Algoritmi e Strutture Dati

Leonardo Baldo

Contents

1 Introduzione

Algoritmo: procedura che descrive tramite passi elementari come risolvere un problema (tramite un modello di computazione).

Uno stesso problema può essere risolto da diversi algoritmi. Di ogni algoritmo siamo interessati a conoscere:

- correttezza
- stabilità
- complessità

1.1 Induzione

L'induzione si struttura con:

- Un caso base $P(0)$.
- Un'ipotesi induttiva (se vale per $P(n)$ allora vale anche per $P(n+1)$).

Matematicamente parlando significa che:

- Normalmente ho una formula, per esempio $n = 1$.
- Se vale per $n = 1$, provo con $n = 2$.
- Se funziona anche con $n = 2$, vuol dire che per $P(n)$ varrà anche $P(n+1)$ e tutti i successivi.

Si ha anche un'ulteriore variante, l'induzione forte:

- U contiene 1 oppure 0.
- Se U contiene tutti i numeri minori di nallora contiene anche n.

La parola "forte" è legata al fatto che questa formulazione richiede delle ipotesi apparentemente più forti e stringenti per inferire che l'insieme U coincida con N per ammettere un numero nell'insieme è richiesto infatti che tutti i precedenti ne faccianogià parte (e non solo il numero precedente).

1.2 Ricorrenza

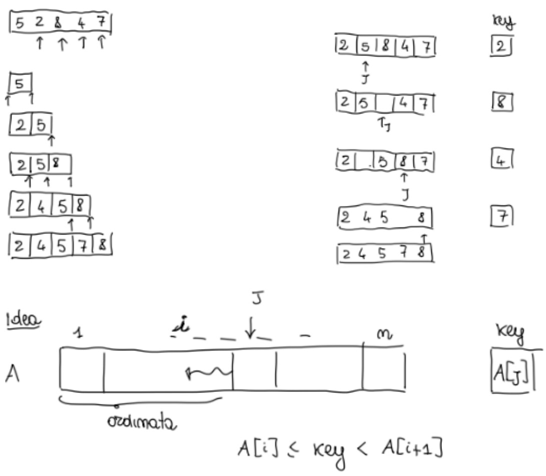
Relazione di ricorrenza è una formula ricorsiva che esprime il termine n-esimo di una successione in relazione ai precedenti. La relazione si dice di ordine r se il termine n-esimo è espresso in funzione al più dei termini $(n-1), \ldots, (n-r)$.

2 Insertion Sort

L'array viene virtualmente diviso in una parte ordinata e una non ordinata. I valori della parte non ordinata vengono prelevati e collocati nella posizione corretta della parte ordinata.
Caratteristiche dell'ordinamento per inserzione:

- Questo algoritmo è uno dei più semplici e di semplice implementazione.
- Fondamentalmente, l'ordinamento per inserzione è efficiente per piccoli valori di dati
- L'ordinamento per inserzione è di natura adattativa, cioè è adatto a insiemi di dati già parzialmente ordinati.

Quello che sostanzialmente fa è esaminare gli elementi a coppie e ordinarli gradualmente per come si presentano.



2.1 Pseudocodice

```
INSERTION-SORT(A)
    n = A.length
    for j = 2 to n
        key = A[j]
        i = j - 1
        while i > 0 and A[i] > key
            A[i + 1] = A[i]
            i = i - 1
        A[i + 1] = key
```

2.2 Correttezza

Definiamo i seguenti passi per A da indice 1 ad indice $j-1$

Inizializzazione: $j=2, A[1,1]$ ordinato

Mantenimento: inserisce $A[j]$ in $A[1 \ldots j-1]$ ordinato

Cancellazione: $j=n+1$

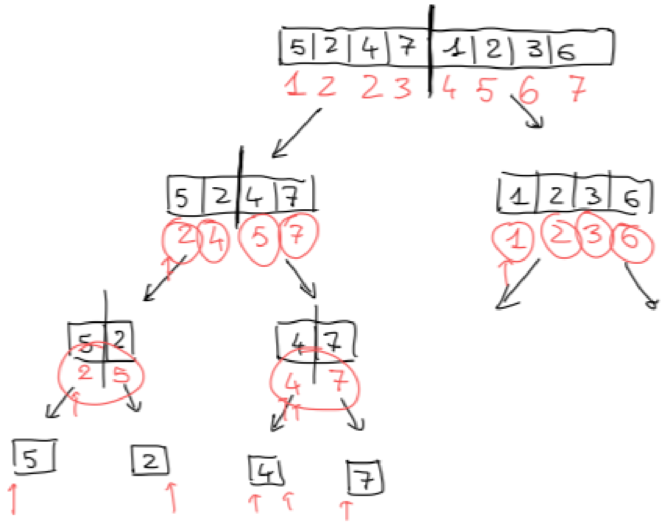
3 Merge Sort

Adotta l'approccio divide et impera, quindi:

- divide: prende il problema originale e lo divide in problemi più piccoli.
- impera: ricorsivamente, risolve i sottoproblemi; se abbastanza piccolo, si risolve subito.
- combina: le soluzioni di P_1,...,P_k si combinano in una soluzionedi P.

Quindi, mergesort adotta i seguenti passi:

- dividere l'array in due parti
- ordina i sottoarray
- fonde i sottoarray ordinati



3.1 Pseudocodice

```
MERGE-SORT(A, p, r)
    if p < r
        q = (p+r)/2
        MERGE-SORT(A, p, q)
        MERGE-SORT(A, q+1, r)
        MERGE(A, p, q, r)
```

```
MERGE(A, p, q, r)
    n1 = q - p + 1
    n2 = r - q
    for i = 1 to n1
        L[i] = A[p+i-1]
    for j = 1 to n2
        R[j] = A[q+j]
    L[n1+1] = R[n2+1] = infinity
    i = j = 1
    for k = p to r
        if L[i] <= R[j]
            A[k] = L[i]
            i = i + 1
        else // L[i] > R[j]
            A[k] = R[j]
            j = j + 1
```

3.2 Complessità

$A[p, ..., k-1]$ contiene $L[1, ..., i-1]$ e $R[i, ..., j-1]$ $A[p, ..., k-1] \leq L[1, ..., n_1 - 1]$ e $R[i, ..., n_2 - 1]$ è ordinato

Inizializzazione: $k = p$ e $A[p, ..., k-1] = A[p, p-1]$

Mantenimento: Divisione in due metà progressive dell'array

Conclusione: $K = r + 1$ e $A[p, ..., r]$ è ordinato e contiene i più piccoli elementi $L[1, ..., n_1 + 1]$ e $R[1, ..., n_2 + 1]$
La dimostrazione del perchésia corretto viene fatta per induzione: se vale per il primo elemento, vale per tutti i casi successivi.