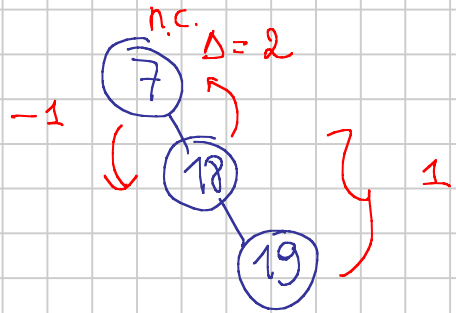
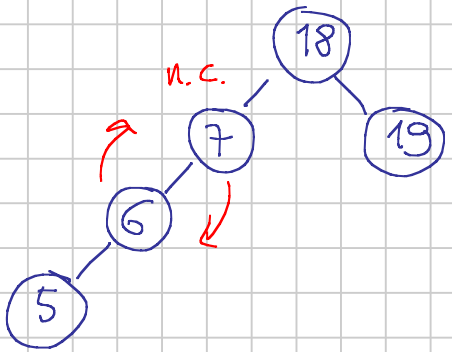
ESERCITAZIONE: dizionari e alberi

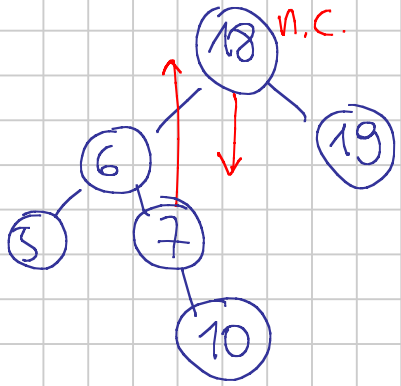
1) $S = \{7, 18, 19, 6, 5, 10, 14, 12\}$



DD(7)

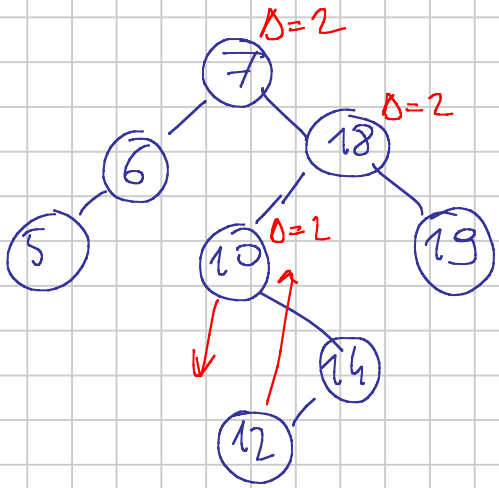


SS(7)

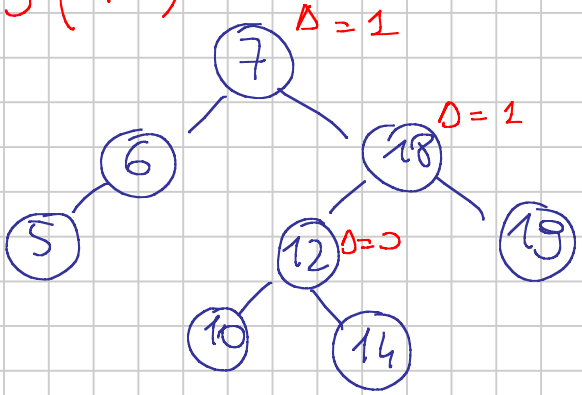


$S = \{7, 18, 19, 6, 5, 10, 14, 12\}$

SD(18)



DS(10)



OPERAZIONI DI DIZIONARIO: Costo

Titolo nota 26/02/2016

m = dim. Tabella

n = # elementi dizionario

$\alpha = \frac{n}{m}$

Struttura dati	Ricerca	Inserimento	Cancellazione (senza ricerca)	Min	Costruzione	Occupazione in Spazio
Tabella hash con liste di trabocco	$O(n)$ c.p. $O(1+\alpha)$ c.m.	$\Theta(1)$ cp $\Theta(1)$ cm	$\Theta(1)$ cp $\Theta(1)$ cm	$\Theta(\max\{n,m\}) = \Theta(m+n)$	$\Theta(n)$	$\Theta(m+n)$
Tabella hash ind. aperto	$O(n)$ c.p. $O\left(\frac{1}{1-\alpha}\right)$ c.m.	$O(n)$ cp $O\left(\frac{1}{1-\alpha}\right)$ cm	(logico) $\Theta(1)$ cp $\Theta(1)$ cm	$\Theta(m)$ cp $\Theta(m)$ cm	$O(n^2)$ cp $O\left(\frac{n}{1-\alpha}\right)$ cm	$\Theta(m)$
ABR $h = O(n)$	$O(h)$	$O(h)$	$O(h)$ (ricerca del successore)	$O(h)$	$\Theta(n \log n)$	$\Theta(n)$
AVL $h = \Theta(\log n)$	$O(h)$	$O(h)$	$O(h)$	$O(h)$	$\Theta(n \log n)$	$\Theta(n)$
Coda con priorità (heap)	$O(n)$	$O(\log n)$	$O(\log n)$ (sostituzione della foglia nella radice)	$\Theta(1)$ MIN HEAP $\Theta(n)$ MAX HEAP	$\Theta(n)$	$\Theta(n)$

