Azzolini Riccardo 2019-03-26

Ordinamento

1 Problema dell'ordinamento

Siano U un insieme di elementi, e $(\leq) \subseteq U \times U$ una relazione d'ordine[1] totale[2] su U.

Problema: **Ordinamento**

- *Input*: una sequenza $X \in U^n$ di dati dello stesso tipo.
- *Output*: la sequenza X ordinata, cioè tale che $X[i] \leq X[i+1]$ per ogni $1 \leq i < n$.

2 Classificazione degli algoritmi di ordinamento

Gli algoritmi di ordinamento si suddividono in due classi:

confronti e scambi: sono basati sulle operazioni atomiche di

- **confronto**, $X[i] \leq X[j]$ (o $X[i] < X[j]$)
- **assegnamento** (o **semi-scambio**), $X[i] := X[j]$

digitali: le operazioni possono riguardare singoli (o gruppi di) bit della rappresentazione di ciascun elemento.

Inoltre, un algoritmo si dice

- **stabile** se rispetta l'ordine relativo degli elementi di ugual valore;
- **adattivo** se i confronti effettuati dipendono dai dati.

Infine, gli algoritmi di ordinamento si possono classificare in base a dove risiedono i dati su cui operano:

- se si trovano in memoria centrale, si parla di **ordinamento interno**;
- se sono in memoria esterna, gli algoritmi si dicono di **ordinamento esterno**.

[1] riflessiva, antisimmetrica e transitiva.
[2] $a \leq b \vee b \leq a \quad \forall a, b \in U$

3 Equivalenza per confronti

Sia $(\approx_c) \subseteq U^n \times U^n$ una relazione di equivalenza su U^n tale che $X \approx_c Y$ se e solo se

$$\forall i, j \quad X[i] < X[j] \iff Y[i] < Y[j]$$

Allora due sequenze X e Y tali che $X \approx_c Y$ si dicono **equivalenti per confronti**: tutti i possibili confronti hanno lo stesso esito su X e su Y.

Le classi di equivalenza di $U^n / \approx_c$ possono essere rappresentate da permutazioni di $\{1, 2, \ldots, n\}$. Ad esempio:

$$[6, 41, 345, 13, 99] \approx_c [1, 3, 5, 2, 4]$$

4 Inversioni

Un'**inversione** in una sequenza $X \in U^n$ è una coppia (i, j) tale che $i < j$ e $X[i] > X[j]$.

Il **numero di inversioni** di $X \in U^n$ è

$$N_{inv}(X) = \#\{(i, j) \mid (i, j) \text{ è un'inversione di } X\}$$

4.1 Numero minimo e massimo di inversioni

$$\forall X \in U^n \quad 0 \le N_{inv}(X) \le \frac{n(n-1)}{2}$$

Dimostrazione: Il minimo, 0, si ha quando la sequenza X è in ordine crescente.

Per calcolare il massimo, invece, si osserva che tutte le possibili inversioni sono

$$\begin{array}{rl}
(1,2),\ (1,3),\ (1,4),\ \ldots,\ (1,n) & \text{totale } n-1 \\
(2,3),\ (2,4),\ \ldots,\ (2,n) & \qquad\quad n-2 \\
(3,4),\ \ldots,\ (3,n) & \qquad\quad n-3 \\
\vdots & \\
(n-1,n) & \qquad\quad n-(n-1) = 1
\end{array}$$

cioè $\sum_{i=1}^{n-1}(n-i)$, che (riordinando gli addendi) è uguale a $\sum_{i=1}^{n-1} i$, il cui valore si ottiene mediante la formula di Gauss:

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

Infatti:

$$\begin{aligned}
2\sum_{i=1}^{n-1} i &= \sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1}(n-i) \\
&= 1 + 2 + \cdots + (n-1) \\
&+ (n-1) + (n-2) + \cdots + 1 \\
&= 1 + (n-1) + 2 + (n-2) + \cdots + (n-1) + 1 \\
&= \underbrace{n + n + \cdots + n}_{n-1 \text{ volte}} = n(n-1) \\
\sum_{i=1}^{n-1} i &= \frac{n(n-1)}{2}
\end{aligned}$$

e, più in generale,

$$\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$$

per qualsiasi k (in questo caso, $k = n-1$).

4.2 Conto delle inversioni

Data una sequenza, ci sono due modi per contarne il numero di inversioni: per ogni elemento, si possono contare

1. gli elementi minori alla sua destra;
2. gli elementi maggiori alla sua sinistra.

