Corso di laurea in Informatica
Algoritmi 1
A.A. 2025/2026

Dizionari:
Alberi binari di ricerca

Tiziana Calamoneri

SAPIENZA
UNIVERSITÀ DI ROMA

Slides realizzate sulla base di quelle preparate da T. Calamoneri e G. Bongiovanni per il corso di Informatica Generale tenuto a distanza nell'A.A. 2019/20

Sommario

Dizionari

Alberi binari di ricerca:

- algoritmo di ricerca e suo costo computazionale
- algoritmo di inserimento
- ricerca di massimo e minimo
- ricerca di predecessore e successore
- algoritmo di cancellazione

Dizionari

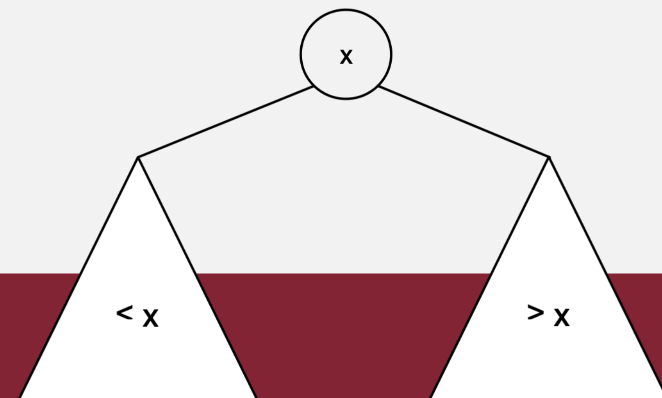
Un dizionario è una struttura dati che permette di gestire un insieme dinamico di dati, che di norma è un insieme totalmente ordinato, tramite queste tre sole operazioni:

- insert: si inserisce un elemento;
- search: si ricerca un elemento;
- delete: si elimina un elemento.

ABR (1)

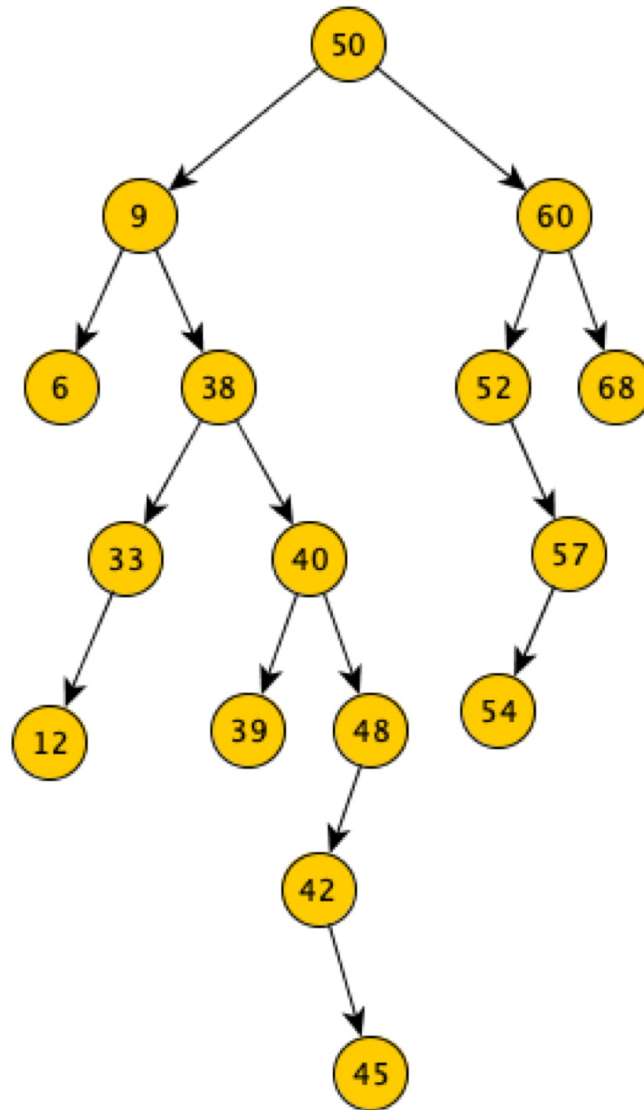
Un albero binario di ricerca (ABR) è un albero in cui vengono mantenute le seguenti proprietà:

- ogni nodo contiene una chiave
- il valore della chiave contenuta in ogni nodo è maggiore della chiave contenuta in ciascun nodo del suo sottoalbero sinistro (se esiste)
- il valore della chiave contenuta in ogni nodo è minore della chiave contenuta in ciascun nodo del suo sottoalbero destro (se esiste)



ABR (2)

Un esempio:



ABR (3)

Gli ABR sono strutture dati che supportano tutte le operazioni già definite in relazione agli insiemi dinamici più alcune altre.

Operazioni di interrogazione:

- Search(T,k): restituisce un puntatore all'elemento con chiave di valore k in T se questo è presente, None altrimenti;
- …

ABR (4)

