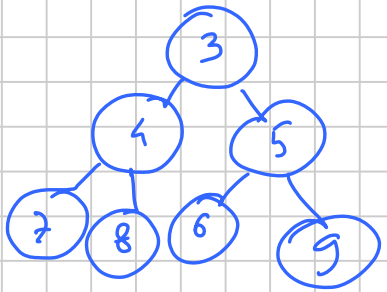
# Linearizzazione di un albero binario

Titolo nota 24/09/2015

Dato un albero binario costruisce una lista con tutti i valori dell'albero.

In che ordine?

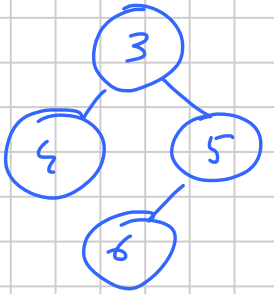
Linearizzazione anticipata: si inserisce nella lista il valore della radice, la linearizzazione (anticipata) del sottoalbero sinistro e poi la linearizzazione del sottoalbero destro




$\Rightarrow$ [3; 4; 7; 8; 5; 6; 9]

Titolo nota 24/09/2015

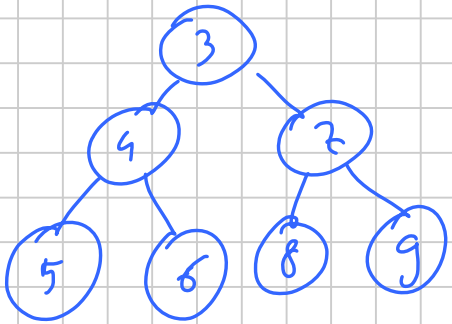
```
# let rec alin bt = match bt with
                    Void -> []
                  | Node(x, lbt, rbt) -> x::(alin lbt @ alin rbt);;

alin : 'a btree -> 'a list = <fun>

# alin Node(3, Node(4, Void, Void), Node(5, Node(6, Void, Void), Void));;
- : int list = [3; 4; 5; 6]
```



# Linearizzazione differita:

Titolo nota 24/09/2015

Prima la linearizzazione differita del sottoalbero sinistro,
poi la " " del " destro e
infine il valore della radice




$\Rightarrow$ [5; 6; 4; 8; 9; 7; 3]

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

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

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

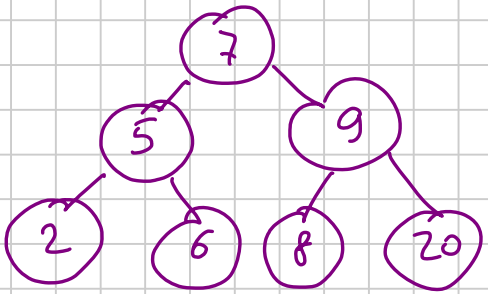
slim : 'a btree → 'a list = <fun>

slim lbt @ [x] @ slim rbt

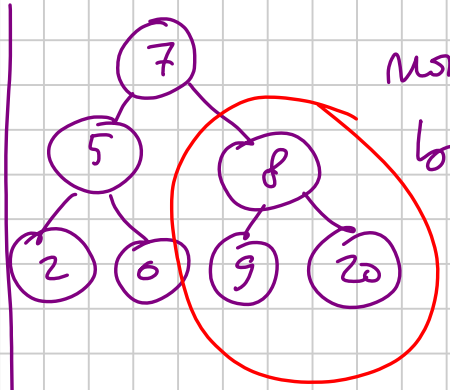
# Alberi binari di ricerca

Titolo nota 24/09/2015

Un "albero binario di ricerca" è un albero binario in cui, in ogni sottoalbero, gli elementi del sottoalbero sinistro sono $\leq$ al valore della radice e gli elementi del sottoalbero destro sono $>$ del valore della radice




è un albero binario di ricerca

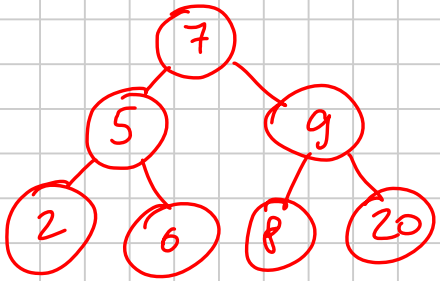



non è un albero binario di ricerca

sottoalbero non è un a.b.r.

