Relazione del progetto di Algoritmi e Strutture Dati e Laboratorio A.A. 2018/2019

D'Urso Mattia - Matricola n. 137836

durso.mattia@spes.uniud.it

Indice

Problema

1.1 Spiegazione

Il problema proposto chiede di calcolare la mediana pesata (inferiore) di un insieme di numeri razionali positivi. Si chiede di ricevere in input i valori tramite standard input e stamparli a video tramite standard output.

Consideriamo n valori $w_1 ... w_n$, ed indichiamo con $W = \Sigma^{n}_{i=1} w_i$ la loro sommatoria. La mediana pesata dei n valori è il peso w_k che rende vera la seguente disequazione:

$$\sum_{w_i < w_k} w_i < W/2 \leq \sum_{w_i \leq w_k} w_i.$$

In altre parole dati $w_1 \ldots w_n$ e calcolata la sommatoria W, la mediana pesata inferiore è il valore più piccolo tale che la sommatoria di tutti i valori ordinati precedenti a w_i è minore della metà della sommatoria W.

1.2 Esempio

Input:
A = [0.1, 0.35, 0.05, 0.1, 0.15 , 0.05, 0.2]

$A_{ordinato}$ = (0.05, 0.05, 0.1, 0.1, 0.15, 0.2, 0.35)

W = (0.05 + 0.05 + 0.1 + 0.1 + 0.15 + 0.2 + 0.35) = 1

W/2 = 0.5 > (0.05 + 0.05 + 0.1 + 0.1 + 0.15)

Output:
0.2

Soluzione

2.1 Idea

L'idea alla base della soluzione proposta è quella più intuitiva. La soluzione prevede pochi passaggi che permettono di generare l'output:

- Si ordina l'array con mergeSort
- Si calcola la sommatoria sum e la sua metà halfSum
- Si implementa un while, il cui contatore j va 0 a n (numero di elementi del vettore), in cui viene fatta la sommatoria dell'array e appena la somma temporanea temp supera halfSum il ciclo viene interrotto e restituito il valore nella j-esima posizione

2.2 codice della soluzione

```
// funzione che trova la mediana inferiore pesata
public static double weightedLowerMedian(double[] array) {
    mergeSort(array);

    double temp = 0.0, sum = 0;
    int i = 0, l = array.length - 1;

    for(int j = 0; j <= l; j++) {
        sum = sum + array[j];
    }

    double halfSum = sum/2;

    while( i < l){
        temp = temp + array[i];
        if (temp >= halfSum) {
            break;
        }
        i++;
    }

    return array[i];
}
```

2.3 Correttezza della soluzione proposta

Si può vedere come la soluzione esegua in ordine mergeSort (la cui correttezza è già stata dimostrata) e poi i due cicli.
Il primo è un ciclo for la cui funzione è fare la sommatoria degli elementi dell'input; svolge il suo compito in un numero finito di passi ovvero l.
Il secondo ciclo è un while che itera finché $i < l$ o viene soddisfatto l'if. Il caso base prevede che i e l siano uguali a 0, quindi il ciclo termina subito e viene restituito l'unico elemento dell'array.
Se $i = n$ e $l = m$, con $n < m$, il ciclo continua a iterare fino a che $i < l$ o $temp \leq halfSum$, in entrambi i casi la funzione restituisce l'ultimo valore che assume i. Nel caso in cui venga soddisfatto l'if i sarà un valore compreso tra 0 e l-1, mentre se l'ultimo valore dell'array è più grande di halfSum, l'if non viene soddisfatto e quindi viene restituito $i=l$ poiché la sommatoria dei numeri precedenti è minore dell'ultimo elemento (di conseguenza solo con l'ultimo elemento si passerebbe halfSum).

Compilare la soluzione

3. 1 Requisiti per eseguire java

Per compilare il codice è necessario avere i seguenti software sul dispositivo:

- Java Runtime Environment (JRE)
- Java Development Kit (JDK)

Entrambi scaricabili dal sito https://www.oracle.com/

È possibile controllare la versione (e quindi se il software risiede sulla macchina) tramite il seguente comando:

- java -version

3. 2 Comandi per eseguire il programma (macOS)

Per compilare il file java si devono eseguire i seguenti comandi:

```
1. cd /Utenti/nomeUtente/Scrivania/progettoASD/
2. javac Main.java
3. java Main
```

dopodiché si può inserire l'input e premere invio. Verrà visualizzato nella riga successiva la mediana pesata (inferiore).

