17 Algoritmi probabilistici

Idea. *Se non sapete cosa fare fare una scelta casuale.*

Nota. *Il calcolo delle probabilità è applicato ai dati di output, non ai dati di input.*

17.1 Algoritmi Montecarlo

Idea. *Sono algoritmi in cui la correttezza è probabilistica.*

17.2 Algoritmi Las Vegas

Idea. *Sono algoritmi corretti, in cui il funzionamento è probabilistico.*

17.2.1 Statistiche d'ordine

```
heapSelect(ITEM[] A, int n, int k)
    heapBuild(A)
    per i = 1 fino a k − 1 fai
        deleteMin(A, n)
    // restituisce il k-esimo valore
    ritorna deleteMin(A, n)
```

Analisi della complessità Non va bene per $k = n/2$, perchè viene $\mathcal{O}(n + n/2 \log n) = \mathcal{O}(n \log n)$ Cambiamo tattica. Utilizziamo una tecnica dividi-et-impera simile al quickSort. Ma, a differenza di quest'ultimo, non è necessario cercare in entrambe le partizioni.

```
selection(ITEM[] A, int start, int end, int k)
    se start == end allora
        // caso base: ho un solo elemento
        ritorna A[start]
    altrimenti
        // calcolo indici
        int j = perno(A, start, end)  %  perno
        int q = j − start + 1  %  mediana
        se k == q allora
            // ho trovato il mio elemento
            ritorna A[j]
        altrimenti se k < q allora
            // cerco a sinistra
            ritorna selection(A, start, j − 1, k)
        allora
            // cerco a destra
            ritorna selection(A, j + 1, end, k − q)
```

Analisi della complessità Effettuiamo un'analisi simile a quella fatta per il quickSort

Caso pessimo Il caso pessimo è lo stesso del quickSort (che è il caso in cui il vettore sia già ordinato). In quanto dividiamo il vettore con 0 elementi a sinistra ed n elementi a destra della partizione.

$$T(n) = \begin{cases} 1 & n \leqslant 1 \\ T(n-1) + n & n > 1 \end{cases}$$
$$T(n) = \mathcal{O}(n^2)$$

Quindi nel caso pessimo questo algoritmo risulta peggiore di ordinare il vettore, per una complessità di $\mathcal{O}(n \log n)$ e di andare a fare la ricerca.

Caso ottimo Nel caso ottimo ogni volta divido esattamente a metà

$$T(n) = \begin{cases} 1 & n \leqslant 1 \\ T(n/2) + n & n > 1 \end{cases}$$
$$T(n) = O(n)$$

Caso medio Assumiamo che perno restituisca con la stessa probabilità una qualsiasi posizione j del vettore A

$$\begin{aligned} T(n) &= n - 1 + \frac{1}{n} \sum_{q=1}^{n} T\left(\max\{q-1, n-q\}\right) \\ &\leqslant n - 1 + \frac{2}{n} \sum_{q=\lfloor n/2 \rfloor}^{n-1} T(q) \end{aligned}$$

per $n > 1$

$\frac{1}{n}$ la media su tutti i possibili valori di q $\sum_{q=1}^{n}$ rappresenta la dimensionalità di j, $q-1$ gli elementi a sinistra, $n-q$ a destra rispettivamente
$\exists c > 0, \exists m \geqslant 0 : T(n) \leqslant cn \forall n \geqslant m$

$$\begin{aligned} T(n) &\leqslant n - 1 + \frac{2c}{n} \sum_{q=\lfloor n/2 \rfloor}^{n-1} q \\ &\leqslant n + \frac{2c}{n} \left(\sum_{q=1}^{n-1} q - \sum_{q=1}^{\lfloor n/2 \rfloor - 1} q \right) \\ &= n + \frac{2c}{n} \left(\frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right) \\ &\leqslant n + \frac{c}{n} \big(n(n-1) - (n/2+1)(n/2) \big) \\ &= n + c(n-1) - (c/2)(n/2+1) \\ &= n + cn - c - cn/4 - c/2 \\ &= cn \left(\frac{1}{c} + \frac{3}{4} - \frac{3}{2n} \right) \leqslant cn \left(\frac{1}{c} + \frac{3}{4} \right) \leqslant cn \end{aligned}$$

divido la sommatoria a metà
algebra
semplifico
eseguo i calcoli
eseguo i calcoli
la mia disequazione risulta vera

