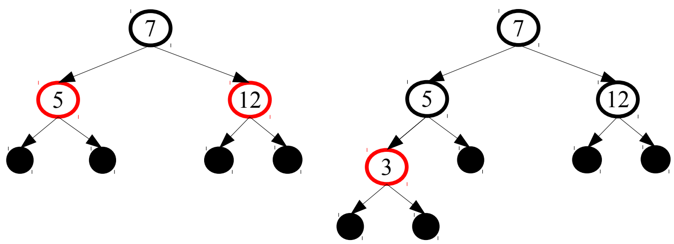
Algoritmi e Strutture Dati
07/01/2014

Esercizio 1

La complessità dell'algoritmo è $O(n \log^2 n)$, in quanto si tratta di tre cicli annidati, dove il primo è eseguito n volte, mentre i due cicli interni sono eseguiti ognuno $\log n$ volte.

Esercizio 2

Si consideri l'albero della figura a sinistra qui sotto e si consideri l'inserimento della chiave 3. La chiave viene inserita a sinistra del 5, colorata di rosso. Ma poiché 5 è colorato di rosso, ci ritroviamo nel caso (3) della procedura di inserimento. Il padre e zio vengono colorati di nero, e il problema si sposta alla radice, che viene colorata di rosso. Poichè la radice ora è rossa, ma non ha padri, ci ritroviamo nel caso 1 e la radice viene colorata di rosso. L'altezza nera è ora pari a 2.



Esercizio 3

Parte 1: La prima parte si risolve modificando una visita di Erdos, evitando di visitare stanze in cui ci siano mostri.

```
erdos(GRAPH G, NODE s, integer[] erdős, NODE[] p, boolean[] M)
    QUEUE S ← Queue()
    S.enqueue(s)
    foreach u ∈ G.V() − {s} do erdős[u] ← ∞
    erdős[s] ← 0
    while not S.isEmpty() do
        NODE u ← S.dequeue()
        foreach v ∈ G.adj(u) do                          % Esamina l'arco (u, v)
            if erdős[v] = ∞ and not M[u] then            % Il nodo u non è già stato scoperto e non c'è mostro
                erdős[v] ← erdős[u] + 1
                S.enqueue(v)
```

Il valore cercato si trova in erdos$[d]$. Sarebbe possibile migliorare ulteriormente l'algoritmo bloccando la ricerca quando si raggiunge d; in ogni caso, la ricerca nel caso pessimo richiede $O(m + n)$.

Parte 2: In questo caso, invece, è possibile utilizzare l'algoritmo di Dijsktra definendo i pesi in modo adeguato. Il peso degli archi è pari a 1 se il nodo di destinazione contiene un mostro, è pari a 0 altrimenti. La complessità risultante è pari a $O(n^2)$ o $O(m \log n)$, a seconda della struttura di dati utilizzata.

Esercizio 4

E' sufficiente utilizzare una soluzione greedy che ordina i libri per altezza (costo $O(n \log n)$) e poi piazza i libri in ordine crescente, utilizzando uno scaffale fino a quando questo contiene ancora libri.

```
integer altezza(integer[] x, integer[] y, integer N, integer L)
  sortY(x, y, N)
  integer altezzza ← 0
  integer scaffale ← 0
  integer maxAltezza ← 0
  for i ← 1 to n do
      if scaffale + x[i] ≤ L then
          { Aggiungi libro a scaffale corrente }
          scaffale ← scaffale + x[i]
          maxAltezza ← y[i]
      else
          { Aggiungi libro a nuovo scaffale }
          scaffale ← x[i]
          altezza ← altezza + maxAltezza
          maxAltezza ← y[i]
  { Aggiungi ultimo scaffale } altezza ← altezza + maxAltezza
  return altezza
```