4.2.1 Esempio

$$[7, 4, 8, 2, 5, 6, 1, 3]$$

Metodo 1:

Elemento	7	4	8	2	5	6	1	3	Inversioni
Minori a destra	6	3	5	1	2	2	0	0	19

Metodo 2:

Elemento	7	4	8	2	5	6	1	3	Inversioni
Maggiori a sinistra	0	1	0	3	2	2	6	5	19

5 Limite inferiore al numero di confronti

Per gli algoritmi *confronti e scambi* esiste un limite inferiore (*lower bound*) al numero di confronti eseguiti nel caso peggiore.

5.1 Lemma 1

Sia $H(k)$ l'altezza di un albero binario con k foglie. Allora,

$$H(k) \geq \lceil \log_2 k \rceil$$

Dimostrazione: Si procede per induzione sul numero di foglie.

- *Caso base*: L'altezza minima di un albero con $k = 1$ foglie è 0, corrispondente all'albero costituito dalla sola radice, cioè $H(1) \geq 0 = \lceil \log_2 1 \rceil$.
- *Passo di induzione*: Sia T un albero con $k > 1$ foglie di altezza minima. Allora, T ha due sottoalberi T_1 e T_2, tali che:
  - T_1 e T_2 non sono nulli, altrimenti sarebbe possibile rimuovere delle foglie dall'ultimo livello e attaccarle alla radice di T, possibilmente diminuendo l'altezza, che quindi non era minima;
  - T_1 (o, equivalentemente, T_2) ha almeno $\lceil \frac{k}{2} \rceil$ foglie, perché se entrambi i sottoalberi ne avessero di meno, il totale sarebbe inferiore a k.

Per ipotesi di induzione,

$$H(T_1) \geq \left\lceil \log_2 \left\lceil \frac{k}{2} \right\rceil \right\rceil \geq \left\lceil \log_2 \frac{k}{2} \right\rceil$$

e, siccome T_1 è sottoalbero di T,

$$H(T) \geq 1 + \left\lceil \log_2 \frac{k}{2} \right\rceil = 1 + \lceil \log_2 k - 1 \rceil = \lceil \log_2 k \rceil \qquad \square$$

5.2 Teorema

Qualunque algoritmo della classe *confronti e scambi* esegue nel caso peggiore almeno

$$n \log_2 n + o(n \log_2 n)$$

confronti per ordinare una sequenza di n dati.

Un algoritmo di ordinamento di questa classe con complessità $O(n \log n)$ nel caso peggiore si dice **ottimale**.

Dimostrazione:

1. Si suppone che l'input sia una permutazione di lunghezza n (ciò non è una limitazione grazie all'equivalenza per confronti).
2. Si costruisce un albero binario che rappresenta il comportamento di qualunque algoritmo di ordinamento *confronti e scambi*:
   - ogni nodo è etichettato con un'operazione $X[i] < X[j]$, quindi il fatto che abbia due figli corrisponde ai due possibili esiti del confronto;
   - un nodo al livello k corrisponde al $(k+1)$-esimo confronto effettuato;
   - le foglie rappresentano l'arresto dell'algoritmo, perciò corrispondono alla sequenza ordinata (ottenuta mediante gli scambi, che per questo teorema si trascurano);
   - ogni cammino dalla radice a una foglia è la sequenza di confronti necessaria a riordinare una delle possibili permutazioni di lunghezza n, che è diversa per ogni permutazione, e di conseguenza il numero di foglie è pari al numero di permutazioni, $n!$;
   - l'altezza dell'albero è il numero massimo di confronti.

3. Per il Lemma 1, l'altezza dell'albero è maggiore o uguale a

$$\lceil \log_2 n! \rceil = \left\lceil \sum_{i=1}^{n} \log_2 i \right\rceil = n \log_2 n + o(n \log_2 n) \qquad \square$$

5.2.1 Comportamento asintotico della sommatoria

È possibile dimostrare che

$$\sum_{i=1}^{n} \log_2 i = \Theta(n \log n)$$

mediante alcune minorazioni e maggiorazioni (si omette invece la dimostrazione della notazione asintotica più precisa, $n \log_2 n + o(n \log_2 n)$).

