Domanda 5 Risolvere la ricorrenza

$$T(n) = \begin{cases} 3 & \text{se } n = 0 \\ T(n-1) + 2 & \text{se } n > 0 \end{cases}$$

utilizzando il metodo di sostituzione per determinare una soluzione esatta (non asintotica).

Soluzione: Mostriamo per induzione che $T(n) = an + b$, determinando a e b. Per $n = 0$ si ottiene $T(n) = 3 = b$, quindi $b = 3$. Nel caso induttivo, si ottiene

$$T(n+1) = T(n) + 2 = an + b + 2$$

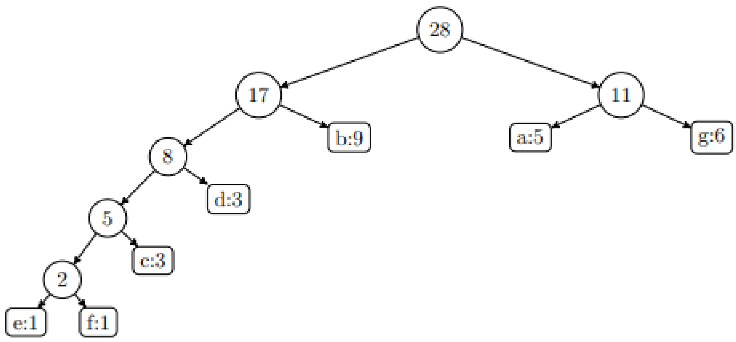
Affinché $an + b + 2 = a(n+1) + b$ occorre che $a = 2$. Quindi, $T(n) = 2n + 3$.

Domanda 40 Indicare il codice prefisso ottenuto utilizzando l'algoritmo di Huffmann per l'alfabeto $\{a, b, c, d, e, f, g\}$, supponendo che ogni simbolo appaia con le seguenti frequenze.

a	b	c	d	e	f	g
5	9	3	3	1	1	6

Spiegare il processo di costruzione del codice.

Soluzione:



Domanda 36 Realizzare una funzione *pred*(x) che dato in input un nodo x, di un albero binario di ricerca T, restituisce il predecessore di x (oppure nil, se il predecessore non esiste). Come a lezione, supporre che ogni nodo abbia i campi *x.left*, *x.right*, *x.p*, *x.key*.

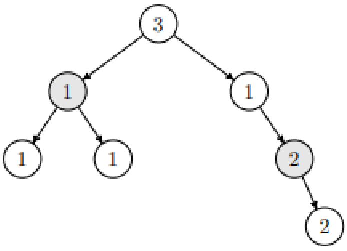
Soluzione:

```
pred(x)
    if x.left <> nil
        return max (x.left)
      else
        y = x.p
        while (y <> nil) and (x == y.left)
           x=y
           y=y.p
        return y

max (x)
    while x.right <> nil
          x = x.right
    return x
```

Esercizio 10 Un nodo x di un albero binario T si dice *fair* se la somma delle chiavi nel cammino che conduce dalla radice dell'albero al nodo x (escluso) coincide con la somma delle chiavi nel sottoalbero di radice x (con x incluso). Realizzare un algoritmo ricorsivo *printFair(T)* che dato un albero T stampa tutti i suoi nodi fair. Supporre che ogni nodo abbia i campi *x.left*, *x.right*, *x.p*, *x.key*. Valutare la complessità dell'algoritmo.

Un esempio: i nodi grigi sono fair



Soluzione: L'algoritmo può essere il seguente:

```
printFair(x,path)       // x = node of the tree
                        // path=sum of the keys in the path from the root to x


                         // Action: print the fair nodes and
                        //         returns the sum of the keys in the subtree
   if (x == nil)
      return 0

   left   = printFair(x.l, path + x.key)
   right  = printFair(x.r, path + x.key)
   sumTree = left + right + x.key
   if (path == sumTree)
      print x
   return sumTree
```

e viene chiamato come `printFair(T.root, 0)`.

Si tratta di una visita, quindi con costo $O(n)$ (più precisamente ottenibile con il master theorem come soluzione della ricorrenza $T(n) = 2T(n/2) + c$.

Esercizio 20 Si ricordi che data una sequenza $X = x_1 \dots x_k$, si indica con X_i il prefisso $x_1 \dots x_i$. Una sottosequenza di X è $x_{i_1} \dots x_{i_h}$ con $1 \leq i_1 < i_2 < \dots i_h \leq k$, ovvero è una sequenza ottenuta da X eliminando alcuni elementi. Quando Y è sottosequenza di X si scrive $Y \sqsubseteq X$.

Realizzare un algoritmo che, date due sequenze $X = x_1 \dots x_k$ e $Y = y_1 \dots y_h$ determina una *shortest common supersequence* (SCS) ovvero una sequenza Z, di lunghezza minima, tale che $X \sqsubseteq Z$ e $Y \sqsubseteq Z$. Ad esempio per $X = abf$ e $Y = afgj$ una SCS è $abfgj$.

i. Dare una caratterizzazione ricorsiva della lunghezza $l_{i,j}$ di una SCS di X_i e Y_j e dedurne un algoritmo;

ii. trasformare l'algoritmo in modo che fornisca una SCS di X e Y;

iii. valutare la complessità dell'algoritmo.

Soluzione: La caratterizzazione ricorsiva è:

$$l_{i,j} = \begin{cases} i + j & \text{if } i = 0 \text{ or } j = 0 \\ l_{i-1,j-1} + 1 & \text{if } i, j > 0 \text{ e } x_i = y_j \\ \min\{l_{i,j-1}, l_{i-1,j}\} + 1 & \text{if } i, j > 0 \text{ e } x_i \neq y_j \end{cases}$$

Ne segue l'algoritmo che riceve in input le stringhe, nella forma di array di caratteri $X[1, k]$, $Y[1, h]$ e usa una matrice $L[0..k, 0..h]$ dove $L[i, j]$ rappresenta la lunghezza della minima SCS di X_i e Y_j.

```
SCSlen (X,Y,k,h)
  for i=0 to k
    L[i,0] = i
  for j=1 to h
    L[0,j] = j

  for i=1 to k
    for j = 1 to h
      if (X[i] = X[j])
         L[i,j] = L[i-1,j-1] + 1
      else
         L[i,j] = min (L[i,j-1], L[i-1,j]) + 1

  return L[k,h]
```

Se vogliamo anche la SCS occorre tener conto delle scelte effettuate per raggiungere l'ottimo. Lo facciamo mediante un array $P[1..n, 1..n]$

```
SCS-data (X,Y,k,h)
  for i=0 to k
    L[i,0] = i
  for j=1 to h
    L[0,j] = j


  for i=1 to k
    for j = 1 to h
      if (X[i] = X[j])
         L[i,j] = L[i-1,j-1] + 1
         P[i,j] = "xy"
      else
         if L[i,j-1] <= L[i-1,j]
            L[i,j] =  L[i,j-1] + 1
             P[i,j] =  y
         else
            L[i,j] =  L[i-1,j] + 1
            P[i,j] =  x


   return L, P


 SCS (X,Y,i,j,P)
   if i==0
      print Y[j]
   elseif j==0
      print X[i]
   else
     if P[i,j] = xy
        SCS(X,Y,i-1,j-1,P)
        print X[i]

     elseif P[i,j] = x
        SCS(X,Y,i-1,j,P)
        print X[i]

     else
        SCS(X,Y,i,j-1,P)
        print Y[j]
```

La complessità è $\Theta(hk)$.