ESERCIZIO DI ASD DEL 15 DICEMBRE 2008

Da Vettore Ordinato a Binary Search Tree

Algoritmo. Per costruire un albero il più possibile bilanciato, utilizziamo l'elemento centrale del vettore come radice dell'albero e procediamo in maniera ricorsiva a sinistra e destra. Come già visto in altre procedure ricorsive su vettori, dobbiamo passare alla procedura gli indici che delimitano la porzione di vettore su cui stiamo lavorando. Inoltre, per poter sistemare i puntatori *parent* passeremo alla procedura anche il nodo che ha "generato" la chiamata ricorsiva. La procedura ritornerà sempre come risultato il nuovo nodo creato.

Algorithm 1 ARRAY_TO_BST(A)

```
1: x ←REC_ARRAY_TO_BST(A, 1, length[A], NIL)
2: root[T] ← x
3: return T
```

Algorithm 2 REC_ARRAY_TO_BST(A, i, j, y)

```
1: if i ≤ j then
2:     k ← (i + j)/2
3:     x ← new_node()
4:     key[x] ← A[k]
5:     parent[x] ← y
6:     left[x] ←REC_ARRAY_TO_BST(A, i, k − 1, x)
7:     right[x] ←REC_ARRAY_TO_BST(A, k + 1, j, x)
8: else
9:     x ← NIL
10: end if
11: return x
```

Correttezza. Dimostriamo prima di tutto che viene costruito un binary search tree.

Lemma 1. *Se il vettore A è ordinato, la procedura* REC_ARRAY_TO_BST(A, i, j, y) *termina sempre restituendo un nodo x che è radice di un binary search tree contenente tutti e soli gli elementi di $A[i..j]$.*

Dimostrazione. Procediamo per induzione sul numero di elementi contenuti in $A[i..j]$, ovvero per induzione su $n = j - i + 1$.
BASE. $n = 0$. In questo caso deve essere $j < i$. Quindi ad x viene assegnato valore NIL, che è radice di un BST che non contiene elementi.
PASSO.

Date: 15 Dicembre 2008.

HpInd) Se $n < m$ ed A è ordinato, allora la procedura REC_ARRAY_TO_BST(A, i, j, y) termina sempre restituendo un nodo x che è radice di un binary search tree contenente tutti e soli gli elementi di $A[i..j]$.

Ts) Se $n = m$ ed A è ordinato, allora la procedura REC_ARRAY_TO_BST(A, i, j, y) termina sempre restituendo un nodo x che è radice di un binary search tree contenente tutti e soli gli elementi di $A[i..j]$.

Abbiamo che $(k-1) - i + 1 < n$ e $j - (k+1) + 1 < n$, quindi per ipotesi induttiva REC_ARRAY_TO_BST$(A, i, k-1, x)$ e REC_ARRAY_TO_BST$(A, i, k-1, x)$ terminano restituendo due nodi z e w radici di BST contenenti rispettivamente gli elementi di $A[i..k-1]$ e $A[k+1..i]$. Ad x viene assegnata chiave $A[k]$, che è maggiore di tutti i valori contenuti nel BST radicato in z e minore di tutti i valori nel BST radicato in w, in quanto A è ordinato. Quindi vale la tesi. □

Dobbiamo anche dimostrare che l'altezza dell'albero costruito è logaritmica.

Lemma 2. REC_ARRAY_TO_BST(A, i, j, y) *restituisce un nodo x che è radice di un albero di altezza $\theta(\log(j - i + 1))$.*

Dimostrazione. Sia $h(n)$ l'altezza dell'albero costruito a partire da $n = j - i + 1$ elementi. Dobbiamo dimostrare che esiste un $\bar{n} \geq 0$ ed una costante $c > 0$ tali che per ogni $n \geq \bar{n}$ vale che $h(n) \leq c * \log n$. Procediamo per induzione su n.

Per il caso base dobbiamo partire con $\bar{n} = 2$, altrimenti non riusciamo a dimostrare la tesi. Abbiamo che nel caso di un vettore con due elementi viene costruito un albero di altezza 1 (la radice ed un figlio). Quindi $1 = h(2) \leq c * \log 2 = c * 1$ è vera a patto che valga $c \geq 1$.

Per il passo induttivo abbiamo $h(n) = max\{h(m), h(n-m-1)\} + 1$, dove $h(m)$ è l'altezza del sottoalbero sinistro e $h(n-m-1)$ è l'altezza del sottoalbero destro. Sappiamo che $m \leq n/2$ e $n - m - 1 \leq n/2$, quindi per ipotesi induttiva abbiamo che $h(m) \leq c * \log(n/2)$ e $h(n-m-1) \leq c * \log(n/2)$. Da questo otteniamo $h(n) \leq c * (\log(n/2)) + 1 = c * (\log n) - c + 1$ e quest'ultimo è minore o uguale a $c * \log n$ se vale $c \geq 1$.

Quindi vale la tesi con $c \geq 1$.

Dovremmo anche dimostrare che $h(n) \geq d * \log n$ per un'opportuna costante d ed n sufficientemente grande, ma questo è sempre vero per un albero binario contenente n nodi. □

Complessità. Ponendo $n = j - i + 1$ l'equazione ricorsiva di complessità della procedura REC_ARRAY_TO_BST(A, i, j, y) è:

$$T(n) = \begin{cases} \Theta(1) & n = 0 \\ T(m) + T(n-m-1) + \Theta(1) & n > 0 \end{cases}$$

Questa equazione ricorsiva è la stessa che abbiamo già visto a lezione per le visite degli alberi. Si intuisce che la sua complessità è $\Theta(n)$, in quanto vengono eseguite operazioni aventi costo $\Theta(1)$ per ogni elemento del vettore. Si dimostra formalmente il risultato applicando il metodo per sostituzione.