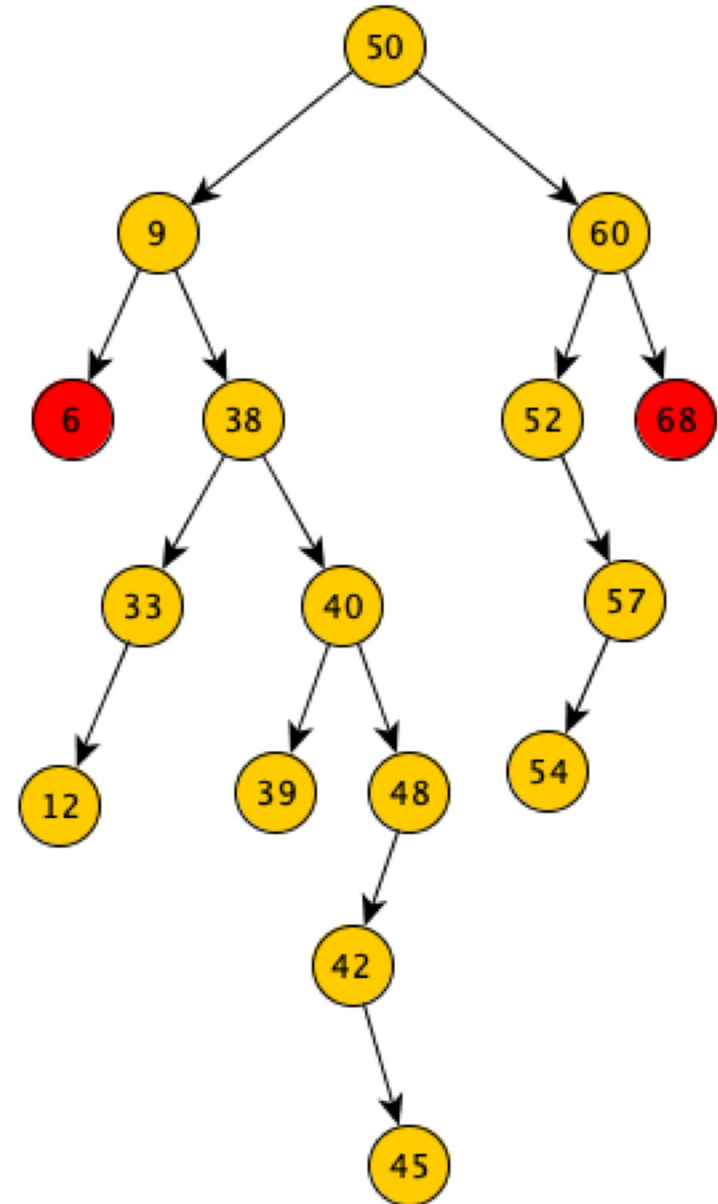
- …
- Minimum(T)/Maximum(T): restituisce un puntatore all'elem. in T con chiave min/max;
- Predecessor(T,p)/Successor(T,p): restituisce un puntatore all'elem. in T con la chiave che precederebbe/seguirebbe in una sequenza ordinata la chiave contenuta nel nodo puntato da p.

Operazioni di manipolazione:

- Insert(T, k): inserisce un elem. di chiave k in T;
- Delete(T, p): elimina da T l'elem. puntato da p.

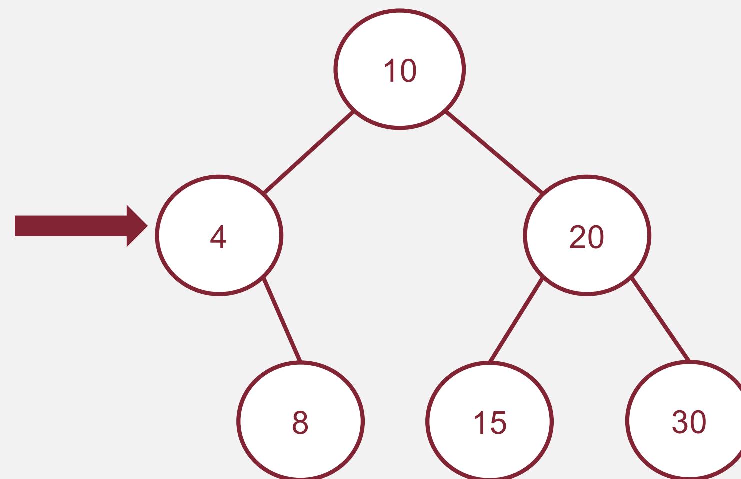
ABR (5)

Un ABR può essere usato sia come dizionario che come coda di priorità: il minimo è sempre nel nodo più a sinistra, il massimo in quello più a destra.



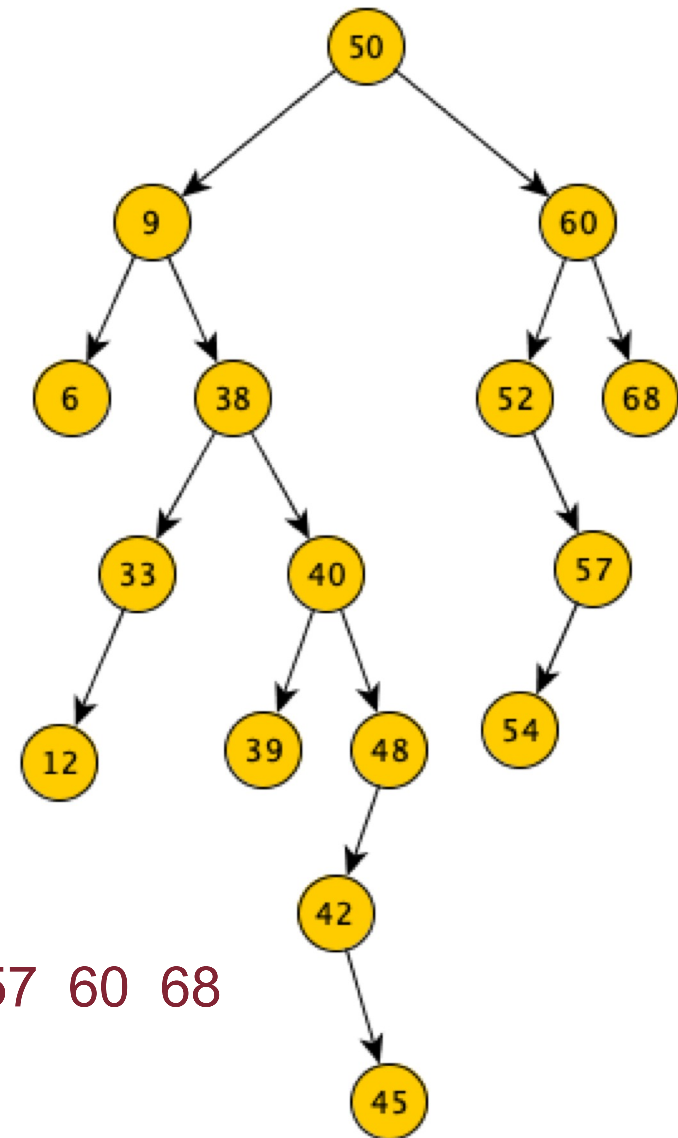
ABR (6)

N.B. il nodo più a sinistra non necessariamente è una foglia: può anche essere un nodo che non ha un figlio sinistro; analoga considerazione per il nodo più a destra.



ABR (7)

Per elencare tutte le chiavi in ordine crescente basta eseguire una visita in-ordine.



