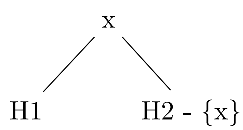
Fusione di Max-Heap

Realizzare una procedura `Fusion(H1,H2)` per fondere due heap `H1` e `H2`, il primo completo, il secondo quasi completo, della stessa altezza, che soddisfano la proprietà di max-heap, ottenendo un unico max-heap. Cercare una soluzione di costo $O(\log n)$ dove n è il numero di elementi totali dei due alberi.

Soluzione Assumiamo che `H1` e `H2` siano dati come strutture linked. Allora si può prendere l'ultima foglia x di `H2`, e usarla come radice comune dei due heap, che ha `H1` come figlio sinistro e quel che resta di `H2` come figlio destro.



A questo punto, la proprietà di max-heap è potenzialmente violata solo dalla radice e quindi può essere ripristinata usando `MaxHeapify`.

```
fusion(H1, H2)                // assume H1, H2 non vuoti
   p = numToParent(H2, H2.size)   // determina il parent dell'ultima foglia di H2

   if p == nil                    // la foglia e' anche la radice
                                  // H2 contiene solo la radice
      x = H2.root
      H2.root = nil
   else
      if (H.size mod 2 == 0)      // se il bit meno significativo e' 0
         x = p.left               // la foglia e' figlio sinistro
         p.left = nil
      else
         x = p.right = p          // altrimenti e' figlio destro
         p.right = nil

   H2.size--                      // ricorda che e' stata rimossa una foglia

   // crea un nuovo heap con radice x
   H.root = x
   H.size = H1.size + H2.size + 1
   x.left = T1.root
   x.right = T2.root

   MaxHeapify(x)                  // ripristina la proprieta' di max-heap

   return H
```

La complessità è data dalla somma della complessità di `numToParent(H, H.size)` e di `maxHeapify(x)` (il resto ha costo costante). Risulta dunque essere $O(\log n)$.

Si noti che se invece `H1` e `H2` fossero stati dati come array, la creazione di un nuovo array che contenga gli elementi di `H1` e `H2` ha almeno costo lineare. Si può provare che il costo resta almeno lineare anche se i due max-heap sono allocati in spazi consecutivi. L'idea dell'argomento è la seguente. Per fissare le idee, supponiamo che `H1` e `H2` siano memorizzati in un array `A[1...p...n` con `A[1...p]` che contiene `H1` e `A[p+1...n]` che contiene `H2`. Le osservazioni fondamentali sono che:

- gli elementi di `H2` sono le foglie di `A` visto come heap, quindi hanno elementi di `H1` come antenati;
- se si assume che ogni elemento di `H1` sia strettamente minore di ogni elemento di `H2`, affinché un elemento di `H2` non venga spostato nella costruzione del max-heap complessivo, occorre che tutti i suoi antenati, che erano elementi di `H1`, vengano spostati;
- l'osservazione precedente implica che almeno metà degli elementi di `H1` devono essere spostati, cosa che richiede un numero di operazioni lineari in n.

Si osservi dunque che quando si utilizzi la rappresentazione dei max-heap come array non si può utilizzare la proprietà di max-heap di `H1` e `H2` in modo significativo, ovvero il costo resta analogo a quello che avrei semplicemente applicando `BuildMaxHeap` all'intero array `A[1...n]`.