Corso di laurea in Matematica
Insegnamento di Informatica generale
Lezioni in modalità mista o a distanza

Dizionari: Alberi binari di ricerca

Giancarlo Bongiovanni
Ivano Salvo

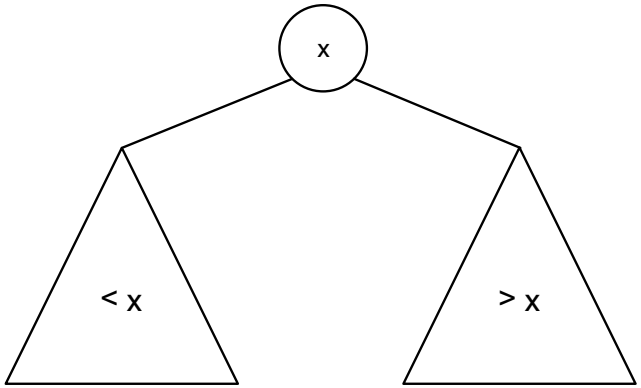
SAPIENZA
UNIVERSITÀ DI ROMA

Queste dispense sono state realizzate sulla base delle slide preparate da T. Calamoneri e G. Bongiovanni per il corso di Informatica Generale tenuto a distanza nell'A.A. 2019/20

ABR (1)

Un albero binario di ricerca (ABR) è un albero nel quale 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)

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, NULL altrimenti;
- Minimum(T)/Maximum(T): restituisce un puntatore all'elemento di minimo/massimo valore presente in *T*;
- Predecessor(T,p)/Successor(T,p): restituisce un puntatore all'elemento presente in *T* con il valore che precederebbe/seguirebbe il valore contenuto nel nodo puntato da *p* in una sequenza ordinata.

Operazioni di manipolazione:

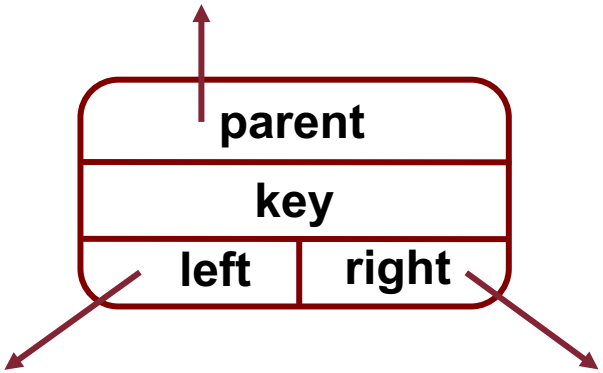
- Insert(T, k): inserisce un elemento di valore *k* in *T*;
- Delete(T, p): elimina da *T* l'elemento puntato da *p*.

ABR (3)

Come si vedrà, alcune delle operazioni sugli ABR (in particolare, la ricerca di predecessore o successore) richiedono di risalire verso la radice.

Al fine di implementare in modo efficiente tali risalite, nel seguito della trattazione considereremo i nodi di un ABR come record che contengono, oltre ai campi già noti, anche un campo ***parent*** che, in ogni nodo, punta al padre del nodo.

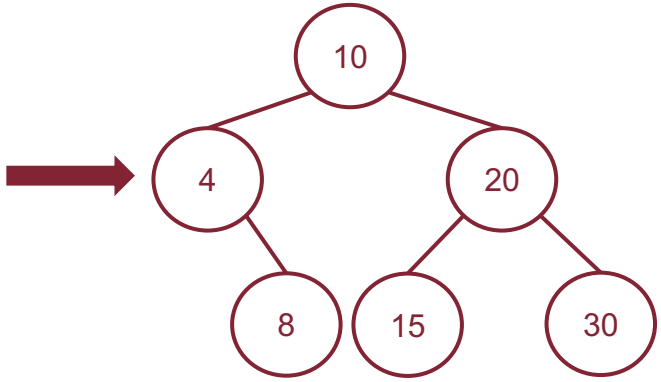
La radice avrà in tale campo il valore **`NULL`**.



