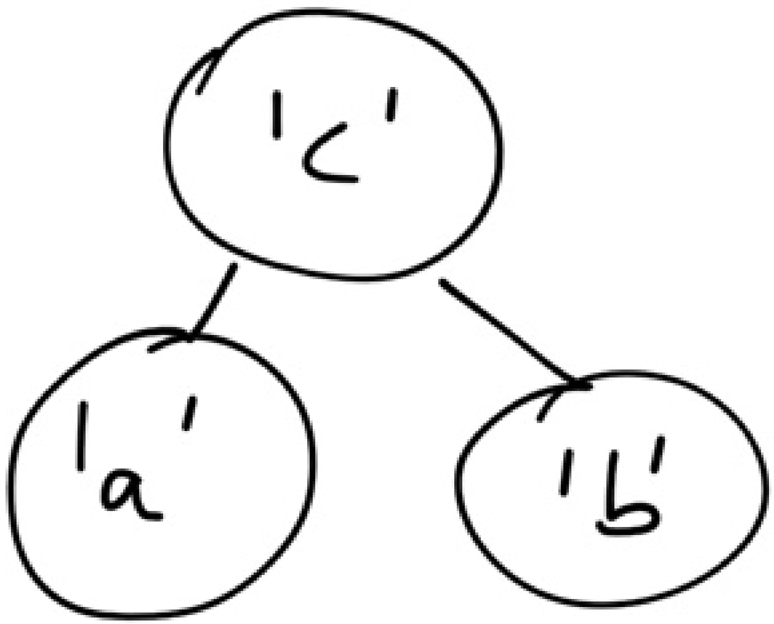
BAMBOOPAPER




```
Node(3, Void,
     Node(4, Void, Void)));;
- : int btree = .....
```