Siamo partiti dall'assunzione che j assume equiprobabilisticamente tutti i valori compresi fra 1 ed n. Se questa assunzione non fosse vera allora i nostri calcoli non avrebbe alcun fondamento. Allora forziamo l'assunzione iniziale scegliendo un valore casuale come perno ($A[$random$(start, end)]$) a differenza di prima dove prendevamo il primo valore ($A[start]$). Questo accorgimento vale anche per quickSort. La complessità nel caso medio diventa quindi $\mathcal{O}(n)$.

Selezione deterministica Supponiamo di avere un algoritmo "black box" che mi ritorni un valore che dista al più $\frac{3}{10}n$ dal mediano nell'ordinamento.

Implementazione

- Suddividi i valori in gruppi di 5. Chiameremo l'i-esimo gruppo S_i, con $i \in [1, \lceil n/5 \rceil]$
- Trova il mediano M_i di ogni gruppo S_i
- Tramite una chiamata ricorsiva, trova il mediano m dei mediani $[M_1, M_2, \ldots, M_{\lceil n/5 \rceil}]$
- Usa m come pivot e richiama l'algoritmo ricorsivamente sull'array opportuno, come nella selection randomizzata
- Quando la dimensione scende sotto una certa dimensione, possiamo utilizzare un algoritmo di ordinamento per trovare il mediano

```
    ITEM[] select(ITEM[] A, int primo, int ultimo, int k)
        // Se la dimensione è inferiore ad una soglia (10), ordina il vettore e
        // restituisci il k-esimo elemento di A[primo...ultimo]
        se ultimo − primo + 1 ⩽ 10 allora
            insertionSort(A, primo, ultimo)                % Versione con indici inizio/fine
            ritorna A[primo + k − 1]

        // Divide A in ⌈n/5⌉ sottovettori di dim. 5 e ne calcola la mediana
        M = new int[1...⌈n/5⌉]
(1)     da i = 1 fino a ⌈n/5⌉ fai
            M[i] = median5(A, primo + (i − 1) · 5, ultimo)

        // individua la mediana delle mediane e usala come perno
(2)     ITEM m = select(M, 1, ⌈n/5⌉, ⌈⌈n/5⌉/2⌉)
        int j = perno(A, primo, ultimo, m)                      % Versione con m in input

        // Calcola l'indice q di m in [primo...ultimo]
        int q = j − primo + 1

        // Confronta q con l'indice cercato e ritorna il valore conseguente
        se q == k allora
            ritorna m
        altrimenti se q < k allora
(3)         ritorna select(A, primo, q − 1, k)
        allora
(3)         ritorna select(A, q+1, ultimo, k − q)

    // calcola la mediana fra 5 elementi
    int nmedian5()
        ritorna m
```

Analisi della complessità

1. il calcolo dei mediani $M[]$ richiede al più $6\lceil n/5 \rceil$ confronti;
2. la prima chiamata ricorsiva dell'algoritmo select viene effettuata su $\lceil n/5 \rceil$ elementi;
3. la seconda chiamata ricorsiva dell'algoritmo select viene effettuata al massimo su $7\frac{n}{10}$ elementi (esattamente $n - 3\left\lceil \frac{\lceil n/5 \rceil}{2} \right\rceil$);

L'algoritmo select esegue nel caso pessimo $O(n)$ confronti

$$T(n) = T(\tfrac{n}{5}) + T(7\tfrac{n}{10}) + \tfrac{11}{5}n = \mathcal{O}(n)$$