6 9 12 33 38 39 40 42 45 48 50 52 54 57 60 68

ABR (8)

Dunque un ABR può anche essere visto come una struttura dati su cui eseguire un algoritmo di ordinamento, costituito di due fasi:

- inserimento di tutte le n chiavi da ordinare in un ABR, inizialmente vuoto;
- visita in-ordine dell'ABR appena costruito.

Il costo computazionale di tale algoritmo è:

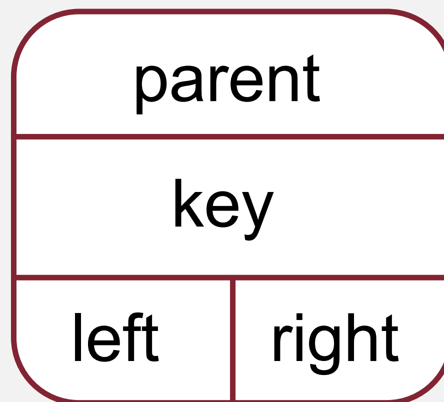
Costo(costruzione ABR) + Costo(visita) =

Costo(costruzione ABR) + $\Theta(n)$

Più avanti determineremo il costo della costruzione di un ABR.

ABR (9)

Nel seguito assumiamo che l'ABR sia memorizzato tramite record a puntatori e che ciascun nodo abbia, oltre ai soliti puntatori ai figli destro e sinistro, anche un puntatore al padre (utile per risalire agli antenati).



ABR – ricerca (1)

Ricerca

Concettualmente simile alla ricerca binaria (anche se i dati non vengono divisi a metà ad ogni passo): si esegue una discesa dalla radice che viene guidata dai valori memorizzati nei nodi che si incontrano lungo il cammino.

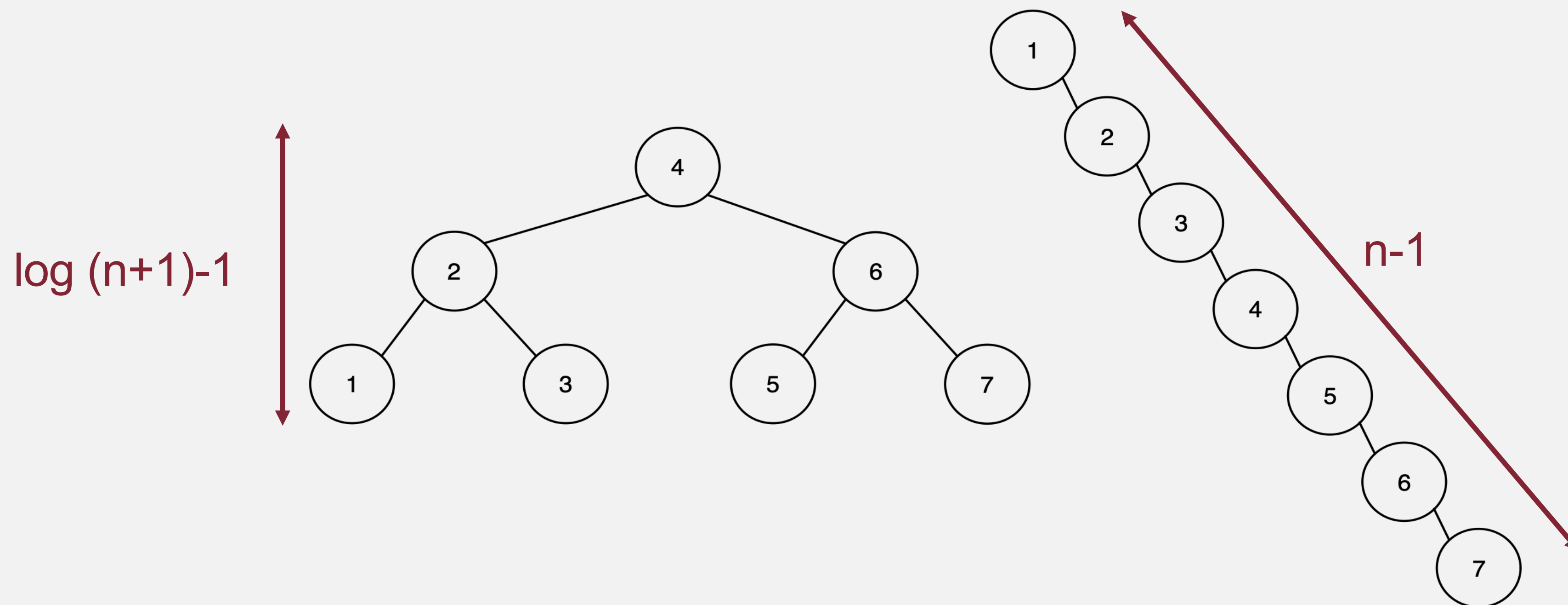
```
def ABR_searchRic(p,k):
  if (p == None) or (p.key = k):
    return p
  if (k < p.key): return ABR_searchRic(p.left, k)
  else:           return ABR_searchRic(p.right, k)
```

ABR – ricerca (2)

Considerando che la ricerca su un ABR ricorda molto da vicino la ricerca binaria (costo computazionale O(log n)), come mai non riusciamo a garantire un costo logaritmico anche per la ricerca su un ABR?

ABR – ricerca (3)

Hint: viene eseguito un passo per ogni livello, ma l'ABR non include fra le sue proprietà alcuna limitazione sulla sua altezza.



ABR – considerazioni sul costo comput. (1)

Analogamente al Quicksort, si dimostra che il comportamento degli ABR nel caso medio è molto più vicino al caso migliore del caso peggiore:

Teorema. L'altezza attesa di un albero binario di ricerca costruito in modo casuale con n chiavi tutte distinte è O(log n).

ABR – considerazioni sul costo comput. (2)

Di conseguenza, una strategia che, **in media**, costruisce un albero bilanciato per un insieme fisso di elementi consiste nel permutare in modo casuale gli elementi e poi nell’inserire gli elementi in quell’ordine nell’albero.