# Albero binario "di ricerca"?

Titolo nota 24/09/2015



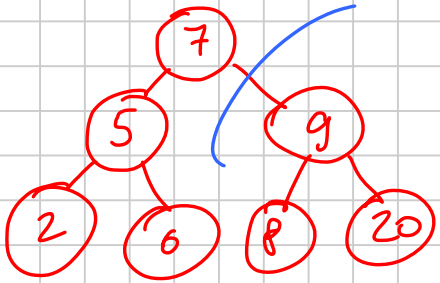

membership in un albero binario in media fa $n/2$ confronti

- nel caso migliore la membership fa un solo confronto (quando l'elemento da cercare si trova nella radice)
- nel caso peggiore (quando l'elemento non compare) si fanno $n$ (numero di nodi dell'albero) confronti.

Albero di $\sim 1000$ nodi $\Rightarrow$ fanno in media $\sim 500$ confronti

# Albero binario "di ricerca"?

Titolo nota 24/09/2015




Dovete cercare un elemento:

- si guarda se l'elemento compare nella radice
- se non compare, si confronta il valore da cercare con il valore della radice, se è minore si cerca nel sottoalbero sinistro se è maggiore si cerca nel sottoalbero destro

(10)

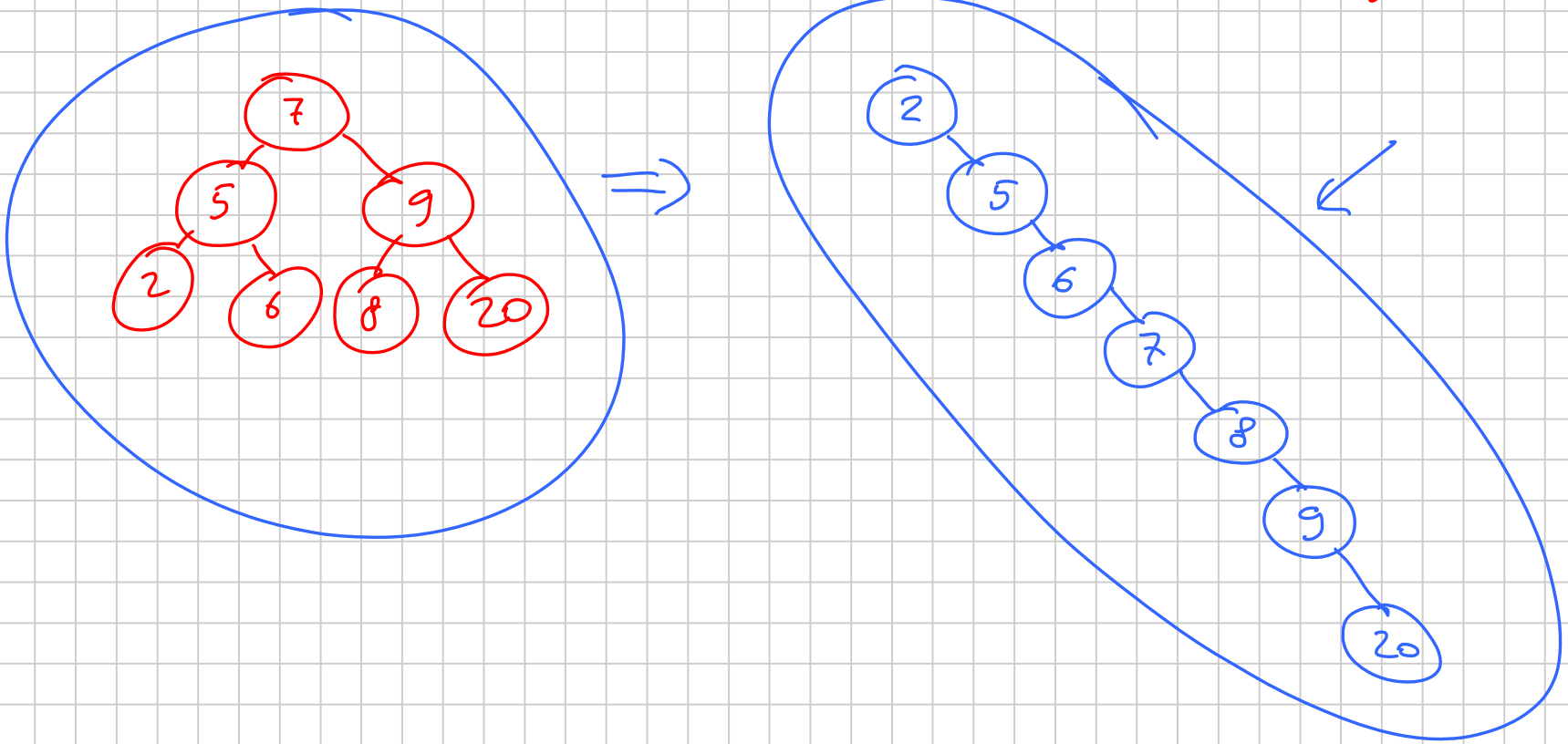
ad ogni passo si eliminano dalla ricerca circa metà dei nodi