ABR (4)

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.

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



ABR (5)

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

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:

$T(\text{costruzione ABR}) + T(\text{visita}) = T(\text{costruzione ABR}) + \Theta(n)$

Più avanti determineremo il costo della costruzione di un ABR.

ABR – ricerca (1)

Ricerca

Concettualmente simile alla ricerca binaria: si esegue una discesa dalla radice che viene guidata dai valori memorizzati nei nodi che si incontrano lungo il cammino.

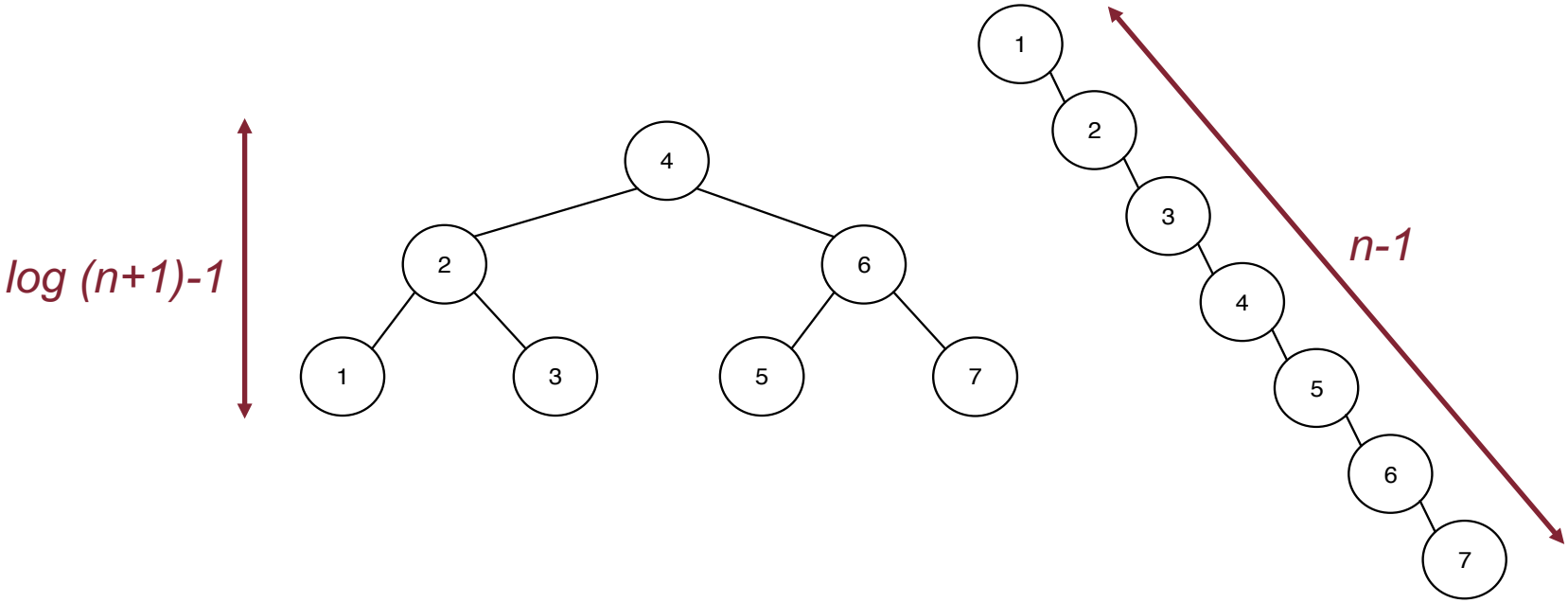
```
Funz.ABR_searchRic(p: punt.radice; k: chiave)
   if ((p = NULL) or (key[p] = k))
      return p
   if (k < key[p])
      return ABR_searchRic (left[p],k)
   else
      return ABR_searchRic (right[p],k)
```

Costo computazionale: O(h), h = altezza dell'albero.

