**ESERCITAZIONI DI** *INTRODUZIONE AGLI ALGORITMI*
**(A.A. 09/10, CANALE E-O)**

APPUNTI ESERCITAZIONE N. 4

## 1. Ricerca Binaria

L'algoritmo Ricerca Binaria risolve il problema della ricerca di una chiave in un vettore. È un esempio di algoritmo che sfrutta una *ipotesi strutturale* sull'input: *si suppone cioè che il vettore sia già ordinato* (nel nostro esempio, in modo non decrescente).

### 1.1. **Versione iterativa.** Diamo lo pseudocodice della versione iterativa.

Ricerca Binaria Iterativa

**Data**: $A$ array ordinato di lunghezza $\ell$, $k$ intero
**Result**: 1 se $k$ è in $A$, 0 altrimenti
**begin**
  $s \longleftarrow 1$
  $d \longleftarrow \ell$
  **while** $(A[\lceil \frac{(s+d)}{2} \rceil] \neq k)$ **do**
    $m \longleftarrow \lceil \frac{(s+d)}{2} \rceil$
    **if** $(A[m] > k)$ **then**
      $d \longleftarrow m - 1$
    **else**
      $s \longleftarrow m + 1$
    **if** $(s > d)$ **then**
      **return** 0
  **return** 1

Stimiamo la complessità nel caso pessimo. Come al solito, siamo interessati al numero di confronti. Dobbiamo stimare (i) quante volte viene eseguito il ciclo $While$ e (ii) il contributo della generica iterazione $i$-esima. Per stimare la lunghezza del ciclo osserviamo che la quantità critica è *la dimensione del vettore* a cui ci si restringe. Tale dimensione viene ridotta - più o meno - della metà. Per semplicità assumiamo che la dimensione del vettore $A$ sia una potenza di 2, così che, ad ogni iterazione del ciclo $While$ la ricerca viene ristretta a un sottovettore di dimensione *esattamente* $1/2$ della dimensione del vettore all'inizio dell'iterazione del ciclo. Quando ci si è ridotti a un sottovettore di dimensione 1 si ha che $s = d$. Se $A[s] \neq k$, allora vene eseguita l'ultima iterazione del ciclo $While$, e o viene aggiornato $s \longleftarrow m + 1 = d + 1$, oppure viene aggiornato $d \longleftarrow m - 1 = s - 1$. In ogni caso si ha $s > d$ e il ciclo ritorna 0. Il numero di iterazioni è dato allora dal numero di volte che possiamo dimezzare il vettore.

Note preparate da Lorenzo Carlucci, carlucci@di.uniroma1.it.

$$n \longrightarrow n/2 \longrightarrow n/2^2 \longrightarrow \ldots \longrightarrow n/2^i \longrightarrow \ldots$$

Il numero di iterazioni del ciclo $While$ è al più $s+1$ per il minimo $s$ tale che $n/2^s = 1$. Dunque il numero di iterazioni è al più $\log_2(n)+1$. Il numero di confronti eseguiti in ciascuna iterazione è costante. Dunque il costo dell'algoritmo è

$$\Theta(\log_2(n)) \cdot \Theta(1) = \Theta(\log_2(n)).$$

1.2. **Versione Ricorsiva.** Diamo ora una versione ricorsiva.

Ricerca Binaria Ricorsiva

**Data**: $A$ array ordinato, $k$ intero, indici $s$ e $d$
**Result**: 1 se $k$ è in $A$, 0 altrimenti
**if** $(s > d)$ **then**
  **return** 0
$m \longleftarrow \lceil (s+d)/2 \rceil$
**if** $(A[m] = k)$ **then**
  **return** 1
**else if** $(A[m] > k)$ **then**
  **return** $RicercaBinaria(A, s, m-1, k)$
**else**
  **return** $RicercaBinaria(A, m+1, d, k)$

La prima chiamata all'algoritmo è con $s = 1$, $d = \ell$, dove $\ell$ è la lunghezza dell'array $A$.

Per scrivere una relazione di ricorrenza, in generale, dobbiamo (a) individuare un caso base, (b) analizzare il caso generico.

Osserviamo che il parametro critico è la dimensione del vettore su cui l'algoritmo lavora, i.e., la dimensione della porzione contenuta tra $s$ e $d$. Chiamiamo $n$ la dimensione del segmento di vettore su cui l'algoritmo lavora, e scriviamo la funzione di costo dell'algoritmo come funzione di $n$.

Come passo base della ricorsione dobbiamo scegliere un caso in cui sappiamo stimare *immediatamente* la complessità dell'algoritmo, senza ridurci ad altri casi. Se l'input è un vettore di dimensione $\leq 1$, allora o non viene eseguito alcun confronto (se $s > d$), oppure $s = d$. In questo caso, si ha $m \longleftarrow \lceil (s+d)/2 \rceil = s$ e vengono eseguito al più 2 confronti: $A[s] = k$, $A[s] > k$. Le eventuali chiamate ricorsive sono $(A, s, s-1, k)$ oppure $(A, s+1, s, k)$. In entrambi i casi sono su valori che soddisfano $s > d$ e dunque non danno luogo ad altri confronti. Possiamo scegliere il seguente caso base della ricorsione.

