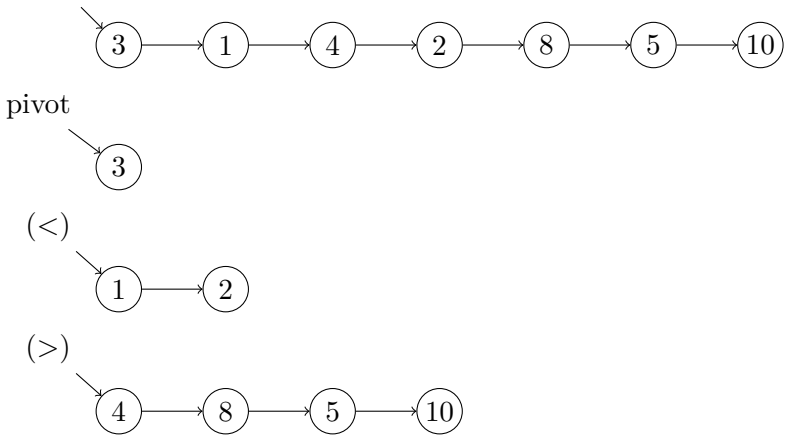
Azzolini Riccardo 2019-04-09

Quicksort

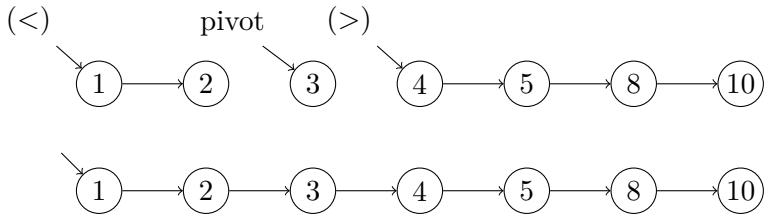
1 Quicksort su lista concatenata

Il quicksort può essere facilmente implementato su una lista concatenata.

Per prima cosa, si esegue il partizionamento mediante la costruzione di due liste, una di valori minori del pivot e una di valori maggiori. Ad esempio:



In seguito, dopo aver ordinano ricorsivamente le liste (<) e (>), le si concatena, inserendo tra le due il pivot.



Per poter eseguire la concatenazione in tempo costante, è necessario che le liste siano dotate di riferimenti all'ultimo elemento, cioè che siano delle `DEQueue`.

2 Versione iterativa

Nel caso peggiore, l'altezza dello stack necessaria per la ricorsione è n. Questo problema può essere risolto sviluppando una versione iterativa che identifica i sottoproblemi mediante uno stack di (coppie di) indici e risolve prima quelli di dimensioni minori.

```
private static void sortIt(Comparable[] a, int l, int r) {
    Stack<Integer> s = new Stack<Integer>();
    s.push(l); s.push(r);

    while (!s.isEmpty()) {
        r = s.top(); s.pop();
        l = s.top(); s.pop();
        if (l >= r) continue;
        int i = partition(a, l, r);

        if (i - l > r - i) {
            s.push(l); s.push(i - 1);
            if (i + 1 < r) {
                s.push(i + 1); s.push(r);
            }
        } else {
            s.push(i + 1); s.push(r);
            if (l < i - 1) {
                s.push(l); s.push(i - 1);
            }
        }
    }
}
```

A ogni iterazione del ciclo `while`, viene salvato sullo stack per primo il sottoproblema più grande, seguito da quello più piccolo (se ha almeno 2 dati), che verrà quindi risolto per primo.

2.1 Spazio necessario

Teorema: Per l'ordinamento di un vettore di n elementi, l'altezza massima dello stack usato da `sortIt` è $\approx \log_2 n$.

Dimostrazione: A ogni iterazione del ciclo `while`, l'altezza dello stack può aumentare di 1, perché viene rimosso un problema e ne vengono salvati al massimo 2.

Se il problema corrente ha dimensione α, e viene suddiviso a sua volta in due sottoproblemi, il nuovo elemento in cima allo stack è il più piccolo dei due, che ha sicuramente

dimensione $\leq \frac{\alpha}{2}$. Se, invece, c'è un solo sottoproblema, esso può anche essere più grande di $\frac{\alpha}{2}$, ma l'altezza dello stack non aumenta.

