Appunti Senza Pretese di Programmazione II:
Costruzione di un Heap in Tempo Lineare

Alessandro Panconesi
DSI, *La Sapienza*, Roma

In queste dispense dimostreremo che utilizzando la procedura `SiftUp` descritta in classe é possibile costruire un heap in tempo lineare. Faremo uso di questa notazione per alberi. $T = (x, L_x, R_x)$ denota l'albero binario costituito dalla radice x con sottoalbero sinistro L_x e sottoalbero destro R_x. I figli sinsitro e destro di x saranno denotati rispettivamente come ℓ_x ed r_x.

Ricordiamo che se $T = (x, L_x, R_x)$ soddisfa la proprietá della forma e le uniche possibili violazioni della proprietá dell'ordine possono essere tra x ed i suoi figli, allora `SiftUp` restituisce un heap H con gli stessi elementi di T. Inoltre risulta chiaro che il costo massimo di una invocazione di `SiftUp` é proporzionale ad $h(T)$, l'altezza di T.

Per l'analisi consideriamo prima il caso in cui T é un albero completo, cioé esso ha 1 nodo a livello 0, la radice, 2 nodi a livello 1, 4 nodi a livello 2 e cosí via, sino ad avere 2^k nodi a livello k. Per cui il numero di nodi di T é

$$n := \sum_{i=0}^{k} 2^i = 2^{k+1} - 1. \tag{1}$$

Per costruire un heap a partire da T basta applicare `SiftUp` sistematicamente a tutti i nodi partendo dall'ultimo nodo in basso a destra e procedendo da destra verso sinistra e dal basso verso l'alto. Abbiamo visto come ogni vettore possa essere interpretato come un albero binario che soddisfa la proprietá della forma. In tale albero i due eventuali figli del nodo di posizione i sono i nodi di posizione $2i$ e $2i + 1$ (i nostri vettori iniziano in posizione 1 e terminano in posizione n). Per cui la procedura di costruzione di un heap si puó reinterpretare in questo modo:

```
for (i = 1; i < n+1; i++)
  siftUp(x, i)
```

Dato che il lavoro di `siftUp` é al massimo proporzioanle all'altezza del nodo su cui si opera, complessivamente si svolge un lavoro pari ad 1 per ogni foglia, un lavoro pari a 2 per ogni nodo a livello $k - 1$, un lavoro pari a 3 per ogni nodo a livello $k - 2$ e cosí via:

$$1 \cdot 2^k + 2 \cdot 2^{k-1} + 3 \cdot 2^{k-2} + \ldots k \cdot 2^1 + k + 1 \cdot 2^0 = \sum_{i=0}^{k} (i+1)2^{k-i}. \tag{2}$$

Si tratta quindi di stimare

$$\begin{aligned} T(k) &:= \sum_{i=0}^{k} (i+1)2^{k-i} \\ &= \sum_{i=0}^{k} i2^{k-i} + \sum_{i=0}^{k} 2^{k-i} \\ &= 2^k \sum_{i=0}^{k} i2^{-i} + \sum_{i=0}^{k} 2^i \\ &= 2^{k-1} \sum_{i=0}^{k} ix^{i-1} + n \end{aligned}$$

dove abbiamo posto $x = \frac{1}{2}$. Si tratta adesso di stimare

$$\sum_{i=0}^{k} i x^{i-1}$$

quando $x = \frac{1}{2}$. Stimeremo la maggiorazione

$$g(x) := \sum_{i=0}^{\infty} i x^{i-1} \geq \sum_{i=0}^{k} i x^{i-1}.$$

Sia $f(x)$ la serie geometrica

$$f(x) := \sum_{i=0}^{\infty} x^i = \frac{1}{1-x}.$$

Allora si ha

$$f'(x) = \frac{1}{(1-x)^2} = \sum_{i=0}^{\infty} i x^{i-1} = g(x).$$

Per cui $g(\frac{1}{2}) = 4$ e quindi, notando che $2^{k-1} = n + 1/4$,

$$T(k) \leq 2^{k-1} g\left(\frac{1}{2}\right) + n = 2n + 1.$$

Se T non é completo basta osservare che il piú piccolo albero completo contenente T ha al massimo $2|T| = 2n$ nodi, per cui $4n + 1$ rappresenta un limite superiore del lavoro svolto per costruire un heap.