Questa tecnica non può essere usata se non abbiamo tutti gli elementi contemporaneamente, perché ad esempio (come spesso accade) riceviamo gli elementi uno alla volta…

ABR – considerazioni sul costo comput. (3)

Quindi: se vogliamo garantire che il costo computazionale della ricerca su ABR sia limitata superiormente da un logaritmo, dovremo preoccuparci di mettere in campo qualche tecnica che ci permetta di tenere sotto controllo la crescita dell'altezza:

bilanciamento in altezza

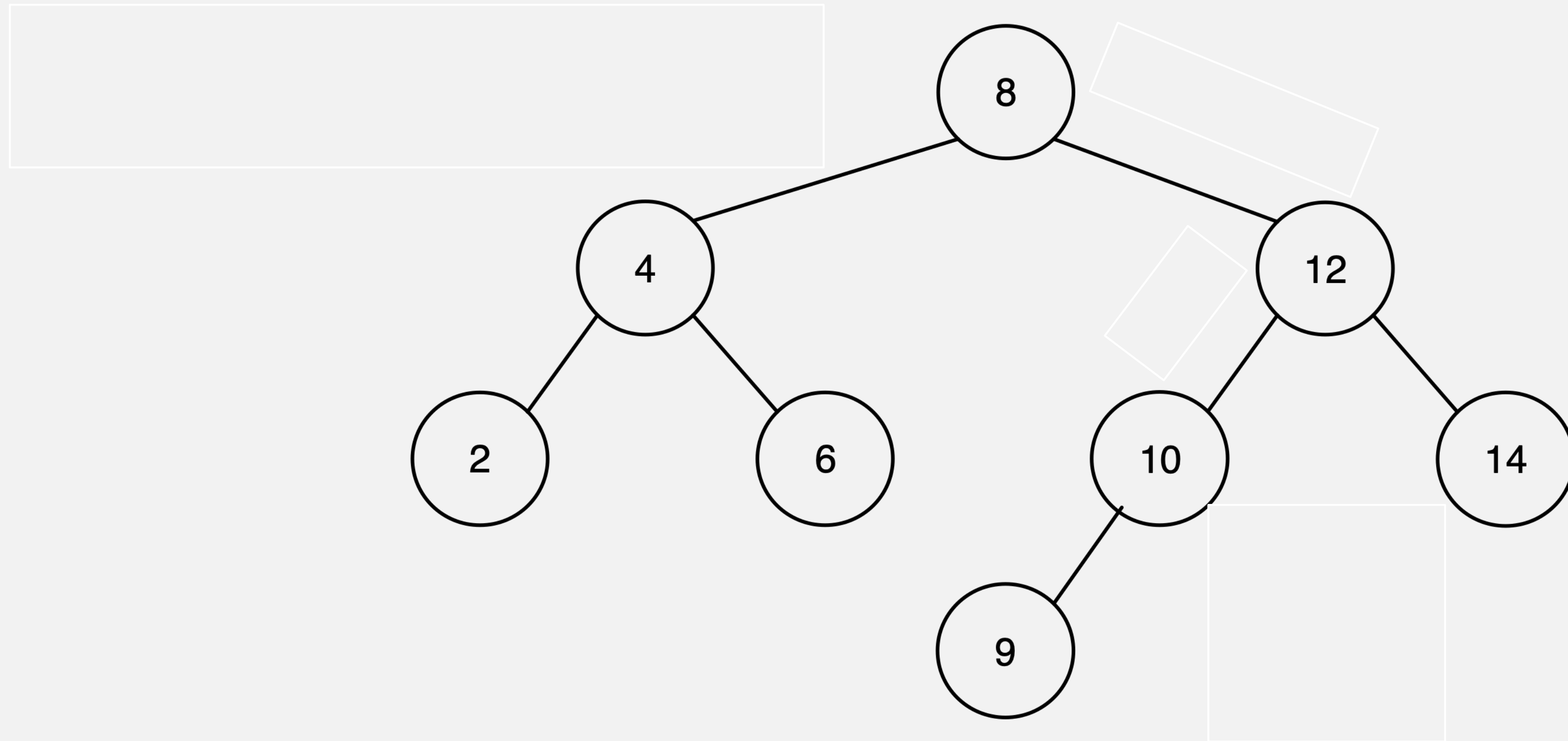
(di cui riparleremo).

ABR – inserimento (1)

Inserimento

- Come nella ricerca, si scende guidati dalle chiavi memorizzate nei nodi che si incontrano lungo il cammino.
- Quando si arriva al punto di voler proseguire la discesa verso un puntatore vuoto (None), si aggiunge il nuovo nodo.
- Il padre di tale nuovo nodo potrebbe essere una foglia (entrambi i suoi figli sono None) ma, più in generale, è un nodo a cui manca il figlio corrispondente a quello che si inserisce.

ABR – inserimento (2)



ABR – inserimento (3)

```
def ABR_insert (p,k):
    y,x=None,p                     #y punta sempre al padre di x
    z=NodoABR(k)                   # si alloca nuova memoria
    while (x ≠ None)               #discesa alla prima pos. disponib.
        y = x
        if z.key < x.key:
            x= x.left
        else:
            x= x.right
    if (y==None)                   #se albero inizialmente vuoto
        p = z
    else:
        if (z.key < y.key):
            y.left= z
        else:
            y.right = z
    z.parent = y                   #collegam. padre - nodo da inser.
    return p                       #p potrebbe essere cambiato
```