(1) array S di n chiavi intere, non necessariamente distinte

trovare il numero r di chiavi distinte di S, con un'unica scansione.

Suggerimento: usare un dizionario D

```
ChiaviDistinte(S)        // dim S = n
D = nuovo Dizionario
ctr = 0
for i = 1 to n {
    if ( Search(D, S[i]) == NULL ) {
        ctr++;
        Insert(D, S[i]);
    }
}
return ctr;
```

D = array ordinato

n = dim S

r = # chiavi distinte

n ricerche in D $\rightarrow O(\log r)$

r inserimenti in D $\rightarrow O(r)$

$|D| = r$

$$T(n, r) = O(n \log r + r^2)$$

D = AVL

n ricerche $\rightarrow O(\log n)$

r inserimenti $\rightarrow O(\log r)$

$$T(n, r) = (n \log r + r \cdot \log r)$$

(2) S: array di n interi di valore non limitato, ma che possono assumere solo $\lfloor \log n \rfloor$ valori distinti.

Progettare un algoritmo di ordinamento di costo $\vartheta(n \log n)$

Usiamo un AVL

ogni nodo u contiene i campi

- $u.key$
- $u.occ$ (# occorrenze della chiave in S)
- $u.p$
- $u.left$
- $u.right$

```
Ordina(S)
T = nuovo albero AVL
for i = 1 to n {
    u = Search(T.root, S[i]);
    if (u ≠ NIL) u.occ++;
    else {
        v = nuovo nodo();
        v.left = NIL;
        v.right = NIL;
        v.key = S[i];
        v.occ = 1;
        Insert-AVL(T.root, v);
    }
}
VisitaSimmetrica(T.root, S, 1);
```

n ricerche e $\lfloor \log n \rfloor$ inserimenti in un AVL T.

T contiene $\lfloor \log n \rfloor$ nodi

$\Rightarrow h = O(\log \log n)$

visita simmetrica

$$T(n) = O(n \log\log n + \log n * \log\log n + n)$$

$$= O(n \log\log n) = o(n \log n)$$

```
VisitaSimmetrica(u, S, i)   // i = posizione in cui si può scrivere su S
if (u == NIL) return i;
k = VisitaSimmetrica(u.left, S, i)
for j = 1 to u.occ {
    S[k] = u.key;
    k++;
}
// k è la prossima cella libera su cui scrivere
return VisitaSimmetrica(u.right, S, k);
```