So rispondere in un numero di passi (e di confronti) che è $\approx \log_2 n$

Titolo nota 24/09/2015

Se abbiamo un albero binario di ricerca di $\sim 1000$ nodi

la ricerca richiede un numero di passi $\sim \log_2 1000 = \sim \underline{\underline{10}}$



# "Member" in un albero binario di ricerca

Titolo nota 24/09/2015

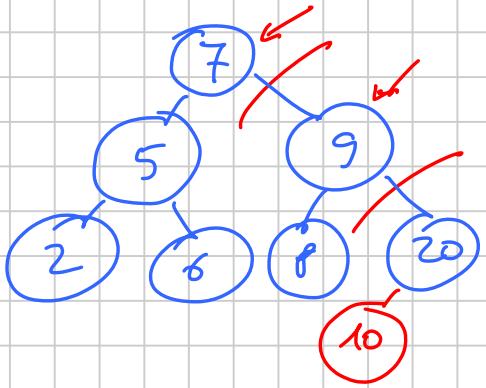
```
# let rec member x bt = match bt with
        Void -> false
  | Node (x, lbt, rbt) when x = x -> true
  | Node (x, lbt, rbt) when x <> x ->
                  if x < x then member x lbt
                           else member x rbt ;;

member : 'a -> 'a btree -> bool = <fun>
```

# Costruzione di un albero binario di ricerca a partire da una lista

Titolo nota 24/09/2015

Definiamo una funzione che prende un valore, un a.b.r. e inserisce il valore nell'albero

ins 10



# ins

Titolo nota 24/09/2015

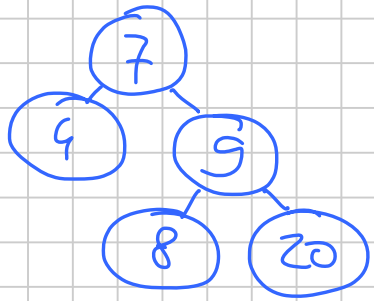
```
# let rec ins n bt = match bt with
        Void -> Node (n, Void, Void)
      | Node(x, lbt, rbt) -> if n <= x then Node (x, ins n lbt , rbt)
                             else Node (x, lbt , ins n rbt) ;;
ins : 'a -> 'a btree -> 'a btree = <fun>

# let rec build l = match l with
        [] -> Void
      | x::xs -> ins x (build xs);;
build : 'a list -> 'a btree = <fun>
```

Titolo nota 24/09/2015

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

build [20;8;9;4;7]
= {def build, 2° p}
ins 20 (build [8;9;4;7])
= { " }
ins 20 (ins 8 (build [9;4;7]))
= { " }
ins 20 (ins 8 (ins 9 (build [4;7]))

= { " }
ins 20 (ins 8 (ins 9 (ins 4 (build [7]))))
= { " }
ins 20 (ins 8 (ins 9 (ins 4 (ins 7 (build []))))
= { " 1° p}
ins 20 (ins 8 (ins 9 (ins 4 (ins 7 Void))))



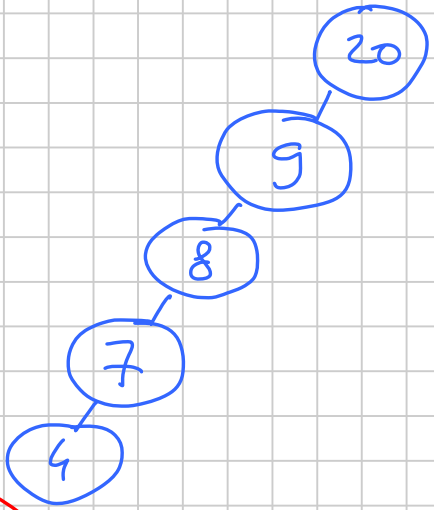
```
let rec build l = match l with
    [] -> Void
  | x :: xs -> ins x (build xs);;
```

build [20;8;9;4;7]

build [4;7;8;9;20]

= :

ins 4 (ins 7 (ins 8 (ins 9 (ins 20 []))))



Titolo nota 24/09/2015

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

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

```
build [3;4;7;8] ;;
```

Titolo nota 24/09/2015

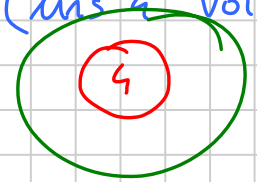
build [4;7;9;8]