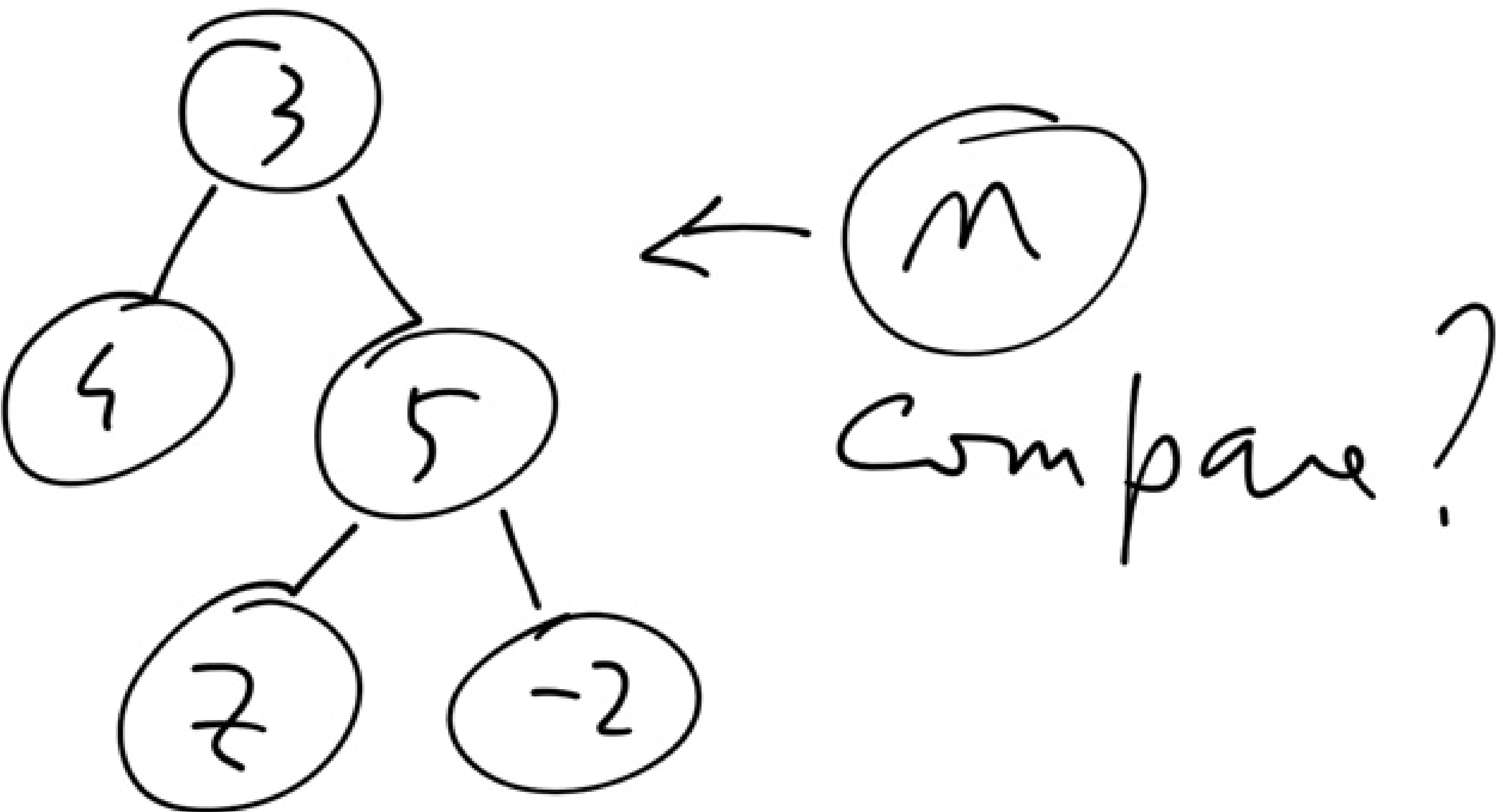
BAMBOO PAPER

BAMBOO PAPER




```
Node('c', Node('a', Void, Void),
          Node('b', Void, Void));;

- : char btree = . ___
```

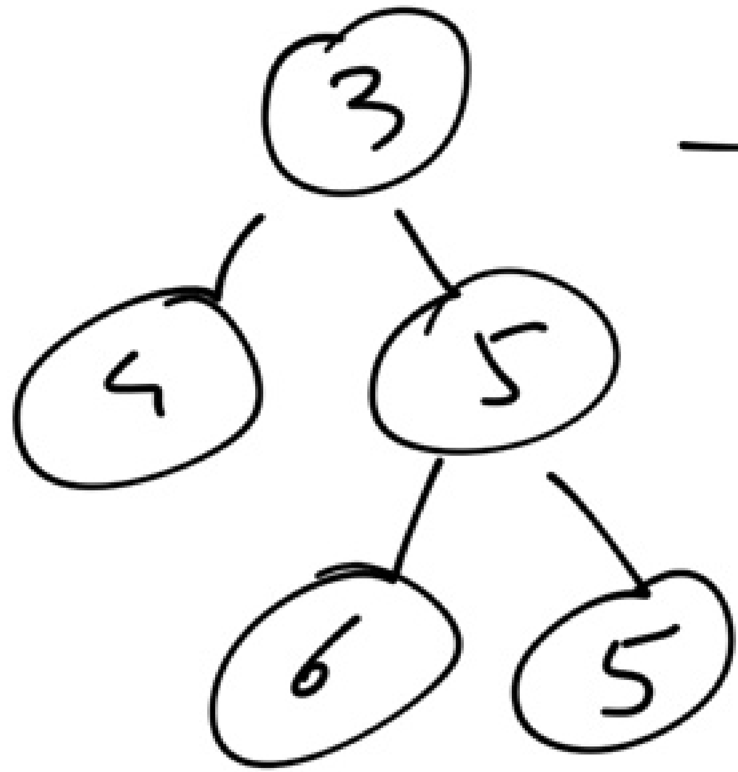


BAMBOO PAPER

```
let rec search el bt =
    match bt with
   Void -> false
  | Node (x, lbt, rbt) when x = el
      -> true
  | Node (x, lbt, rbt) when x <> el
      -> search el lbt or
         search el rbt ;;
```

BAMBOO PAPER

linearizzazioni



→ [3;4;5;6;5]

- anticipata (prima la radice, poi il sottoalbero sinistro infine il sottoalbero destro)

BAMBOO PAPER

BAMBOO PAPER

```
let rec lina bt =
        match bt with
   Void -> []

  Node (x, lbt, rbt) ->
       x :: (lina lbt @ lina rbt);;

lina : 'a btree -> 'a list
         = <fun>
```

differente (linearizza il sotto sinistro poi il sottoalbero destro infine la radice)

mette

BAMBOO PAPER

```
let rec lind bt = match bt with
    Void -> []
  | Node(x, lbt, rbt) ->
    (lind lbt) @ (lind rbt)
      @ [x] ;;

lind : 'a btree -> 'a list
          = <fun>
```