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?

Problema: l'ABR non include fra le sue proprietà alcunché in relazione alla sua altezza.



ABR – ricerca (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***.

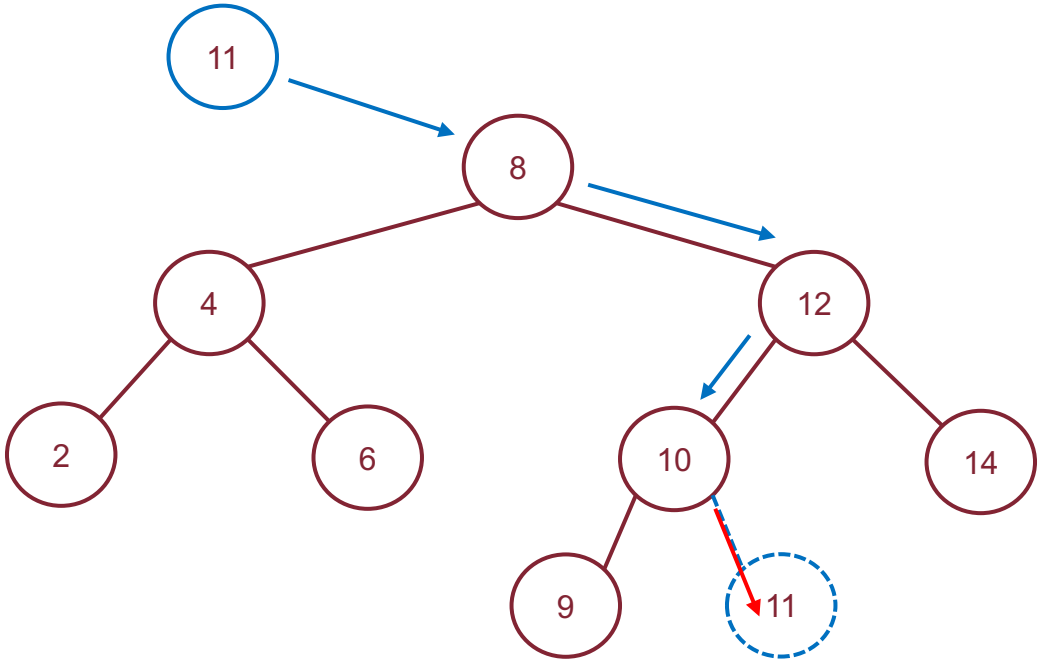
ABR – inserimento (1)

Inserimento

- Si esegue una discesa che, come nel caso della ricerca, viene guidata dai valori memorizzati nei nodi che si incontrano lungo il cammino.
- Quando si arriva al punto di voler proseguire la discesa verso un puntatore vuoto (NULL) allora, in quella posizione, si aggiunge un nuovo nodo contenente il valore da inserire.
- Il padre di tale nuovo nodo potrebbe essere una foglia (entrambi i suoi figli sono NULL), ma, più in generale, è un nodo a cui manca il figlio corrispondente alla direzione che si dovrebbe prendere per proseguire.

NOTA: inserire una chiave uguale ad una già contenuta nell'ABR la renderebbe poi inaccessibile alle ricerche, in quanto esse si fermerebbero su quella che era già contenuta nell'ABR.

ABR – inserimento (2)



NOTA: l'inserimento deve provvedere anche a sistemare il campo parent del nodo 11, che deve puntare al nodo 10.

