ESERCIZIO DI ASD DEL 13 OTTOBRE 2008

Ricerca per Bisezione in un Vettore Ordinato

BisectSearch(A, k, i, j) ricerca l'elemento k nel vettore $A[i..j]$ (A tra la posizione i e la posizione j). L'algoritmo procede ricorsivamente eliminando ogni volta almeno metà elementi. Il caso base è il caso di una porzione priva di elementi, ovvero $A[i..j]$ con $i > j$. In questo caso sicuramente k non occorre in $A[i..j]$ e quindi viene ritornato il valore -1. Altrimenti si determina l'elemento $A[h]$ in posizione centrale tra i e j e si distinguono tre casi:

- se $A[h] > k$, si analizza ricorsivamente solo la metà di sinistra, ovvero $A[i..h-1]$;
- se $A[h] = k$, possiamo restituire come risultato h;
- se $A[h] < k$, si analizza ricorsivamente solo la metà di destra.

Si noti che era possibile diminuire il numero di casi da considerare da tre facendo attenzione agli indici nel caso limite $j = i+1$. Si presti attenzione al fatto che tutti i casi (anche quelli non base) devono restituire un valore.

MainBisectSearch(A, k) ricerca l'elemento k nel vettore A richiamando la procedura BisectSearch$(A, k, 1, length(A))$.

Algorithm 1 BisectSearch(A, k, i, j)

```
1: if i > j then
2:    return -1
3: else
4:    h ← (j + i)/2
5:    if A[h] > k then
6:       return BisectSearch(A, k, i, h − 1)
7:    else
8:       if A[h] = k then
9:          return h
10:      else
11:         return BisectSearch(A, k, h + 1, j)
12:      end if
13:   end if
14: end if
```

Algorithm 2 MainBisectSearch(A, k)

```
1: return BisectSearch(A, k, 1, length(A))
```

Date: 13 Ottobre 2008.

Correttezza.

Lemma 1 (Correttezza di BisectSearch(A, k, i, j))**.** BisectSearch(A, k, i, j) *termina sempre ed al termine:*

- *se viene restituito p, allora* $A[p] = k$*;*
- *viene restituito* -1 *se e soltanto se* k *non compare in* $A[i..j]$*.*

Dimostrazione. Sia $m = j - i + 1$ la lunghezza della porzione $A[i..j]$. Procediamo per induzione su m.
Base: $m \leq 0$, ovvero $i > j$.

Se $i > j$ la condizione alla riga 1 è soddisfatta e quindi si entra nel primo if che non contiene chiamate ricorsive e cicli, quindi banalmente la procedura termina. Dopo aver eseguito la riga 1 si passa alla riga 2. La riga 2 impone di ritornare come valore -1. Effettivamente se $i > j$ non ci sono elementi nella porzione $A[i..j]$, quindi k non può occorrere in tale porzione.
Passo:

HpInd) Se m è minore di $q > 0$ allora BisectSearch(A, k, i, j) termina restituendo -1 sse k non occorre in $A[i..j]$ e p se $A[p] = k$.

TsInd) Se m è uguale a $q > 0$ allora BisectSearch(A, k, i, j) termina restituendo -1 sse k non occorre in $A[i..j]$ e p se $A[p] = k$.

Visto che $m = q > 0$ la condizione alla riga 1 non è soddisfatta e si passa alla riga 3. Alla riga 4 si calcola la posizione centrale in $A[i..j]$ e si memorizza la posizione nella variabile h. Alla riga 5 si controlla se $A[h] > k$. Se $A[h] > k$, allora, visto che il vettore A è ordinato, l'elemento k non si può trovare nella porzione $A[h..j]$, ma si può trovare nella porzione $A[i..h-1]$. Infatti se $A[h] > k$ si passa alla riga 6 e si ritorna lo stesso valore ritornato da BisectSearch$(A, k, i, h-1)$. Siccome $(h-1) - i + 1 < m$, per ipotesi induttiva BisectSearch$(A, k, i, h-1)$ termina sempre restituendo -1 sse k non occorre in $A[i..j]$ e p se $A[p] = k$. Quindi se $A[h] > k$ abbiamo che vale la tesi. Se non è soddisfatta la condizione alla riga 5, allora $A[h] \leq k$ e si passa direttamente alla riga 7. Alla riga 8 si controlla se $A[h] = k$ ed in questo caso si ritorna correttamente il valore h e si termina. Se neanche la condizione alla riga 8 è soddisfatta, allora $A[h] > k$ e si passa alla riga 10. A questo punto la riga 11 effettua correttamente la chiamata ricorsiva sulla porzione $A[h+1..j]$ e per ipotesi induttiva otteniamo la tesi anche in questo caso. □

