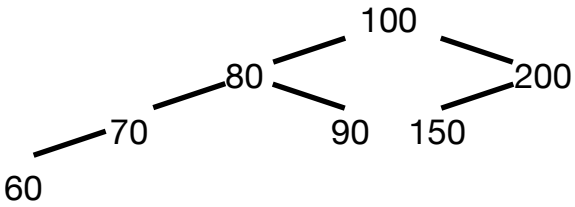
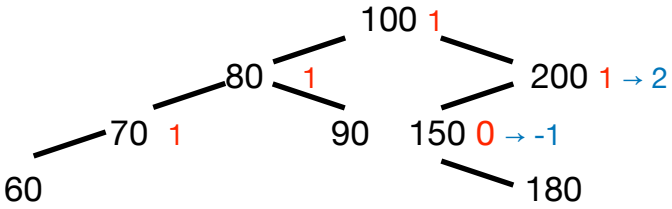
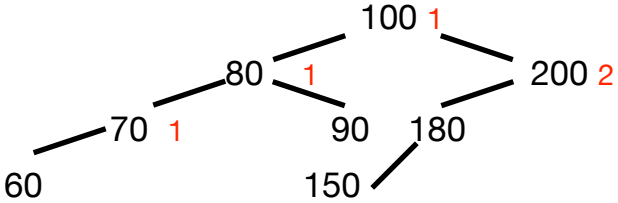
Es. 1. Dato l' albero di Fibonacci di altezza 3 qui sotto disegnato si inserisca 180 e si ribilanci l'albero. Ogni rotazione deve essere specificata indicando il nodo attorno al quale si esegue la rotazione, l'albero risultante della rotazione e le variazioni del fattore di bilanciamento



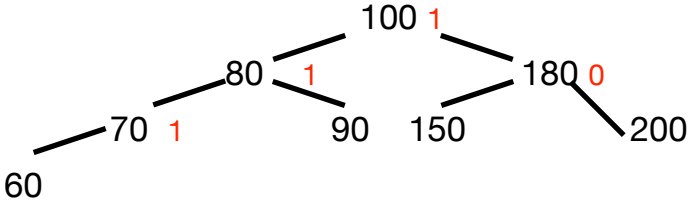
Sol. evidenziamo gli fb di ogni nodo in rosso e la variazione dovuta all'inserimento in blu:



Serve una doppia rotazione: una a sinistra su 150 e una a destra su 200.
Dopo la prima rotazione:



Dopo la seconda rotazione:



Es.2. Si imposti la relazione di ricorrenza che definisce il tempo di esecuzione della seguente funzione e la si risolva usando il metodo della sostituzione. Si commentino opportunamente i passaggi del calcolo, si disegni l'albero

della ricorsione e come si giunge alla previsione sull'andamento del tempo di calcolo, si imposti l'induzione con chiarezza, sia nello scrivere quanto si vuole dimostrare sia nel formulare l'ipotesi induttiva.

```
test (intero n)
        if n ≤ 1 then return 1
        k = n*n
            while k ≥ 1 do k = k-2
        return k + 2*test(n/4)
```

Sol.
La funzione è chiamata una sola volta su n/4 e le istruzioni al di fuori della chiamata sono eseguite in tempo quadratico, esattamente il ciclo è eseguito $n^2/2$ volte. Quindi la relazione è
$T(n) = T(n/4) + \Theta(n^2)$.
Si deve quindi risolvere la ricorrenza
$T(n) = T(n/4) + cn^2$ se $n \geq 1$
$T(n) = d$ altrimenti

L'albero della ricorsione per fare una previsione sull'andamento è qui abbozzato:

cn^2

$c(n/4)^2$

$c(n/4^2)^2$

Ponendo $n = 4^h$, l'albero ha altezza h, ogni nodo ha un figlio e al livello i ogni nodo è etichettato con $c(n/4^i)^2$.
$T(n) = d + \sum_{i=0\ldots h-1} (n^2/2^{4i})c = d + cn^2 \sum_{i=0\ldots h-1} 1/2^{4i} \leq d + cn^2 \sum_{i=0\ldots\infty} 1/2^{4i} =$
$O(n^2)$, perchè la serie di potenze converge a un a costante.