ABR – inserimento (3)

```
Funz. ABR_insert(p:radice;z:nodo;par:parent)//tutti punt.

   if (p = NULL)
      parent[z] ← par
      return z
   if (key[p]=key[z])
      scrivi "Chiave già contenuta"; return p
   if key[p] < key[x]
      left[p] ← ABR_insert(left[p],z,p)
   else
      right[p] ← ABR_insert(right[p],z,p)
   return p
```

Chiamata iniziale: `p ← ABR_insert(p,z,NULL)`

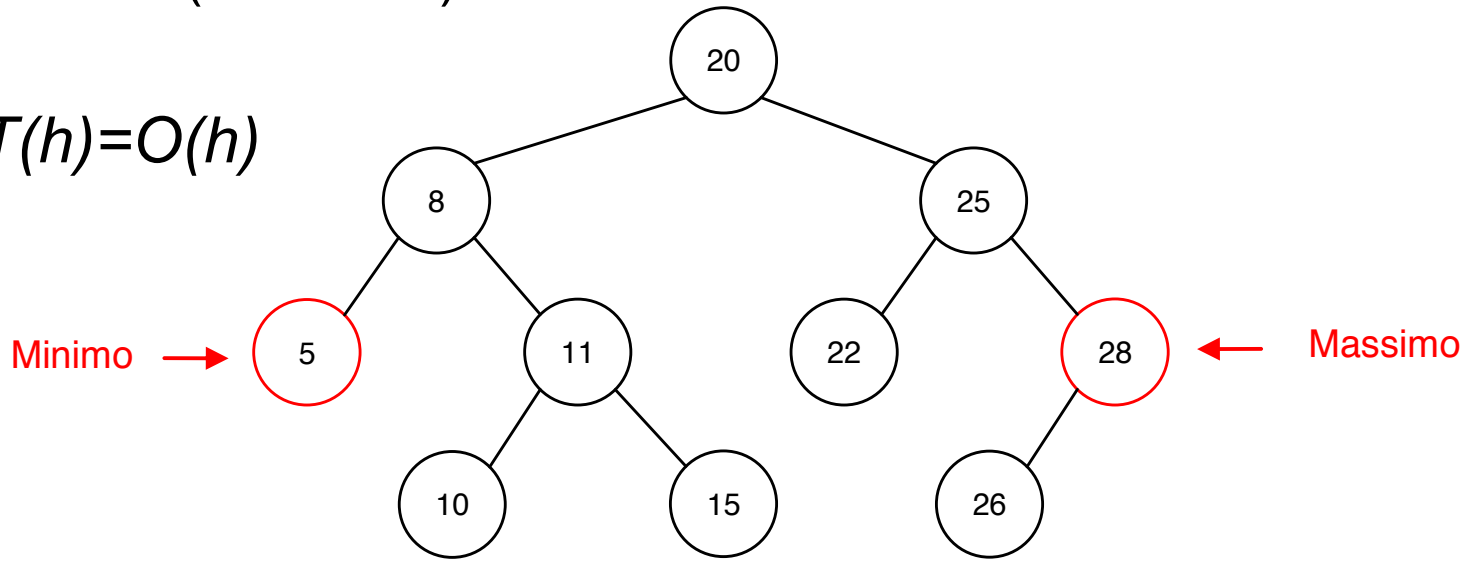
Costo computazionale (nel caso peggiore) $\Theta(h)$, in quanto:
$T(h) = T(h-1) + \Theta(1)$
$T(0) = \Theta(1)$