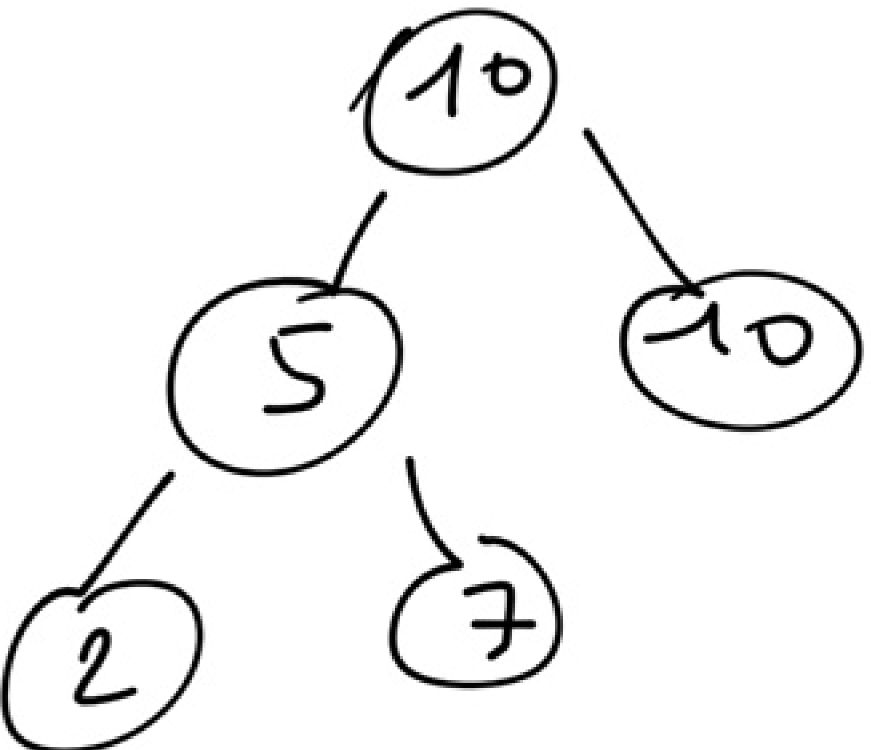
BAMBOO PAPER

Albero binario di ricerca
non vuoto

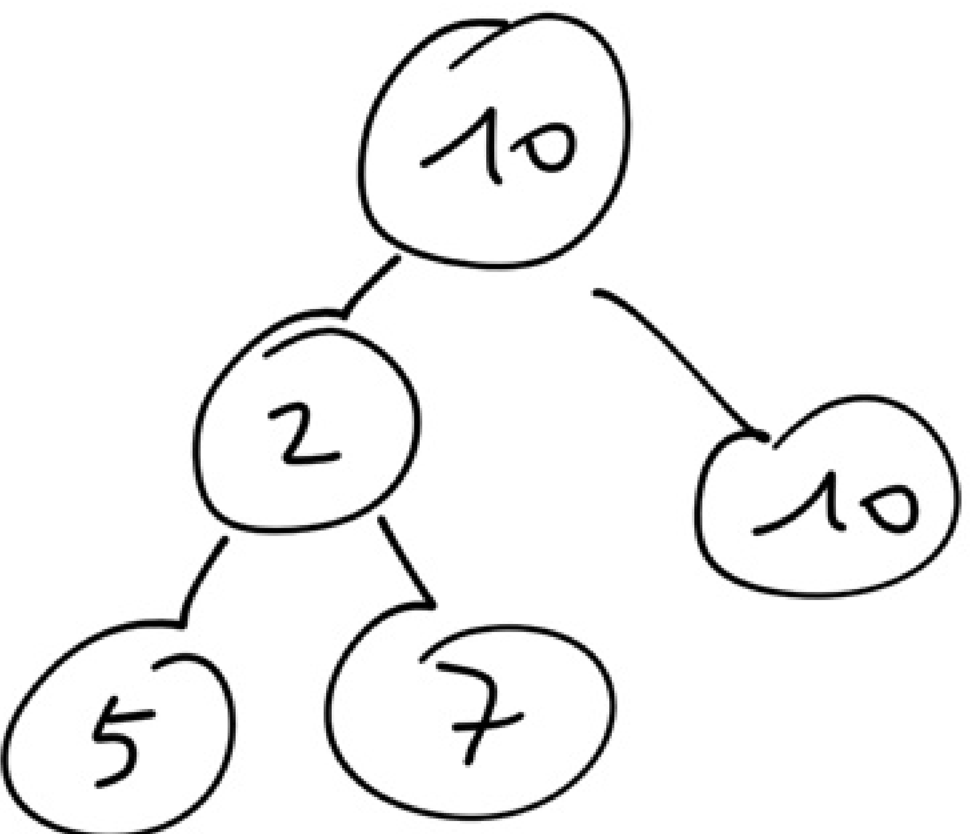
- la radice è maggiore di tutti i nodi del sott-sinistro e minore o uguale a tutti quelli del sott. destro