= {def build}

builda [4;7;9;8] Void

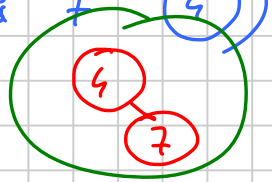
= {def. builda, 2° p}

builda [7;9;8] (ins 4 Void)



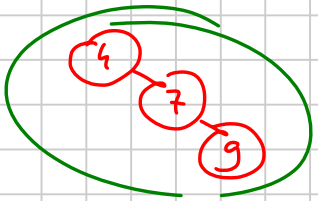

= {"}

builda [9;8] (ins 7 (4))



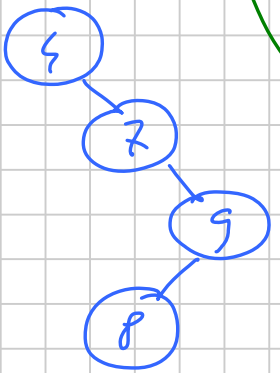

= {"}

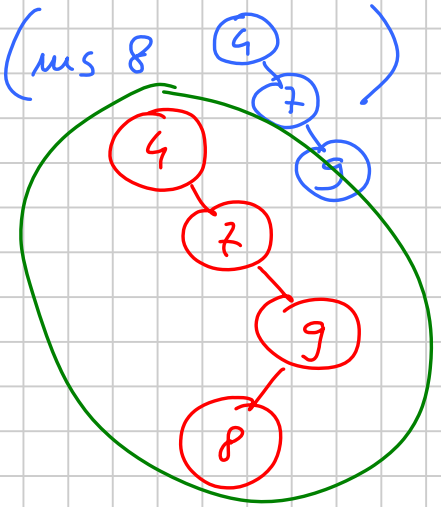
builda [8] (ins 9 (4 (7)))




builda [] (ins 8 (4 (7 (9))))

=



# Trovare il valore massimo in un a.b.r.

Titolo nota 24/09/2015

```
let rec maxabr bt = match bt with
      Node(x, Void, Void) -> x
    | Node (x, lbt, Void) when lbt <> Void -> x
    | Node (x, lbt, rbt) when rbt <> Void -> maxabr rbt ;;
```

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

# Ricerca max in un albero binario

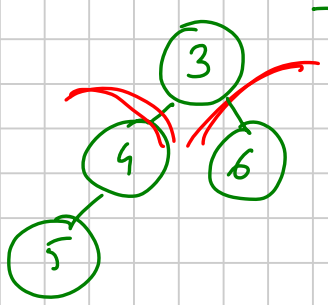
Titolo nota 24/09/2015

```
let rec maxb bt = match bt with
      Node (x, Void, Void) -> x
    | Node (x, Void, rbt) when rbt <> Void -> max (x, maxb rbt)
    | Node (x, lbt, Void) when lbt <> Void -> max (x, maxb lbt)
    | Node (x, lbt, rbt) when lbt <> Void & rbt <> Void ->
                max (x, max (maxb lbt, maxb rbt)) ;;
```

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

Titolo nota 24/09/2015

# Profondità di un albero (la lunghezza del cammino più lungo dalla radice a una foglia)




profondità = 3

Void ha profondità $\varnothing$

```
let max (n, m) =
    if n > m then n
             else m ;;
```

```
let rec prof bt = match bt with
    Void -> ∅
  | Node (x, lbt, rbt) ->
        1 + max (prof lbt , prof rbt)
```

Titolo nota 24/09/2015

```
let rec prof bt = match bt with
    Void -> 0
  | Node (x, lbt, rbt) -> 1 + ( if prof lbt > prof rbt
                                then prof lbt
                                else prof rbt )
```

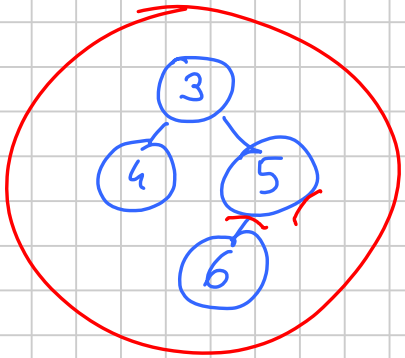
---

```
-> 1 + ( let m1 = prof lbt
         and m2 = prof rbt
         in if m1 > m2 then m1
                       else m2 ) ;;
```

# Alberi binari

Titolo nota 24/09/2015

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




```
Node(3, Node(4, Void, Void),
        Node(5, Node(6, Void, Void), Void))
```

```
- : int btree = <abstr>
```

```
let build l = ....
```

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