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

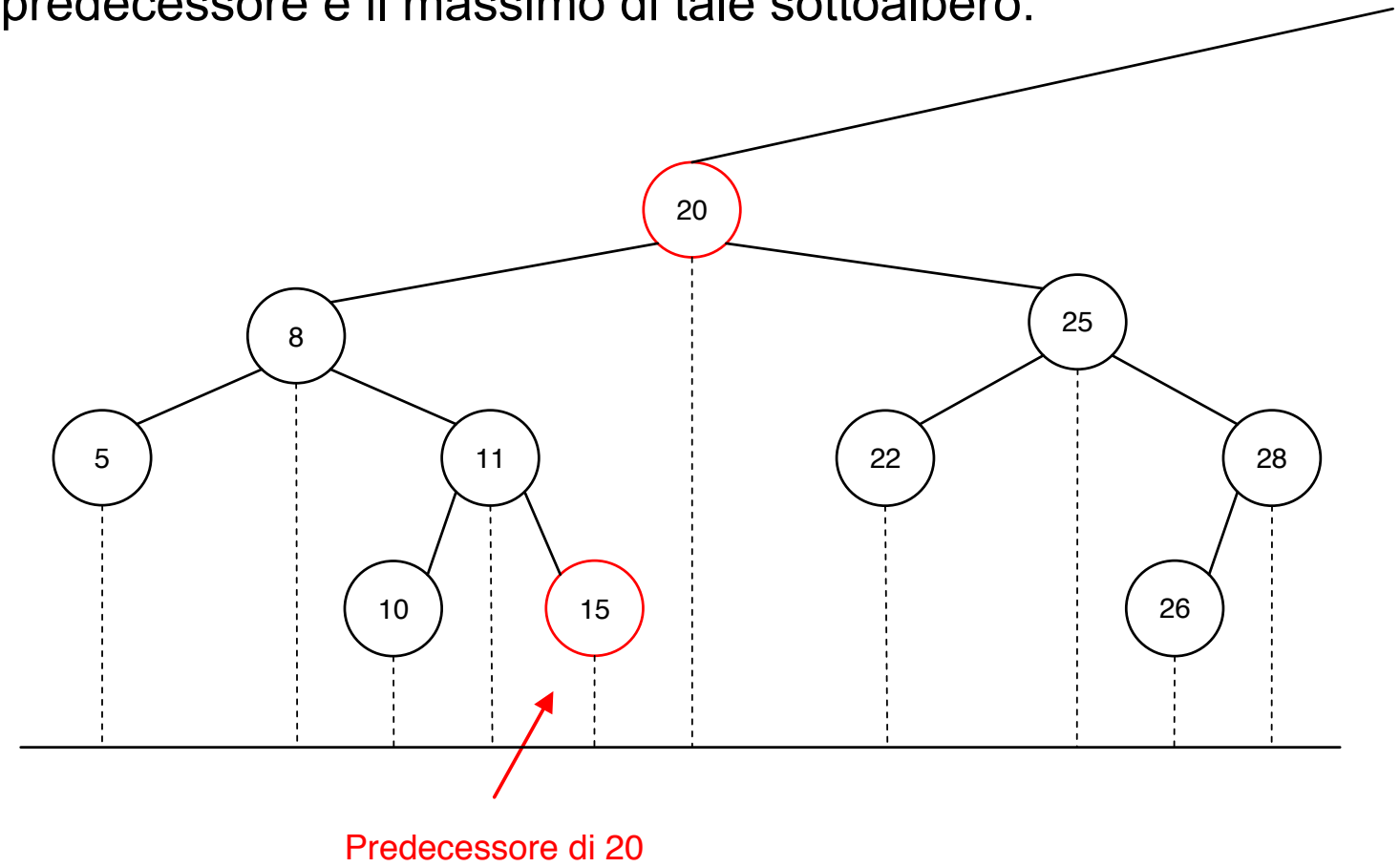
Ricordiamo che:

- per predecessore si intende il nodo dell'albero contenente 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:

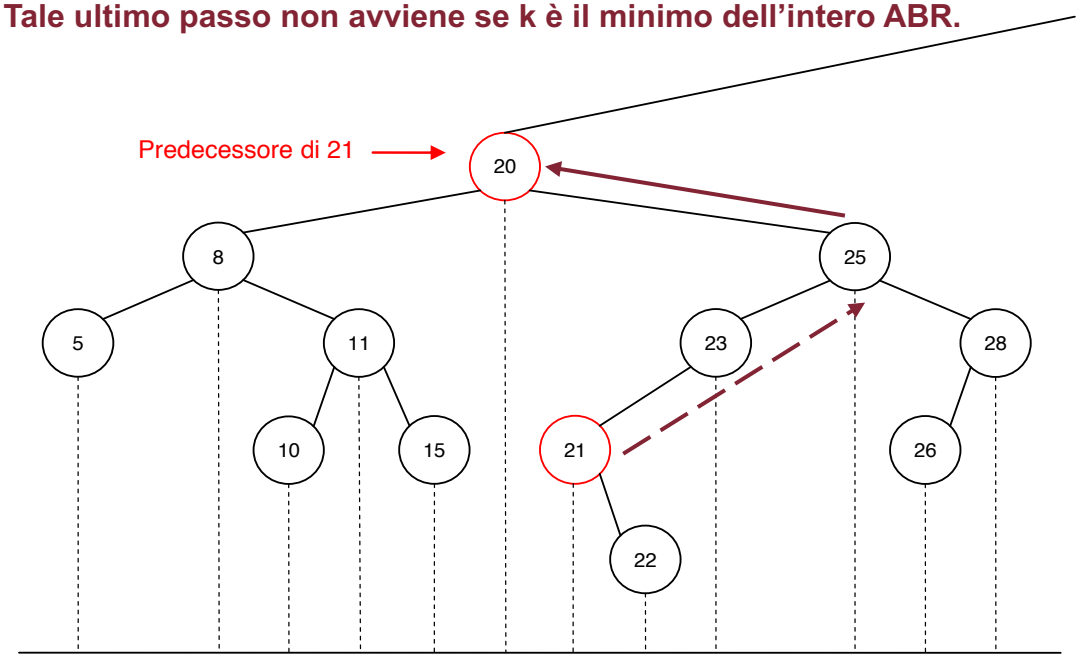


ABR – predecessore e successore (3)

Caso 2: se il nodo che contiene *k* **non** ha sottoalbero sinistro vuol dire che esso è il nodo più a sinistra di un certo sottoalbero, e quindi il minimo di tale sottoalbero. 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*.

Nota: **Tale ultimo passo non avviene se k è il minimo dell'intero ABR.**



ABR – predecessore e successore (4)

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

ABR – cancellazione (1)

Cancellazione:

Problema: Se la chiave da eliminare è contenuta in un nodo che ha entrambi i figli è necessario riaggiustare l'albero dopo l'eliminazione per evitare che si disconnetta, il che produrrebbe di conseguenza due ABR separati.

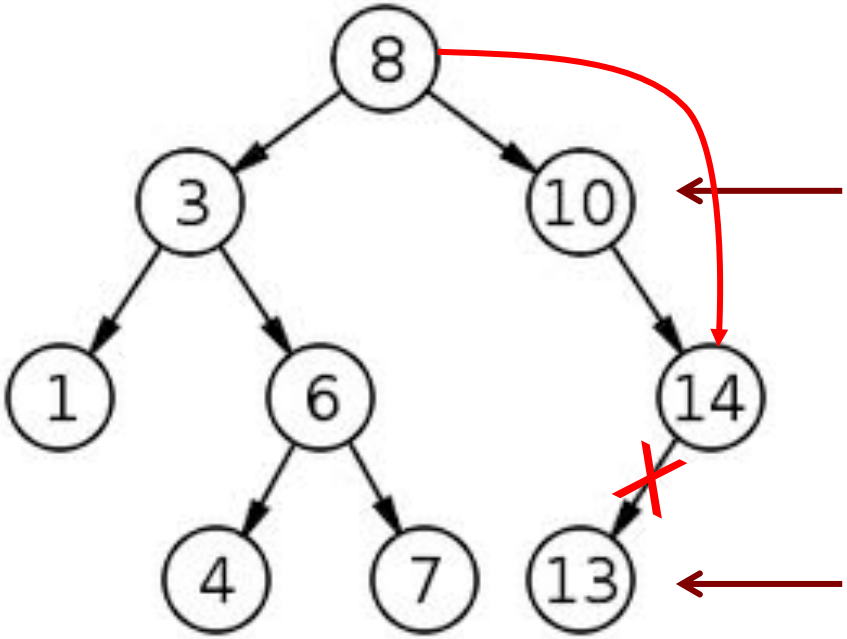
Riaggiustare l'albero significa trovare un nodo da collocare al posto del nodo che va eliminato, così da mantenere l'albero connesso e da garantire il mantenimento della proprietà fondamentale degli ABR.