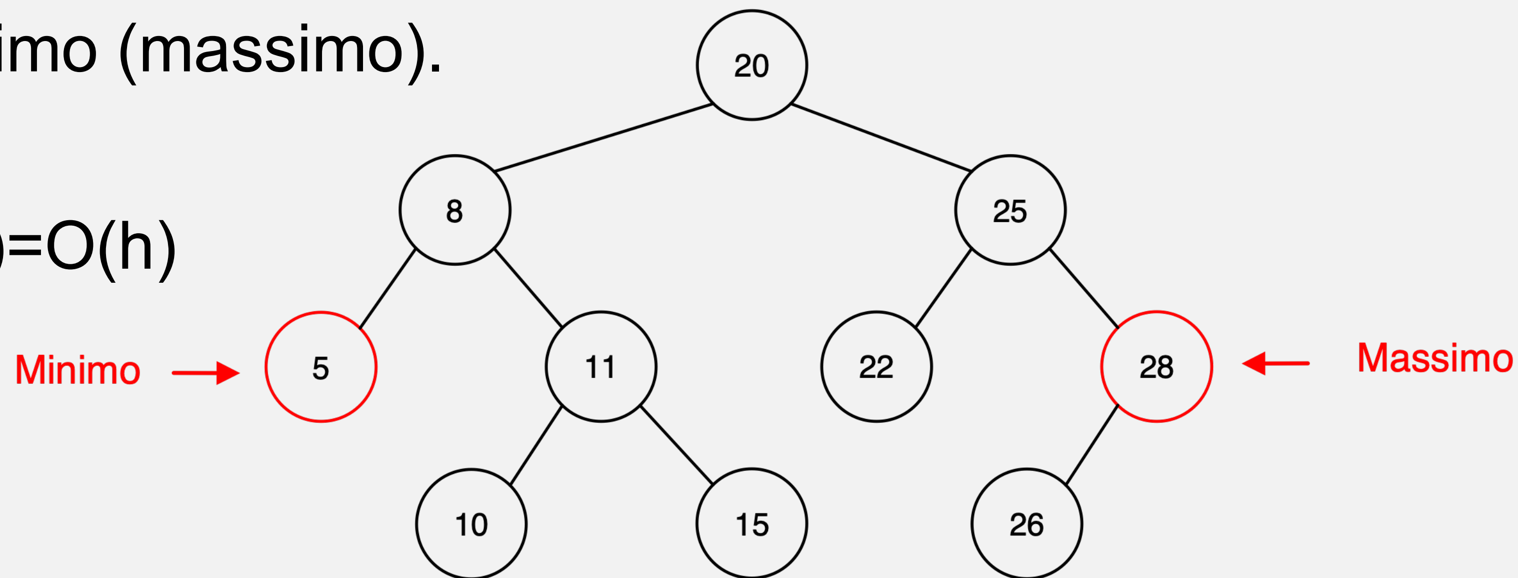
ciclo eseguito al max h volte quindi $T(h)=O(h)$

ABR – minimo e massimo

Minimo e massimo

Il minimo (massimo) si trova nel nodo più a sinistra (destra), quindi per trovarlo si scende sempre a sinistra (destra) a partire dalla radice. Ci si ferma quando si arriva a un nodo che non ha figlio sinistro (destro): quel nodo contiene il minimo (massimo).

T(h)=O(h)



ABR – predecessore e successore (1)

Predecessore e successore

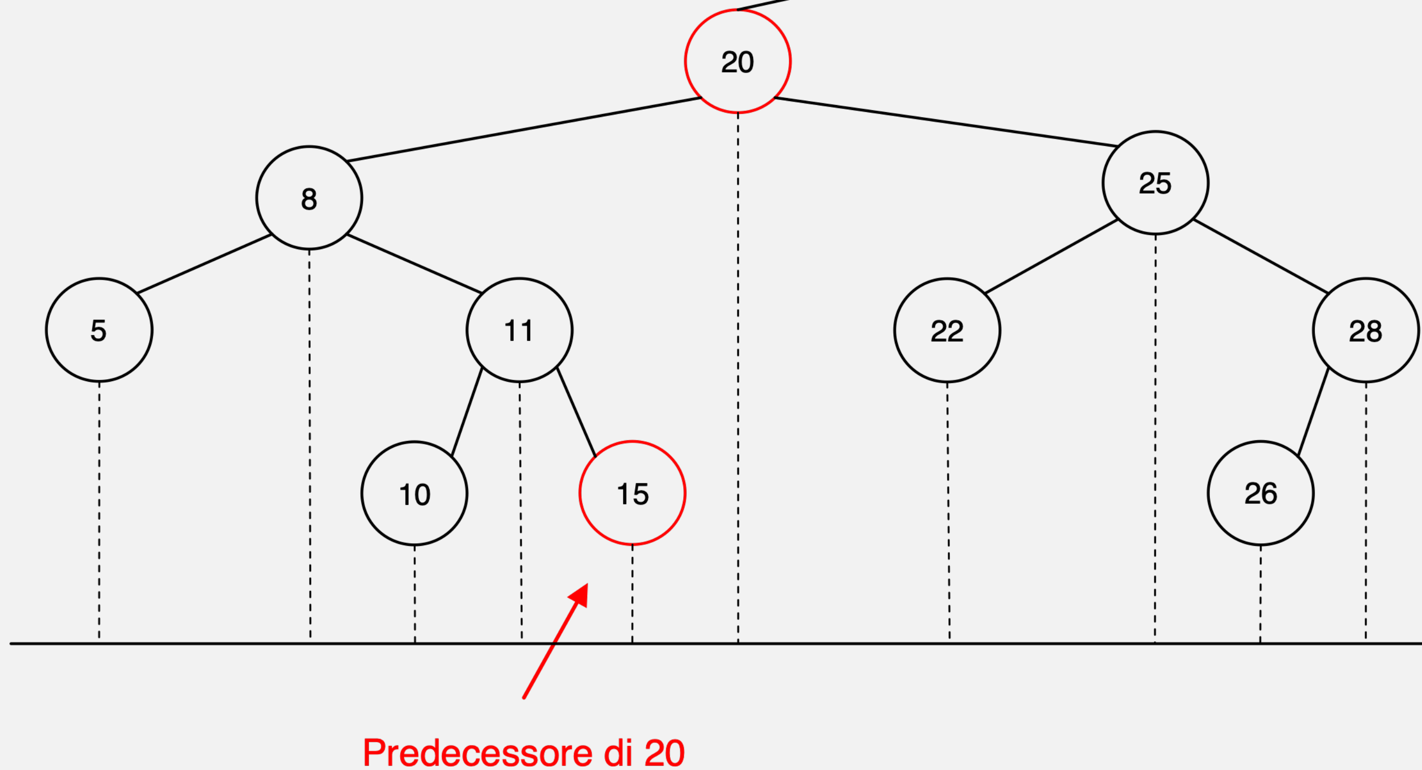
Ricordiamo che:

- il predecessore è il nodo che contiene la chiave che precederebbe immediatamente k se le chiavi fossero ordinate
- il successore è il nodo che contiene la chiave che seguirebbe immediatamente k se le chiavi fossero ordinate.