Theorem 1. MainBisectSearch(A, k) *termina sempre ed al termine:*

- *se viene restituito p, allora* $A[p] = k$*;*
- *viene restituito* -1 *se e soltanto se* k *non compare in* A*.*

Dimostrazione. Segue immediatamente dal lemma precedente, visto che MainBisectSearch(A, k) esegue solamente BisectSearch$(A, k, 1, length(A))$. □

Complessità. Indichiamo con $m = j - i + 1$, ovvero la lunghezza di $A[i..j]$. Analizziamo la complessità di BisectSearch(A, k, i, j). Ogni riga di codice, a parte le righe 6 e 11, ha complessità $\Theta(1)$ e viene eseguita al più una volta. Inoltre la riga 1 viene sicuramente eseguita una volta. Quindi, escludendo le righe 6 e 11 abbiamo una complessità pari a $\Theta(1)$. La riga 6 comporta una chiamata ricorsiva

su dimensione $m/2$. La riga 11 comporta una chiamata ricorsiva su dimensione $m/2$. La riga 6 e la riga 11 si trovano su due rami di un if, quindi al più una di esse viene eseguita. Nel caso peggiore una delle due chiamate sempre verrà eseguita. Per esempio questo si verifica se k occorre solo in posizione 1. Quindi otteniamo la seguente equazione ricorsiva di complessità:

$$T(m) = \begin{cases} \Theta(1) & m = 1 \\ \Theta(1) + T\left(\frac{m}{2}\right) & m > 1 \end{cases}$$

Risolviamo l'equazione con l'albero delle chiamate ricorsive (si veda Figure). Ricordiamo che $\Theta(1)$ va sostituito con una costante c.

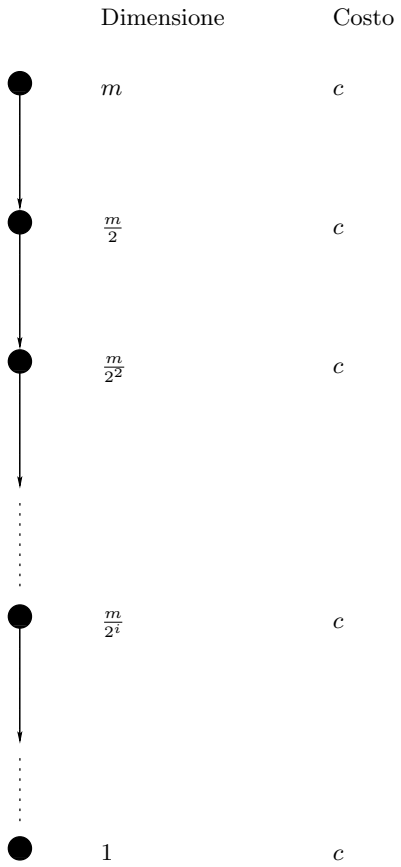

FIGURA 1. Albero delle chiamate ricorsive

Osserviamo che all'i-esimo livello dell'albero la dimensione su cui viene effettuata la chiamata ricorsiva è $\frac{m}{2^i}$. Quindi ci fermeremo al livello x tale che $\frac{m}{2^x} = 1$, cioè $x = \log m$. Da ciò otteniamo

$$T(m) = \sum_{i=0}^{\log m} c = c \log m = \Theta(\log m)$$

Versione Non Ricorsiva. Nella versione non ricorsiva gestiamo i due indici i e j e li aggiorniamo all'interno di un ciclo while. Nel ciclo while k viene confrontato con l'elemento $A[h]$ che si trova in posizione centrale tra la posizione i e la posizione j. Se $A[h]$ è maggiore di k, l'indice j viene spostato verso il basso in posizione $h-1$. Se $A[h]$ è uguale a k si ritorna il valore h, altrimenti l'indice i viene spostato verso l'alto in posizione $h+1$.

Algorithm 3 WhileBisectSearch(A, k)

```
1: i ← 1
2: j ← length(A)
3: while i ≤ j do
4:    h ← (j + i)/2
5:    if A[h] > k then
6:       j ← h − 1
7:    else
8:       if A[h] = k then
9:          return k
10:      else
11:         i ← h + 1
12:      end if
13:   end if
14: end while
15: return −1
```