Tale nodo può quindi essere solamente il predecessore o il successore del nodo da eliminare.

ABR – cancellazione (2)

Per eliminare un nodo in un ABR:

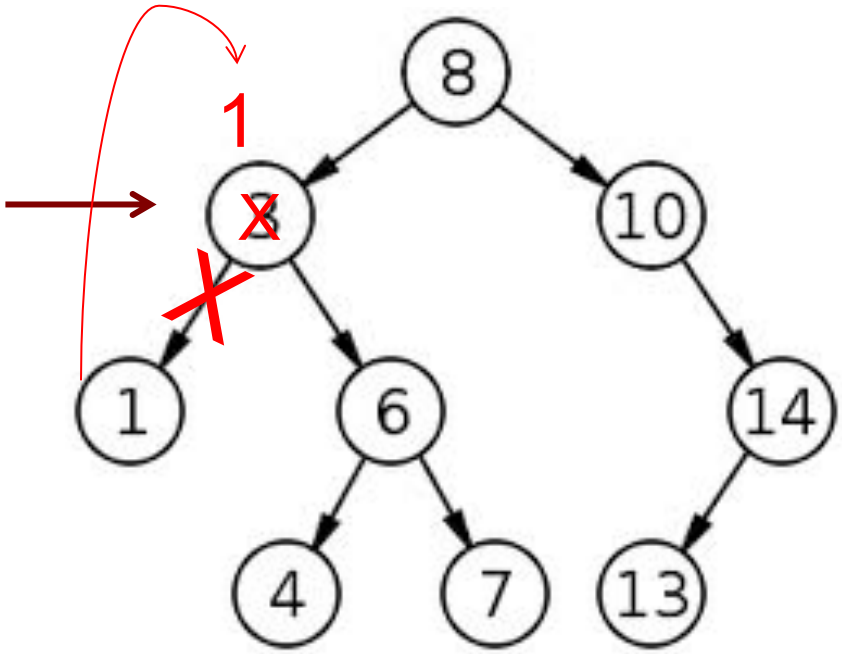
- **Caso 1**: se il nodo è una foglia lo si elimina, ponendo a *NULL* l'opportuno campo nel nodo padre;
- **Caso 2**: se il nodo ha un solo figlio lo si "cortocircuita", cioè si collegano direttamente fra loro suo padre e il suo unico figlio, indipendentemente che sia destro o sinistro;



ABR – cancellazione (3)

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 (operazione che ricade in uno dei due casi precedenti).

N.B. Per trovare il predecessore (o il successore) del nodo da cancellare siamo sicuramente nel *caso 1* dell'algoritmo per la ricerca del predecessore (o successore) perché il nodo da cancellare ha due figli.



Esercizio svolto (1-1)

Esercizio. 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.

… Sapendo che l'operazione di inserimento in un ABR ha un costo dell'ordine dell'altezza dell'albero si ha, nel caso peggiore, che 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 nel caso peggiore $O(h_1) = O(n_1)$ ed $O(1 + 2 + \ldots + n_2) = O(n_2^2)$, si ottiene che il costo di questo approccio è $O(n_1 n_2) + O(n_2^2) = O(n_1 n_2)$, essendo per ipotesi $n_1 > n_2$.

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. 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.