Concentriamoci sulla ricerca del predecessore.

ABR – predecessore e successore (2)

Caso 1: Se il nodo ha il sottoalbero sinistro, il suo predecessore è il massimo di tale sottoalbero:



ABR – predecessore e successore (3)

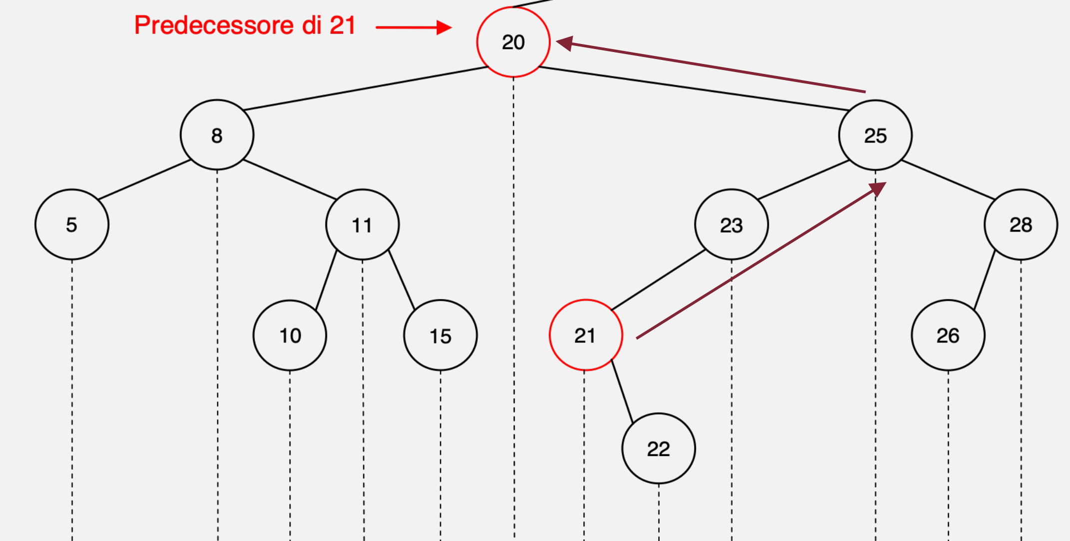
Caso 2: se il nodo che contiene k non ha sottoalbero sinistro allora esso è il nodo più a sinistra del suo sottoalbero, e quindi il min di tale sottoalbero.
Il suo predecessore sarà il nodo immediatamente alla sua sinistra che, non stando nel sottoalbero, deve stare tra i suoi antenati…

ABR – predecessore e successore (4)

Caso 2 (segue)

Per trovare il predecessore di k bisogna quindi:

- risalire alla radice di quel sottoalbero, il che significa salire a destra finché è possibile;
- una volta giunti nella radice del sottoalbero, si risale (con un singolo passo di salita a sinistra) a suo padre che è il predecessore di k.



ABR – predecessore e successore (5)

Una situazione perfettamente simmetrica esiste per il problema di trovare il successore di un nodo.

Entrambe tali operazioni richiedono una discesa lungo un singolo cammino a partire dalla radice oppure una singola risalita verso la radice.

Per entrambe le funzioni il costo è limitato superiormente dall'altezza dell'albero: O(h).

ABR – risalita verso destra o verso sinistra?

Come si fa a sapere se il passo di risalita dal nodo x al padre di x avviene salendo verso destra o verso sinistra?

Bisogna controllarlo esplicitamente con un test, ad ogni passo, prima di salire:

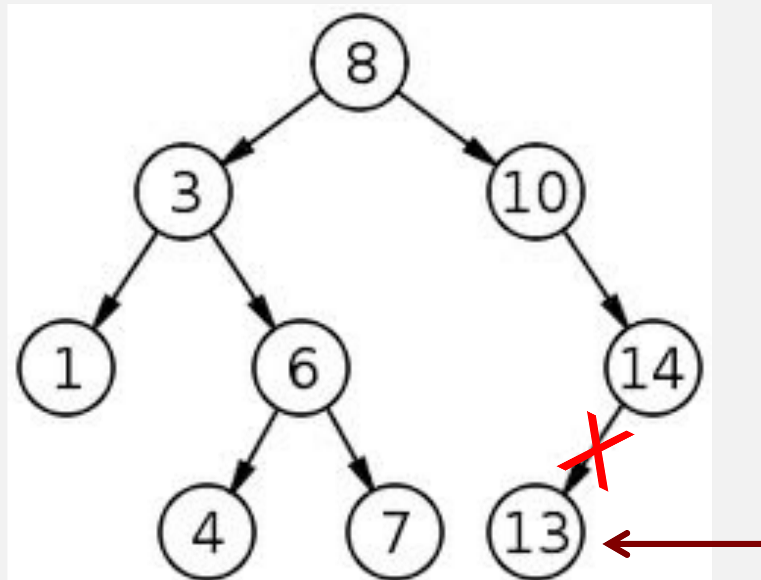
- `if (x == x.parent.left` allora la risalita sarà verso destra;
- `if (x == x.parent.right` allora la risalita sarà verso sinistra.

ABR – cancellazione (1)

Cancellazione:

Per eliminare un nodo in un ABR:

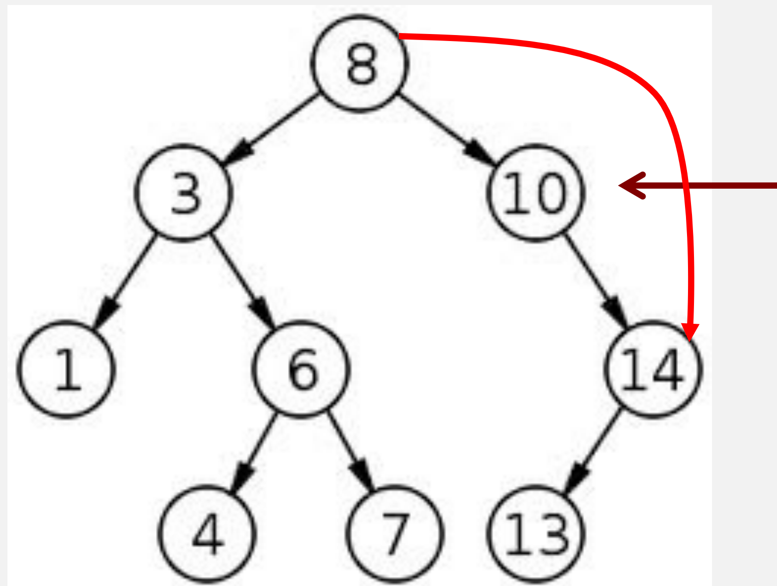
- Caso 1: se il nodo è una foglia lo si elimina, ponendo a None l'opportuno campo nel nodo padre;