- sott. sinistro e destro sono alberi binari di ricerca

BAMBOO PAPER

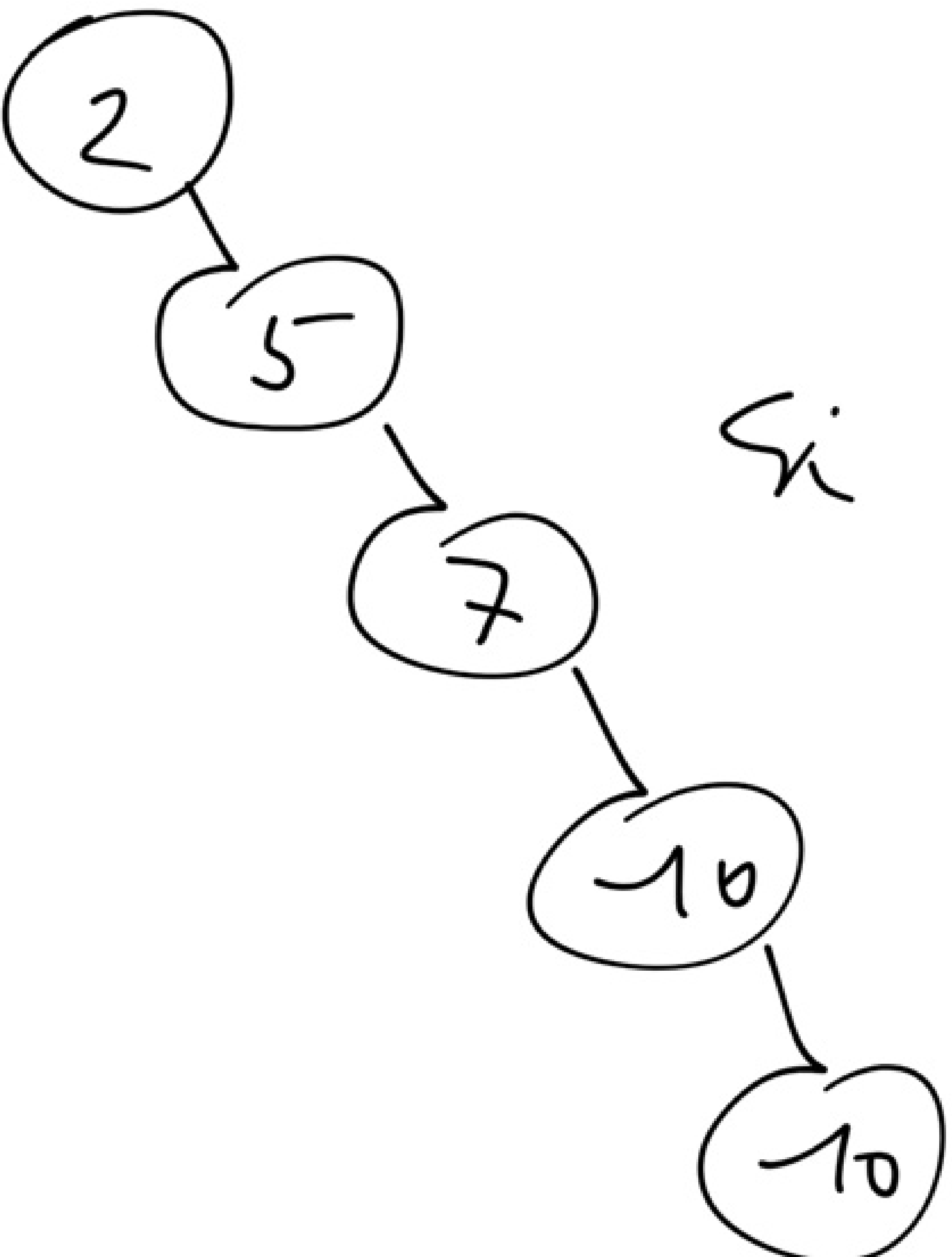


Sì

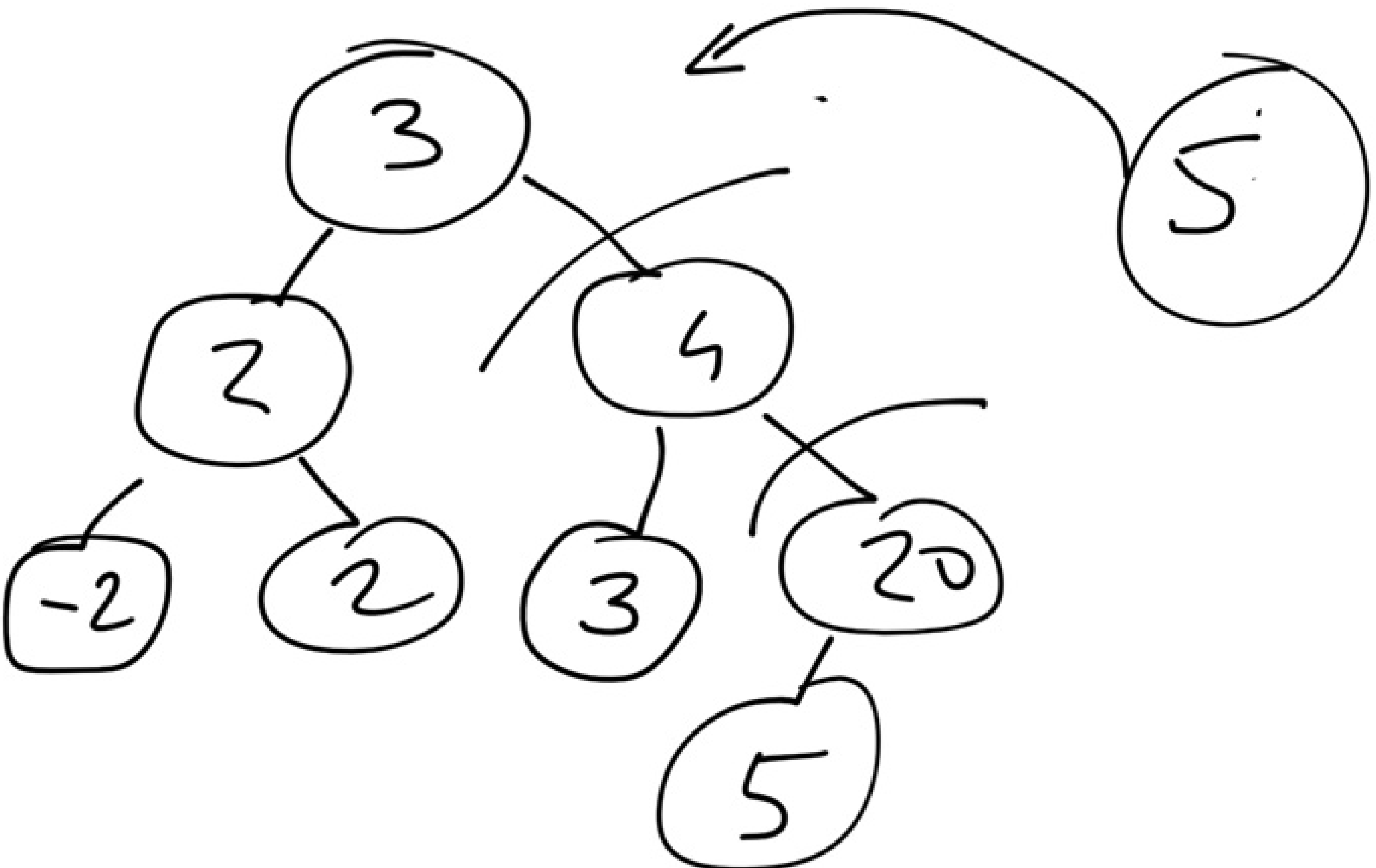


No

BAMBOO PAPER



BAMBOO PAPER



BAMBOO PAPER

2013_11_14

```
let rec ins el sbt =
        match sbt with
    Void -> Node (el, Void, Void)
| Node (x, lsbt, rsbt) when el < x
    -> Node (x, ins el lsbt, rsbt)
Node (x, lsbt, rsbt) when el >= x
 -> Node (x, lsbt, ins el rsbt);;
```


BAMBOO PAPER

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

buildsbt : 'a list -> 'a btree
                    = <fun>
```

BAMBOO PAPER