Dimostrazione di correttezza per Insertion Sort

Felice Cardone
`felice@di.unito.it`

[La seguente dimostrazione è tratta da un lavoro del 1982 di E. W. Dijkstra e A. J. M. van Gasteren (*An introduction to three algorithms for sorting in situ*, Information Processing Letters, **15**(3), 1982, pp. 129–134).]

Si definisce una **catena** come una sequenza finita e non vuota di **elementi**, ciascuno dei quali **contiene** un numero intero. Attenzione alla terminologia: in particolare si intende che gli elementi **non** sono identificati con gli interi che contengono. Si assumono intuitivamente noti il concetto di **predecessore** e **successore** di un elemento di una catena, quando questi esistono, cioè quando l'elemento in questione non è il primo o l'ultimo, rispettivamente. La **discendenza** di un elemento di una catena è un insieme di elementi della catena definito ricorsivamente nel modo seguente:

- La discendenza dell'ultimo elemento (quello privo di successore) è vuota;
- La discendenza di un nodo diverso dall'ultimo è costituita dal suo successore con la discendenza di tale successore.

Un elemento di una catena e **domina** un elemento e' se l'intero contenuto in e non è minore dell'intero contenuto in e'. Un elemento e domina un insieme X di elementi se domina ciascuno dei membri di X.

Un elemento e è **forte** se domina la sua discendenza. Una catena è **decrescente** se ogni elemento che ha un successore domina tale successore.

Proposizione 1 *La proprietà di una catena c di essere decrescente equivale alla proprietà:*

> $P_0(c)$*: ogni elemento di c è forte.*

Dimostrazione: La dimostrazione di questa equivalenza è divisa in due parti:
(1) c decrescente $\Rightarrow P_0(c)$: supponiamo, per assurdo, che c sia decrescente ma che non sia vera la proprietà P_0, cioè che ci sia un elemento di c che non è forte. Chiamiamo e_0 questo elemento. Allora la discendenza di e_0 ha un membro e' tale che $e_0 < e'$. Si può assumere che e' sia l'elemento più a sinistra nella catena con tale proprietà. Allora il predecessore e di e' è tale che $e_0 \geq e$, ma poiché $e' > e_0$ abbiamo $e' > e$, che contraddice l'ipotesi che c sia decrescente.
(2) $P_0(c) \Rightarrow c$ decrescente: supponiamo che e sia un elemento di c con un successore. Per ipotesi c è forte, quindi domina (per definizione) la sua discendenza ed il particolare il suo successore.

Supponiamo che c soddisfi inizialmente la proprietà:

> $P_1(c)$: per ogni elemento e di c: e è forte oppure e è il primo elemento di c.

Consideriamo le proprietà:

$P_2(c, w)$: per ogni elemento e di c: e è forte oppure $e = w$,

$P_3(c, w)$: se w ha un successore s in c, allora w domina s,

Proposizione 2 $P_2(c, w)$ & $P_3(c, w) \Rightarrow P_0(c)$.

DIMOSTRAZIONE: per ogni elemento e di c, se $e \neq w$ allora e è forte per $P_2(c, w)$. Se $e = w$, allora

- o w è l'ultimo elemento di c, ed in questo caso è forte banalmente,
- o w non è l'ultimo elemento, e domina il suo successore s per l'ipotesi $P_3(c, w)$. D'altra parte, per $P_2(c, w)$, s è forte, quindi anche $e = w$ lo è (per la definizione della discendenza di e).

Consideriamo ora il seguente pseudo-algoritmo:

ALGORITMO A: w = "il primo elemento di c";
// $P_2(c, w)$ invariante
while ("w ha un successore s e s domina w") {
"scambia il contenuto di w e s";
$w = s$;
}

Proposizione 3 *Se la proprietà $P_1(c)$ è vera prima dell'esecuzione dell'Algoritmo A, allora la proprietà $P_0(c)$ è vera al termine dell'esecuzione.*

DIMOSTRAZIONE: Prima di tutto è necessario notare che l'esecuzione di queste istruzioni termina, infatti l'assegnamento $w = s$ diminuisce il numero di elementi nella discendenza di w. Prima dell'ingresso nel ciclo while $P_2(c, w)$ è vera per l'ipotesi che inizialmente $P_1(c)$ sia vera. Inoltre, la proprietà $P_2(c, w)$ è invariante se la proprietà

$P_2(c, s)$: per ogni elemento e di c: e è forte oppure $e = s$

è vera dopo lo scambio dei contenuti di w ed s. Infatti,

- per ogni elemento e diverso da w e da s la proprietà $P_2(c, w)$ implica che e è forte anche dopo lo scambio perché né il valore di e né la sua discendenza sono modificati dallo scambio,
- se $e = w$, allora prima dello scambio s domina w perché la condizione del while è verificata, ed s domina la sua discendenza per $P_2(c, w)$ e perché $s \neq w$, quindi e è forte dopo lo scambio,
- se $e = s$ allora la seconda alternativa di $P_2(c, s)$ è verificata

quindi $P_2(c, s)$ è vera. Ma se $P_2(c, w)$ è invariante, all'uscita dal ciclo while – poiché la condizione è falsa, cioè $P_3(c, w)$ è vera – segue dalla Proposizione 2 che è vera $P_0(c)$.

L'algoritmo INSERTION SORT per un array di interi

$$\mathtt{A} = \mathtt{A}[0] \ldots \mathtt{A}[\mathtt{N} - 1]$$

ordina `A` in ordine crescente. Si consideri la proprietà, per $i > 0$:

$P_4(i)$: la catena `A[i-1]`...`A[0]` è decrescente.

Ovviamente la proprietà $P_4(i)$ è vera per $i = 1$, e la verità di $P_4(i)$ & $i = N$ implica che l'array `A` è ordinato in ordine crescente. L'Algoritmo A, codificato in Java, conduce immediatamente al seguente algoritmo Insertion Sort, dove il ciclo while esterno è stato sostituito da un ciclo for, per comodità:

```
int j,v;
for (int i = 1; i < A.length; i++) {
    // P4(i) invariante, per la Proposizione 1
    j = i;
    while (j > 0 && A[j-1] > A[j]) {
          v = A[j];
          A[j] = A[j-1];
          A[j-1] = v;
          j--;
    }
}
```

Una variante dell'algoritmo precedente consiste nel sostituire lo scambio di elementi contigui "fuori posto" con un inserimento di ciascun elemento `A[i]` nella posizione corretta dell'array `A[0]...A[i]` :

```
int j,v;
for (int i = 1; i < A.length; i++) {
    v = A[i];
    j = i;
    while (j > 0 && A[j-1] > v) {
          A[j] = A[j-1];
          j--;
    }
    A[j] = v;
}
```