ABR – cancellazione (1)

Cancellazione (segue)

- Caso 2: se il nodo ha un solo figlio lo si “cortocircuita”, cioè si collegano direttamente fra loro suo padre e il suo unico figlio, indipendentemente che sia destro o sinistro;



ABR – cancellazione (3)

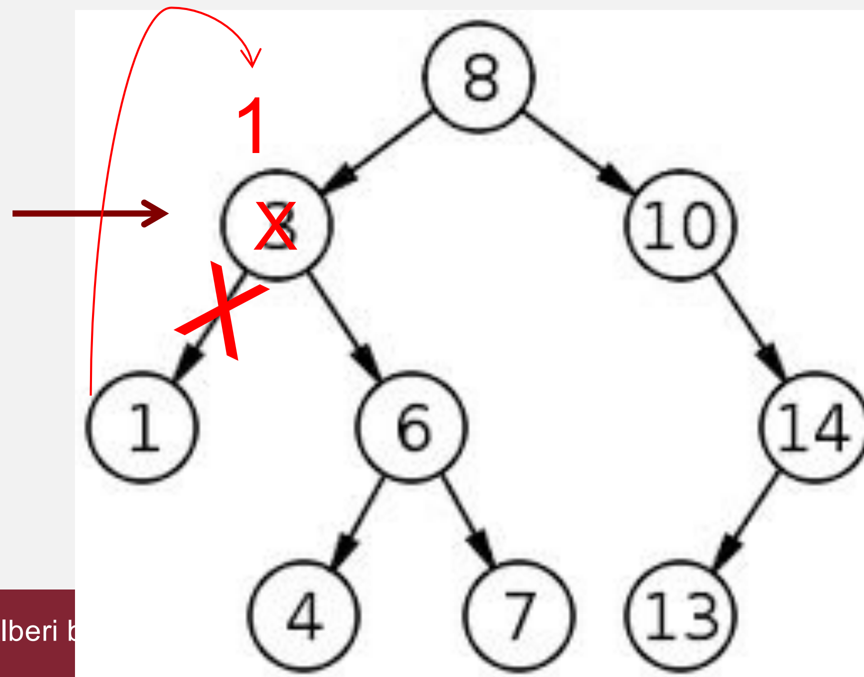
Problema: Se la chiave da eliminare è in un nodo con due figli va riaggiustato l'albero dopo l'eliminazione per evitare che si disconnetta.

Riaggiustare l'albero significa trovare un nodo da collocare al posto del nodo eliminato, per mantenere l'albero connesso e garantire la proprietà fondamentale degli ABR.
Tale nodo può quindi essere solamente il predecessore o il successore del nodo da eliminare.

ABR – cancellazione (4)

Cancellazione (segue)

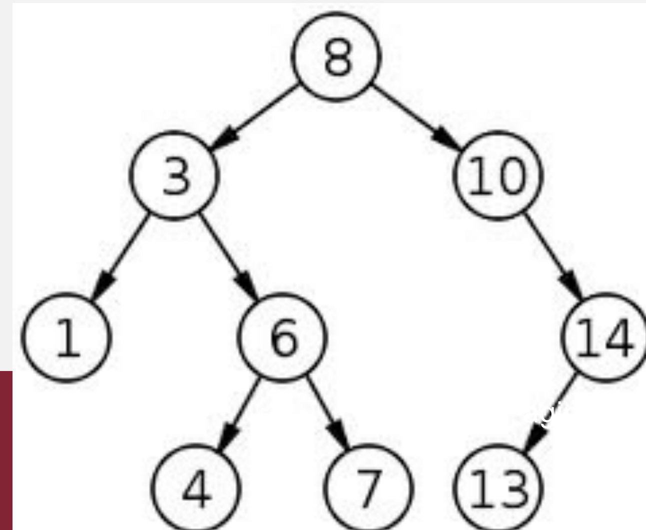
Caso 3: se il nodo ha entrambi i figli lo si sostituisce col predecessore (o col successore), che va quindi tolto (ossia eliminato) dalla sua posizione originale.



ABR – cancellazione (5)

Cancellazione (segue)

Il predecessore (o il successore) del nodo da cancellare ricade sicuramente nel *caso 1* dell'algoritmo per la ricerca del predecessore perché il nodo da cancellare ha due figli. **L'eliminazione del predecessore** ricade necessariamente in uno dei due casi precedenti perché non ha figlio sinistro.



Esercizio svolto 1 (1)

Esercizio 1. Progettare un algoritmo che, dati due ABR T_1 e T_2, rispettivamente con n_1 ed n_2 nodi, ed altezza h_1 ed h_2, dia in output un ABR che contiene tutti gli n_1+n_2 nodi.

Fare le opportune osservazioni sul costo computazionale e sull'altezza dell'ABR risultante, come funzione di h_1 e h_2.

Soluzione. Inseriamo nell'albero con il maggior numero di nodi (senza perdere di generalità sia esso T_1) i nodi dell'altro albero uno ad uno…

Esercizio svolto 1 (2)

segue Esercizio 1.

… Sapendo che l'operazione di inserimento in un ABR ha un costo dell'ordine dell'altezza dell'albero, nel caso peggiore, il costo computazionale è:

$$O(h_1) + O(h_1+1) + O(h_1+2) + \ldots + O(h_1 + n_2 - 1) =$$
$$= n_2 O(h_1) + O(1 + 2 + \ldots + n_2 - 1)$$

Poiché $O(1 + 2 + \ldots + n_2) = O(n_2^2)$ e, nel caso peggiore, $O(h_1) = O(n_1)$, il costo di questo approccio è $O(n_1 n_2) + O(n_2^2) = O(n_1 n_2)$, perché $n_1 > n_2$ per ipotesi.