- Siccome $0 \le \log_2 i \le \log_2 n$ per $1 \le i \le n$, si ha

  $$0 \le \sum_{i=1}^{n} \log_2 i \le n \log_2 n$$

  da cui si ricava che $\sum_{i=1}^{n} \log_2 i = O(n \log n)$.
- Si suddivide in due la somma

  $$\sum_{i=1}^{n} \log_2 i = \sum_{i=1}^{\frac{n}{2}} \log_2 i + \sum_{i=\frac{n}{2}+1}^{n} \log_2 i \ge \sum_{i=\frac{n}{2}+1}^{n} \log_2 i$$

  e, sapendo che $\log_2 i \ge \log_2 \left(\frac{n}{2} + 1\right) \ge \log_2 \frac{n}{2}$ per $\frac{n}{2} + 1 \le i \le n$, si effettua la minorazione

  $$\sum_{i=1}^{n} \log_2 i \ge \sum_{i=\frac{n}{2}+1}^{n} \log_2 i \ge \frac{n}{2} \log_2 \frac{n}{2}$$

  verificando così che $\sum_{i=1}^{n} \log_2 i = \Omega(n \log n)$.

Infine, per le proprietà delle notazioni asintotiche:

$$\left.\begin{aligned}\sum_{i=1}^{n} \log_2 i = O(n \log n) \\ \sum_{i=1}^{n} \log_2 i = \Omega(n \log n)\end{aligned}\right\} \iff \sum_{i=1}^{n} \log_2 i = \Theta(n \log n) \qquad \square$$

6 Implementazione degli algoritmi di ordinamento

Le implementazioni degli algoritmi di ordinamento verranno inserite in una classe contenente alcuni metodi di supporto:

```
public class NomeAlgoritmo {
    private static boolean less(Comparable v, Comparable w) {
        return v.compareTo(w) < 0;
    }

    private static void exch(Comparable[] a, int i, int j) {
        Comparable t = a[i];
        a[i] = a[j];
        a[j] = t;
    }

    public static void sort(Comparable[] a) {
        // Codice dell'algoritmo
    }
}
```

7 Selection sort

```
public class Selection {
    public static void sort(Comparable[] a) {
        int N = a.length;
        for (int i = 0; i < N, i++) {
            int min = i;
            for (int j = i + 1; j < N; j++) {
                if (less(a[j], a[min])) min = j;
            }
            exch(a, i, min);
        }
```

```
    }
}
```

L'algoritmo **selection sort** (**ordinamento per selezione**) utilizza due cicli `for`:

- il ciclo esterno fissa la posizione da "riempire";
- il ciclo interno trova l'elemento da inserire (mediante uno scambio) in tale posizione, cercando il più piccolo a destra.

7.1 Complessità

- Numero di confronti:

$$\begin{aligned}\sum_{i=0}^{n-1}\sum_{j=i+1}^{n-1} 1 &= \sum_{i=0}^{n-1}((n-1)-(i+1)+1) = \sum_{i=0}^{n-1}(n-i-1) \\ &= (n-1)+(n-2)+\cdots+(n-(n-2)-1)+(n-(n-1)-1) \\ &= (n-1)+(n-2)+\cdots+1+0 \\ &= \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = \Theta(n^2)\end{aligned}$$

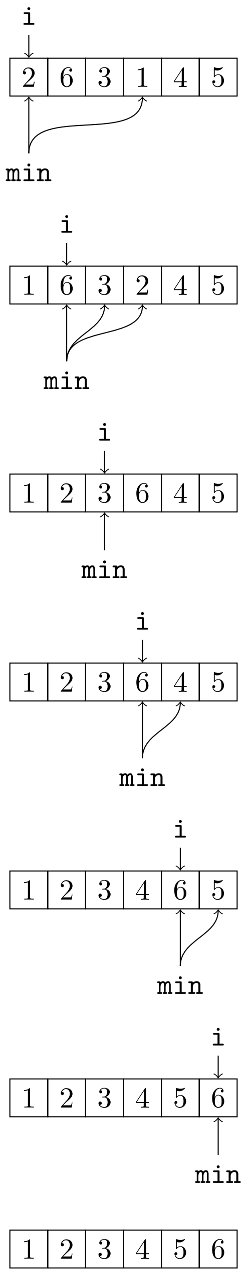
- Numero di scambi:

$$\sum_{i=0}^{n-1} 1 = n = \Theta(n)$$

- Per ogni confronto e ogni scambio viene effettuato un numero costante di altre operazioni a costo costante.

In ogni caso, il numero complessivo di operazioni è $\Theta(n^2)$. Ciò significa che questo algoritmo *non* è adattivo.

7.2 Esempio


i
2 6 3 1 4 5
min
i
1 6 3 2 4 5
min
i
1 2 3 6 4 5
min
i
1 2 3 6 4 5
min
i
1 2 3 4 6 5
min
i
1 2 3 4 5 6
min
1 2 3 4 5 6