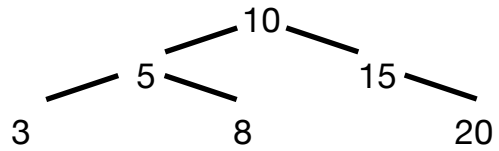
Ora verifichiamo la previsione per induzione.
Faremo vedere che esistono due costanti positive k e n' tali che $T(n) \leq kn^2$, per ogni $n \geq n'$.
Sia vero per ipotesi induttiva per ogni $m < n$, consideriamo T(n):
$T(n) = T(n/4) + cn^2 \leq$
$k (n/4)^2 + cn^2 = k/16n^2 + cn^2$
Vediamo se troviamo due valori per k e n' tali che $k/16n^2 + cn^2 \leq kn^2$,

$k/16n^2 + cn^2 \leq kn^2 \Leftrightarrow kn^2 + 16cn^2 \leq 16kn^2 \leq 0 \Leftrightarrow 16cn^2 \leq 15kn^2$ quest'ultima disuguaglianza è vera per $k \geq 2c$ e per ogni $n \geq 1$

Considerando il caso base abbiamo che $T(1) = d$ e $T(2) = d+4c$ e $T(2) \leq 4k$ è vero prendendo $k = d+2c$ e $n' = 1$.
Poiché il termine additivo è quadratico nella relazione possiamo dire che $T(n) = \Omega(n^2)$ e concludiamo che $T(n) = \Theta(n^2)$

Es. 3. Dato un ABR T e due sue chiavi x e y, si progetti un algoritmo che dà in output il numero di nodi che è necessario attraversare per andare da x a y, lungo un cammino padri-figli, se c'è e 0 altrimenti. Si descriva a parole l'idea algoritmica, si analizzi il tempo di esecuzione asintotico, specificando caso peggiore e caso migliore, e si produca lo pseudocodice.

Esempio: se x = 10 e y = 3 la risposta è 3. Se x= 5 e y= 10 allora la risposta è 0. Se x = y la risposta è 1



P.S. Soluzioni ridotte al solo pseudocodice non saranno corrette.

Sol.
Si risolve il problema cercando prima x nell'albero e poi usando una funzione di ricerca modificata in modo da contare i nodi che si attraversano per cercare y nel sotto albero radicato in x. Il tempo asintotico nel caso peggiore è $\Theta(h)$, nel caso x fosse la radice e y una foglia di profondità massima, e nel caso migliore è $\Theta(1)$ quando per esempio x=y è la radice dell'albero.

Pseudocodice:

La funzione è

```
PadriFigli(T,x,y)
input: un puntatore T e due interi x e y
prec: T è ABR, x e y sono due chiavi di T
output: il numero dei nodi attraversati in un cammino che da y porta a x
scendendo da padre a figlio, se c'è, 0 altrimenti.
p = Tree-Search(T, x)
return CalcolaNodi(p,y)
```

nella quale si fa uso delle seguenti funzioni:

Tree-Search(T, k)

Input: un puntatore T e una chiave k

prec: T è un ABR

output: il puntatore (riferimento) al nodo di chiave k, se presente, nil altrimenti

CalcolaNodi(x,y)

Input: un puntatore x e un intero y

prec: x un nodo in un ABR e y una chiave presente nell'ABR

output: il numero dei nodi attraversati in un cammino che da y porta a x scendendo da padre a figlio se c'è, 0 altrimenti.

```
if x.key == y then return 1

if x è una foglia then return 0 // qui x.key ≠ y
if y < x.key then c = CalcolaNodi(x.left, y)
             else c = CalcolaNodi(x.right,y)
if c == 0 then  return c else return c++
```