Esercizio svolto 1 (3)

segue Esercizio 1.

Oss. questo procedimento è corretto perché un albero binario di ricerca **non** è necessariamente bilanciato, e quindi poco importa che l'altezza dell'albero risultante possa diventare $O(h_1 + n_2)$.

Esercizio svolto 2 (1)

Esercizio 2. Progettare un algoritmo che, dati due ABR T_1 e T_2, rispettivamente con n_1 ed n_2 nodi, ed altezza h_1 ed h_2, dia in output un ABR che contiene tutti gli n_1+n_2 nodi, assumendo che tutti gli elementi in T_1 siano minori di quelli in T_2.

Soluzione. Oltre alla soluzione dell'esercizio precedente, proponiamo un altro approccio per tentare di sfruttare l'ipotesi…

Esercizio svolto 2 (2)

segue Esercizio 2.

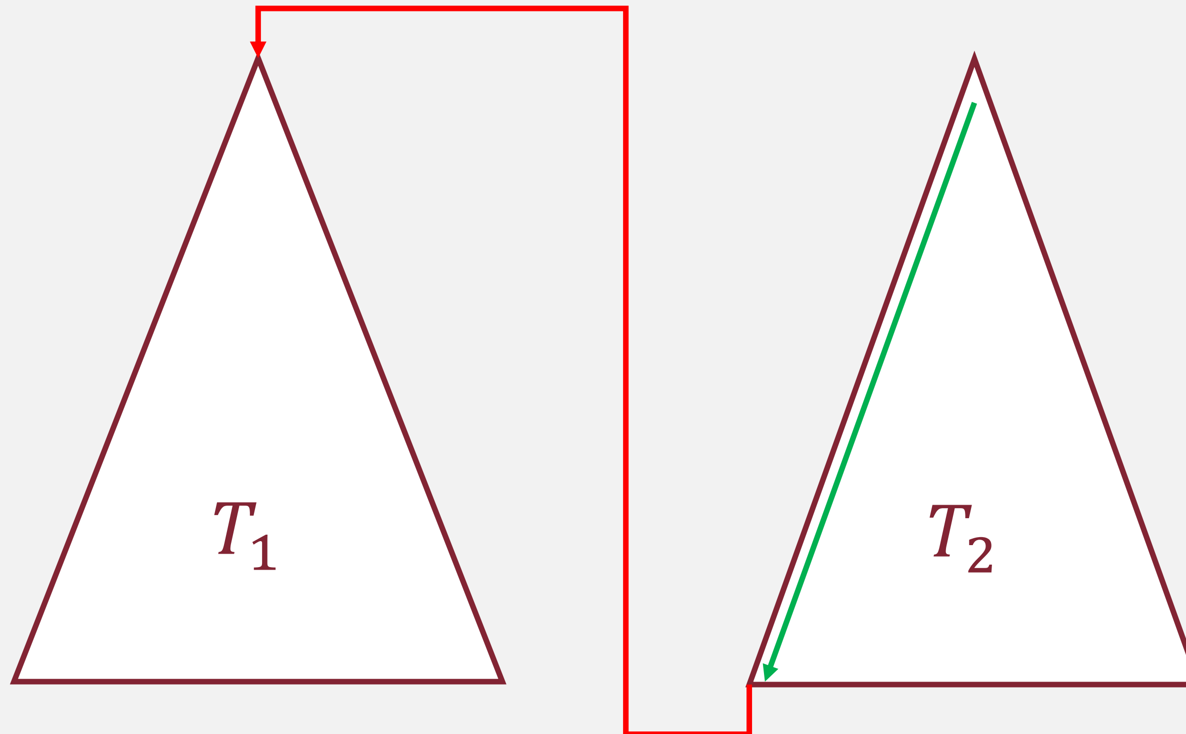
… "appendiamo" l'albero T_1 come figlio sx del min di T_2. Questo si può sempre fare perché il min di un ABR è il nodo più a sx che non possiede figlio sx, pertanto basta:

- far scendere un puntatore dalla radice di T_2 verso il figlio sx finché esso esista
- giunti al nodo che non ha figlio sx (= min dell'albero), agganciargli a sx la radice di T_1.

Il costo è dominato dal costo della ricerca del minimo, cioè da $O(h_2)$.

Esercizio svolto 2 (3)

segue Esercizio 2.



Esercizio svolto 3 (1)

Esercizio. Scrivere lo pseudocodice dell'inserimento di una nuova chiave z in un ABR memorizzato mediante la notazione posizionale.

Assunzioni:

- L'array contiene n posizioni (indici da 0 a n-1);
- Nell'array si usa il simbolo '-' per denotare un nodo dell'albero mancante.

Esercizio svolto 3 (2)

segue Esercizio 3.

```
def ABR_insert (A, z):
   n=len(A)
   x=0
   while (x < n) AND (A[x] ≠ '-'):      #discesa
      if z < A[x]:
         x = 2*x+1
      else:
         x = 2*x+2
   if (x < n)
         A[x] = z
   return
```

Costo computazionale: $O(h)$. Nota: $h = \Omega(\log n)$ e $h = O(n)$

Corso di laurea in Informatica
Algoritmi 1
A.A. 2025/2026

Esercizi per casa

SAPIENZA
UNIVERSITÀ DI ROMA

Esercizi (1)

- Scrivere lo pseudocodice (sia iter. che ric.) della funzione che calcola il minimo in un ABR.
- Scrivere lo pseudocodice (sia iter. che ric.) della funzione che calcola il massimo in un ABR.
- Scrivere lo pseudocodice (sia iter. che ric.) della funzione che calcola il predecessore di un valore dato in un ABR.
- Scrivere lo pseudocodice (sia iter. che ric.) della funzione che calcola il successore di un valore dato in un ABR.

Esercizi (2)

(esame del 3/4/2025)

Si consideri un albero binario di ricerca T in cui a ciascun nodo è associata una chiave numerica memorizzato tramite record e puntatori.

Scrivere un algoritmo efficiente che, dato in input il puntatore alla radice di T e due valori reali *a* e *b*, con $a<b$, restituisca *true* se e solo se TUTTE le chiavi di T sono comprese nell’intervallo *[a,b]*.