$T(n) = \Theta(n)$

(3) T = ABR

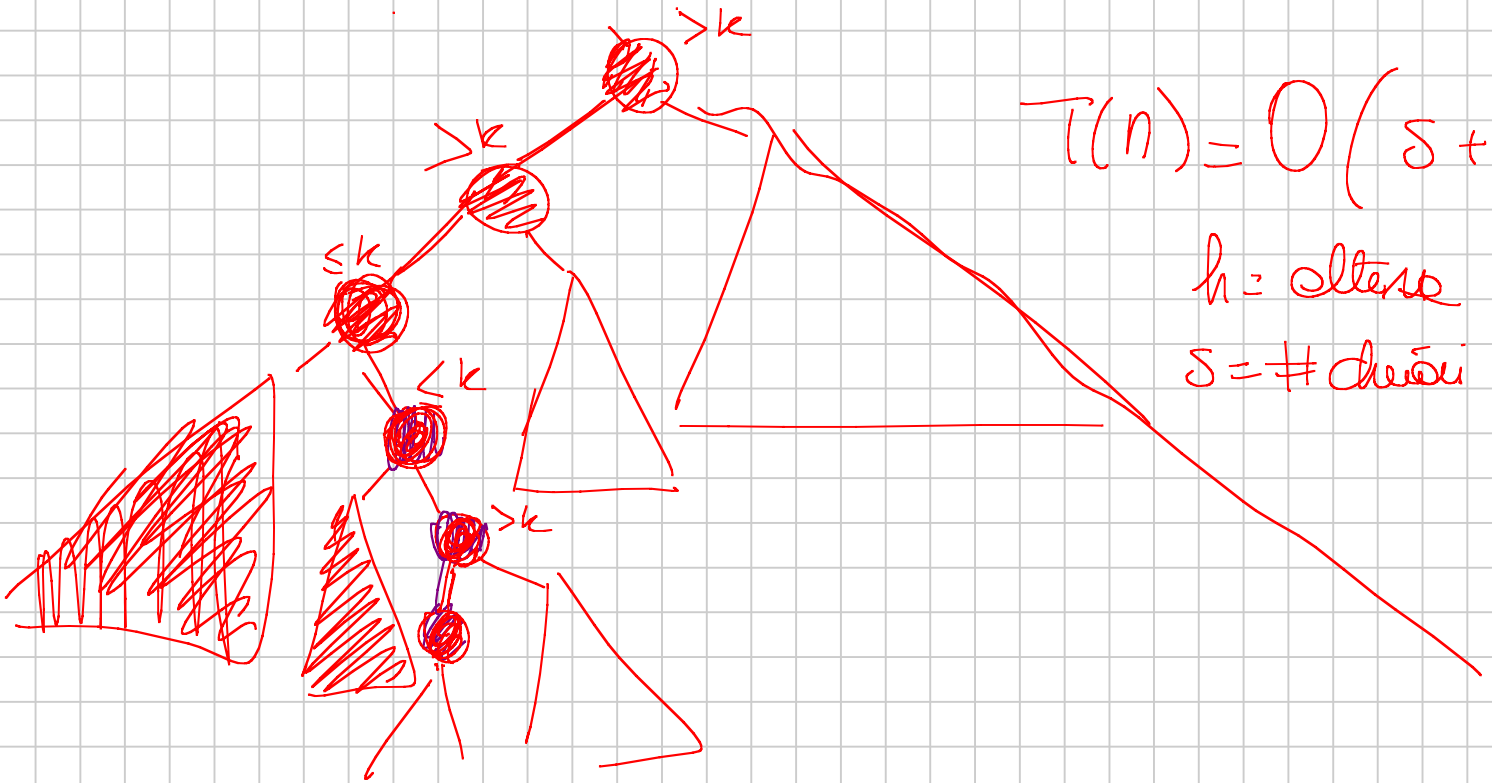
Progettare un algoritmo che restituisca il numero di elementi di T di chiave $\leq k$.

$T(n) = O(n)$

```
COUNTLE(u, k)
if (u == NIL) return 0;
if (u.key > k) return COUNTLE(u.left, k);
else return 1 + COUNTLE(u.left, k) + COUNTLE(u.right, k)
```

s = # chiavi $\leq k$

$S = \Theta(n)$ visita lineare T.



$T(n) = O(s + h)$

h = altezza

s = # chiavi $< k$.

ESERCIZIO 4

T = albero binario completo (ogni nodo ha 2 figli, eccetto le foglie)

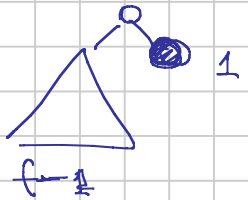
v nodo di T

$$Imbalance(v) = | \#\text{foglie del sottoalbero sx di } v - \#\text{foglie sottoalbero dx} |$$

$$Imbalance(T) = \max_{v \in T} Imbalance(v)$$

(1) Valore massimo $Imbalance(T)$, dove T è un ABC di dato n.

$\#\text{foglie } T = f$



$Imbalance(T) = (f-1) - 1 = f - 2$

$$f = \frac{n+1}{2} = \left\lceil \frac{n}{2} \right\rceil$$

(si dimostra per induzione)

$$\text{Imbalance}(T) \leq \frac{n+1}{2} - 2 = \frac{n-3}{2}.$$

(2) per esercizio

(3) Algoritmo per calcolare Imbalance(T)

Imbalance(u) // due parametri di uscita < maxImb, #foglie >
dell'albero radicato in u

```
if (u==NIL) return <0, 0>;
if (u.left==NIL && u.right==NIL) return <0, 1>
<ImbSx, fSx> = Imbalance(u.left);
<ImbDx, fDx> = Imbalance(u.right);
f = fSx + fDx;
imb = max{ImbSx, ImbDx, |fSx - fDx|}
return <imb, f>
```

$T(n) = \Theta(n)$