UNIVERSITÀ DEGLI STUDI DI PADOVA

Appunti di Algoritmi e Strutture Dati

a.a. 2017/2018

Autore:
Timoty Granziero

Repository:
https://github.com/Vashy/ASD-Notes

22 marzo 2018

Indice

8 Lezione del 21/03/2018

Ordinamento Finora abbiamo visto due algoritmi di ordinamento, in cui avevamo le seguenti premesse:

`IN`: $a_1 \dots a_n$;

`OUT`: permutazione $a'_1 \dots a'_n$ ordinata.

In particolare, abbiamo concluso che:

- `Insertion Sort`: $O(n^2)$, basato su scambi;
- `Merge Sort`: $\Theta(n \log n)$, ma con un costo in termini di *memoria*.

Memoria

- `Insertion Sort`:

$$\textit{input} \; + \; 1 \text{ variabile} \Rightarrow \text{spazio } \textit{costante } \Theta(1) \text{ (detto ``in loco'')}$$

- `Merge Sort`: spazio con costo lineare.

$$\begin{aligned} S_{MS}(n) &= max\Big\{S\Big(\Big\lfloor\frac{n}{2}\Big\rfloor\Big),\ S\Big(\Big\lceil\frac{n}{2}\Big\rceil\Big),\ \Theta(n)\Big\} \\ &= \Theta(n) \end{aligned}$$

8.1 Heapsort

L'Heapsort[1] è un algoritmo di ordinamento basato su una struttura chiamata heap, che prende le caratteristiche positive di `Insertion Sort` e `Merge Sort`:

- in "loco" (spazio $\Theta(1)$);
- complessità $\Theta(n \log n)$.

Cos'è un heap? Un `heap` è una struttura dati basata sugli alberi che soddisfa la "proprietà di heap": se A è un genitore di B, allora la chiave di A è ordinata rispetto alla chiave di B conformemente alla relazione d'ordine applicata all'intero heap.

Seguono alcune definizioni.

[1] Anche qui, si consiglia di dare un occhio ad altre fonti. In classe, sono stati viste molte rappresentazioni grafiche degli heap, e, come già detto, in LaTeX non è per me facile rappresentarli.

Altezza: è la distanza dalla radice alla foglia più distante;

Albero completo: è un albero di altezza h con $\sum_{i=0}^{h} 2^i - 1$ nodi;

Albero quasi completo: è un albero completo a tutti i livelli eccetto l'ultimo, in cui possono mancare delle foglie.

Gli heap verranno rappresentati in array monodimensionali, nel modo descritto di seguito:

$$\forall i > 0$$

- `A[i]` è il nodo genitore;
- `A[2i]` è il figlio sx del nodo `A[i]`;
- `A[2i+1]` è il figlio dx.

Inoltre, ogni array `A` sarà dinamico, e avrà:

- *A.length* potenziale spazio, capacità massima dell'array;
- *A.heapsize* celle effettive dell'array.

Vediamo alcune funzioni di utilità che verranno usate.

```
LEFT(i)
   // restituisce il figlio sx del nodo i
1  return 2 * i
```

```
RIGHT(i)
   // restituisce il figlio dx del nodo i
1  return 2 * i + 1
```

```
PARENT(i)
   // restituisce il genitore del nodo i
1  return ⌊i/2⌋
```

8.1.1 Max Heap

Max Heap è uno heap che soddisfa la seguente proprietà:

$$\forall \text{ nodo } A[i],$$
$$A[i] \geq \text{discendenti}$$
$$\Downarrow$$
$$A[i] \geq A[\mathit{Left(i)}],\ A[\mathit{Right(i)}]$$

Osservazioni

- Uno heap con un solo elemento è un *Max Heap*.
- Dati due Max Heap T_1 e T_2 e un nodo N, possiamo "combinarli" in uno heap con N come radice, T_1 come *left* e T_2 come *right*.

Ecco ora una procedura che, dato un nodo i, trasforma in un Max Heap il sotto-albero eradicato in esso (con radie i).

```
MaxHeapify(A, i)
l = Left(i)
r = Right(i)
if (l ≤ A.heapsize) and (A[l] > A[i])
else
      max = i
if (max ≠ i)
      A[i] ↔ A[max]
      MaxHeapify(A, max)
```

L'algoritmo ha un costo di $O(h)$, con h altezza del sotto-albero radicato in i, con

$$O(h) \cong O(\log n) \qquad \text{(Omessa la dimostrazione)}$$

Ora vogliamo scrivere una procedura che costruisce un *Max Heap* da un array qualunque.

Quali sono i nodi foglia?

- Se $i \geq \lfloor \frac{n}{2} \rfloor + 1$

$$2i = 2\Big(\frac{n}{2} + 1\Big) \geq n + 2 - 1 = n + 1$$
$$\Rightarrow i \text{ foglia}$$

- Se $i \leq \lfloor \frac{n}{2} \rfloor$

$$2i = 2\lfloor \frac{n}{2} \rfloor \leq n$$
$$\Rightarrow i \text{ non foglia}$$

```
BuildMaxHeap(A)
A.heapsize = A.length
for i = ⌊A.length/2⌋ down to 1
      MaxHeapify(A, i)
```