Di conseguenza, la dimensione del problema in cima allo stack si dimezza con ogni aumento di altezza, fino ad arrivare a 1 (cioè a un problema che non può avere sottoproblemi),

$$\underbrace{n \to \frac{n}{2} \to \frac{n}{4} \to \cdots \to \frac{n}{2^k} = 1}_{k=\log_2 n \text{ passi}}$$

quindi lo stack conterrà al massimo $\log_2 n$ problemi. □

3 Versioni avanzate

Esistono varie modifiche dell'algoritmo quicksort che ne migliorano le prestazioni.

Combinando tali modifiche, il numero di confronti (e quindi il tempo di calcolo) nel caso medio si riduce da $\approx 1.39n \log_2 n$ a $< 1.2n \log n$.

Non esiste, invece, un modo efficiente per rendere stabile quest'algoritmo.

3.1 Selezione del pivot

Selezionando come pivot il primo elemento, la probabilità che sia il valore minimo o massimo, cioè che si verifichi il caso peggiore, è $\frac{2}{n}$ (se non ci sono elementi ripetuti).

Tale probabilità, e in generale la probabilità di scegliere un valore anche solo vicino al minimo/massimo, può essere ridotta selezionando come pivot la mediana tra un numero dispari k di elementi scelti a caso.

Teoricamente, se si scegliesse sempre la mediana dell'intero vettore ($k = n$) si avrebbe complessità $\Theta(n \log n)$ anche nel caso peggiore, ma, in pratica, individuare la mediana "vera" costa troppo nel caso medio.

Effettuando delle analisi sperimentali, si ricava che le prestazioni migliori si hanno con $k = 3$ o $k = 5$ (o al massimo 7). Ad esempio, per $k = 3$ la probabilità del caso peggiore diventa:

$$\frac{2}{n(n-1)(n-2)} \sim \frac{2}{n^3}$$

3.2 Combinazione con insertion sort

Durante l'esecuzione del quicksort si creano molti sottoproblemi di piccole dimensioni.

Per migliorare le prestazioni, e possibile evitare di ordinare le sequenze di lunghezza $\leq k$ (con $5 \leq k \leq 25$ per ottenere il massimo vantaggio, in base a dati sperimentali).

Nel vettore "quasi ordinato" risultante, i dati non ordinati sono comunque confinati tra due pivot, quindi ogni elemento dista al massimo k dalla sua posizione finale, e allora ha un numero di inversioni associate limitato da k.[1] Perciò, si può usare infine l'insertion sort per ordinare questo vettore in tempo lineare, $\Theta(n)$.

Osservazione: Se il quicksort non fosse implementato correttamente, l'algoritmo complessivo funzionerebbe comunque, ma lentamente, perché l'insertion sort dovrebbe "riparare i danni" causati dall'implementazione errata del quicksort.

3.3 Variante di Bentley-McIllroy

Se il vettore può contenere valori ripetuti, questa variante modifica l'operazione di partizionamento in modo da mettere subito in posizione definitiva tutti i valori uguali al pivot.

Un vettore di n elementi con k valori diversi viene così ordinato in tempo $O(kn)$, perché servono al massimo k operazioni di partizionamento (ognuna "mette a posto" tutte le istanze di uno dei k valori). Tali prestazioni sono comparabili a quelle del distribution counting, ma dipendono dal numero effettivo di valori invece che dalla dimensione del range.

L'implementazione di questa variante è molto semplice sulle liste: è sufficiente utilizzare, oltre alle liste di valori minori e maggiori, una terza lista di valori uguali al pivot, che viene poi concatenata in mezzo alle altre due.

Sui vettori, invece, si utilizzano due indici aggiuntivi per accumulare i valori uguali al pivot alle estremità del vettore: quelli "trovati" dall'indice `i` vengono accumulati a sinistra, mentre quelli trovati da `j` si accumulano a destra. Successivamente, tutti questi valori vengono disposti intorno alla posizione del pivot. Il numero di confronti non cambia rispetto alla versione di base, ma si effettuano degli scambi in più perché i valori uguali al pivot vengono spostati due volte, prima verso l'esterno e poi verso il centro.

[1] In particolare, i pivot hanno 0 inversioni.

4 Sintesi delle caratteristiche

Il quicksort è un algoritmo di ordinamento:

- molto veloce nel caso migliore e nel caso medio (soprattutto le versioni avanzate);
- non ottimale, perché nel caso peggiore ha sempre complessità $\Theta(n^2)$, quindi non è adatto alle situazioni in cui servono prestazioni garantite;
- non stabile.