$$T(n) = \Theta(1) \text{ se } n \leq 1.$$

Dobbiamo ora trattare il caso in cui la dimensione del vettore sia $> 1$. In generale, per scrivere una relazione di ricorrenza occorre stabilire (i) quante chiamate ricorsive avvengono, (ii) per ciascuna di queste, su quali input vengono eseguite, e (iii) come vengono utilizzati i risultati delle chiamate ricorsive. Alla prima chiamata l'algoritmo lavora su tutto il vettore in input, mentre le possibili chiamate ricorsive lavorano su un segmento di dimensione 1/2 la dimensione del segmento considerato nella chiamata precedente. L'algoritmo può fare due diverse chiamate ricorsive: sulla prima metà o sulla seconda metà, ma una sola di queste chiamate viene eseguita (in base alla distinzione dei casi $A[m] > k$ o $A[m] < k$). Oltre alla

chiamata ricorsiva l'algoritmo esegue un numero costante di confronti ($A[m] = k$ e $A[m] > k$). Dunque possiamo scrivere il passo di ricorsione come segue.

$$T(n) = T(n/2) + c \text{ se } n > 1.$$

Mettendo insieme i due casi, abbiamo la seguente equazione ricorsiva per $T(n)$.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ T(n/2) + c & \text{if } n > 1 \end{cases}$$

Per stimarla dobbiamo fare qualcosa di analogo a quanto avviene nell'analisi di algoritmi iterativi. Dobbiamo stimare la lunghezza della ricorsione e trovare la funzione che ci indica il contributo di ciascun passo della ricorsione. Iterando la relazione di ricorrenza abbiamo

$$\begin{aligned} T(n) &= T(n/2) + c \\ &= T(n/2^2) + c + c \\ &\dots \\ &= T(n/2^i) + i \cdot c \\ &\dots \end{aligned}$$

Il termine generico $T(n/2^i) + i \cdot c$ ci dà una formula per il costo della $i$-esima iterazione della ricorrenza, come funzione di $i$. Conosciamo dunque il contributo del passo generico della ricorsione, e non ci resta altro da fare che stimare la lunghezza della ricorsione. Per questo basta chiederci: per quale valore di $i$ *ricadiamo nel caso base*? Il caso base è quando $n/2^i \leq 1$, e dunque la ricorsione si svolge in $i = \log_2(n)$ passi. Dunque abbiamo

$$T(n) = T(n/2^{\log n}) + (\log n) \cdot c = T(1) + (\log n) \cdot c = \Theta(1) + (\log n) \cdot c = \Theta(\log n).$$

## 2. Bubble Sort

Il Bubble Sort è un algoritmo di ordinamento basato su confronti. L'efficienza è pessima ma l'idea di base è semplice e l'analisi del costo un utile esercizio. L'idea del Bubble Sort è quella di scandire ripetutamente l'array $A$ da sinistra a destra confrontando le coppie di elementi adiacenti $A[i], A[i+1]$, e scambiando gli elementi se $A[i] > A[i+1]$. Lo pseudocodice per il Bubble Sort è il seguente.

Bubble Sort

```
Data: A array di dimensione n
Result: A ordinato
for i = 1 to (n − 1) do
    for j = 2 to (n − i + 1) do
        if (A[j − 1] > A[j]) then
            scambia A[j − 1], A[j]
```

Si osserva molto facilmente quanto segue. Dopo la scansione 1 l'elemento massimo è in posizione $A[n]$. *Per lo stesso motivo* – dato che alla seconda scansione l'algoritmo lavora esclusivamente sul sottovettore $A[1, n-1]$ – dopo la scansione 2 l'elemento massimo di $A[1, n-1]$ è in posizione $A[n-1]$. *Per lo stesso motivo*

– dato che alla terza scansione l'algoritmo lavora esclusivamente sul sottovettore $A[1, n-2]$ – dopo la scansione 3 l'elemento massimo di $A[1, n-2]$ è in posizione $A[n-2]$. Ripetendo questa osservazione, abbiamo che, dopo la scansione $i$-esima, l'elemento massimo di $A[1, n-i+1]$ è in posizione $A[n-i+1]$.

Si osserva ora facilmente che: la posizione $A[n]$ è la posizione corretta dell'elemento massimo di $A[1, n]$ nel vettore è ordinato, la posizione $A[n-1]$ è la posizione corretta dell'elemento massimo di $A[1, n-1]$ nel vettore è ordinato, la posizione $A[n-2]$ è la posizione corretta dell'elemento massimo di $A[1, n-2]$ nel vettore è ordinato, e – in generale – la posizione $A[n-i+1]$ è la posizione corretta dell'elemento massimo di $A[1, n-i+1]$ nel vettore è ordinato. Pertanto le osservazioni di sopra si riassumono nel fatto che il Bubble Sort soddisfa il seguente **invariante**.

> Dopo la scansione $i$-esima gli elementi $A[n-i+1], A[n-i], \ldots, A[n]$ sono nella loro destinazione finale. Ossia, vale $A[n-i+1] \leq A[n-i] \leq \cdots \leq A[n]$ e per ogni $j \in [1, n-i]$, $A[j] \leq A[n-i+1]$.

Dunque: dopo $n-1$ scansioni il vettore è ordinato! Quanto costa ogni scansione? Si osserva facilmente che la scansione 1 fa $n-1$ confronti, la scansione 2 fa $n-2$ confronti, e in generale la scansione $i$ fa $n-i$ confronti. Dunque abbiamo tutti gli elementi per stimare la complessità del Bubble Sort (i.e., il numero di scansioni e il costo della scansione $i$-esima come funzione di $i$).

$$\sum_{i=1}^{n-1}(n-i) = \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i = (n-1)\cdot n - \frac{(n-1)\cdot n}{2} = \frac{(n-1)\cdot n}{2} = \Theta(n^2).$$