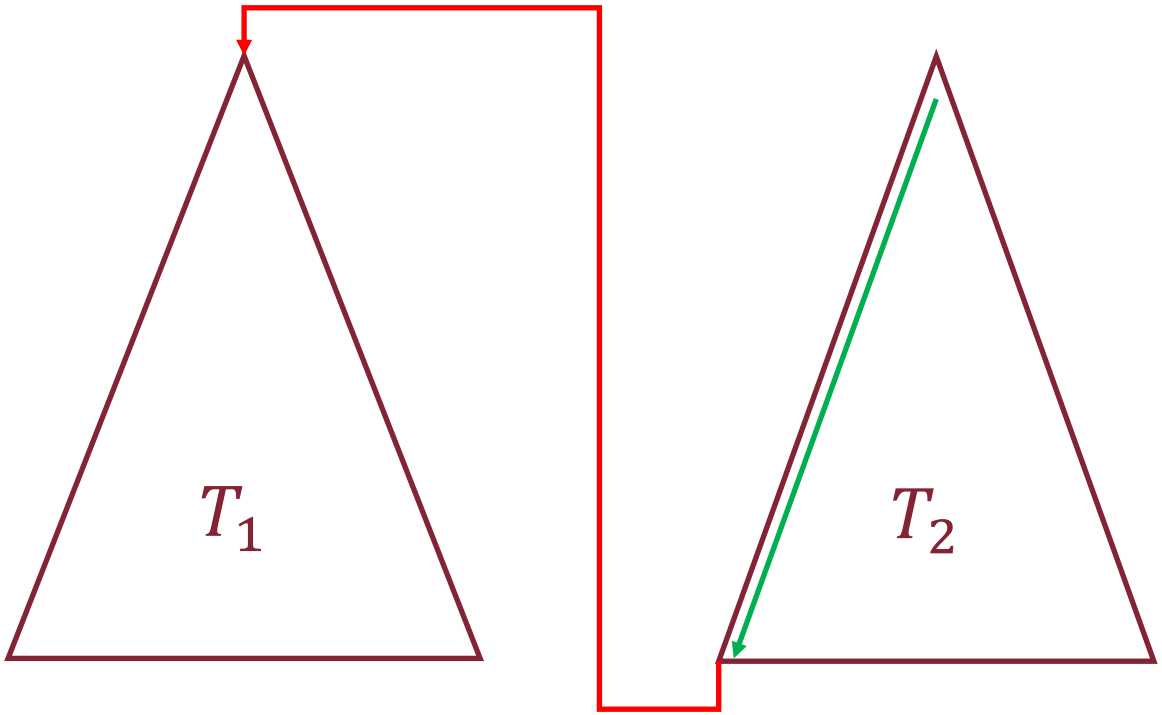
… “appendiamo” l’albero T_1 come figlio sinistro del minimo di T_2. Questo si può sempre fare facilmente perché il minimo di un ABR è il nodo più a sx che non possiede figlio sx, pertanto è sufficiente:

- settare un puntatore sulla radice di T_2,
- scendere verso il figlio sx finché esso esista
- giunti al nodo che non ha figlio sx (che è il minimo 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.



Esercizio svolto (3-1)

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

Assunzioni:

- Il vettore contiene *n* posizioni (indici da 1 a n);
- nel vettore si usa il simbolo '-' per denotare un nodo dell'albero mancante.

Esercizio svolto (3-2)

```
Funzione ABR_insert (A: vettore; n: intero;
                                z: valore da inserire)
   x ← 1
   while (x <= n) AND (A[x] ≠ '-') //discesa
      if (z = A[x])
         scrivi "Chiave già contenuta"; return
      if (z < A[x])
         x ← 2*x
      else
         x ← 2*x+1
   if (x <= n)
         A[x] ← z
   return
```

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

Corso di laurea in Matematica
Insegnamento di Informatica generale
Lezioni a distanza

ESERCIZI per casa

SAPIENZA
UNIVERSITÀ DI ROMA

Esercizi

- Scrivere lo pseudocodice della funzione che calcola il minimo in un ABR.
- Scrivere lo pseudocodice della funzione che calcola il massimo in un ABR.
- Scrivere lo pseudocodice della funzione che, in un ABR, trova il predecessore della chiave contenuta in un nodo di cui è dato il puntatore.
- Scrivere lo pseudocodice della funzione che, in un ABR, trova il successore della chiave contenuta in un nodo di cui è dato il puntatore.