Complessità della soluzione proposta

4. 1 Calcolo della complessità della soluzione

La soluzione proposta utilizza due algoritmi principali:

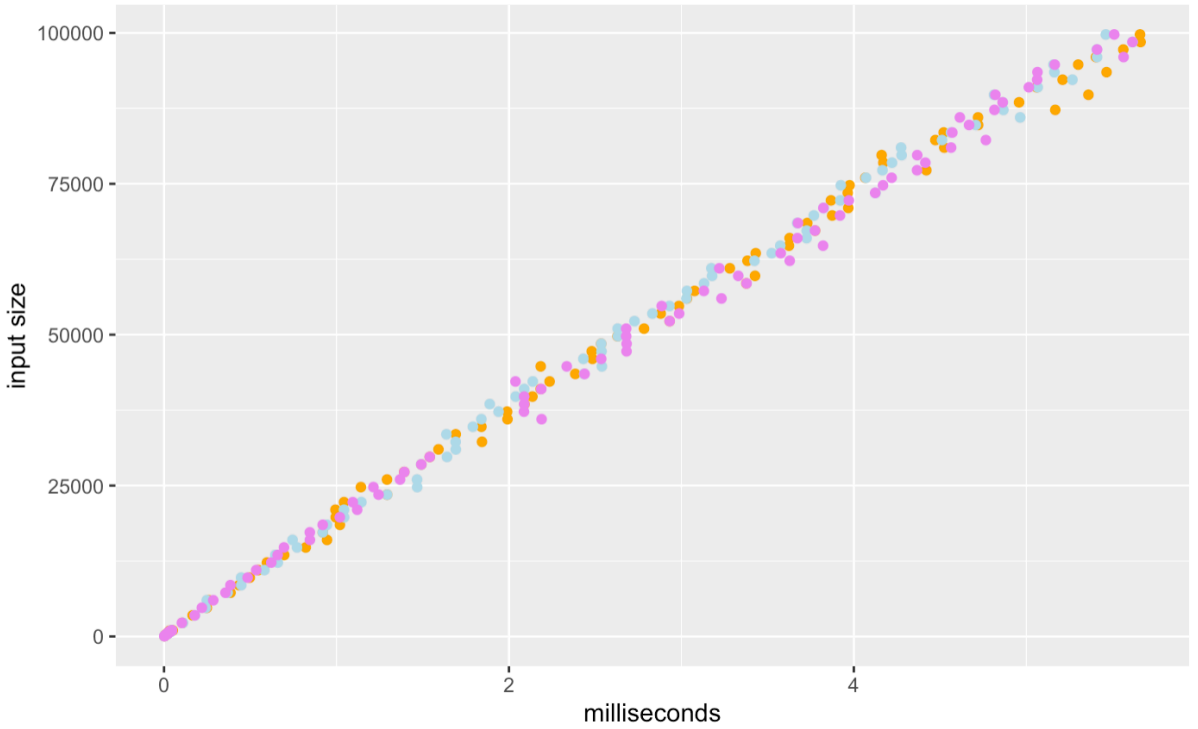
- mergeSort
- weigthedLowerMedian

La complessità del primo algoritmo è stata dimostrata in classe.
Il secondo algoritmo (vedi punto 2.1) utilizza mergeSort, un ciclo for che itera per n volte (dimensione dell'input) eseguendo istruzioni dal peso di O(1) e un ciclo while che, nel peggiore dei casi, si ripete n volte eseguendo istruzioni, anche qui, dal peso di O(1).

$$T(n) = mergeSort + weigthedLowerMedian = \theta(n*log(n)) + \theta(n) + O(n) = \theta(n*log(n))$$

4.2 Misurazione dei tempi e grafico

Il seguente grafico è stato calcolato con array, di numeri generati casualmente, di dimensione compresa tra 50 e 100.000 di elementi. Ogni punto corrisponde a una esecuzione di weigthedLowerMedian, mentre ogni colore evidenzia il comportamento all'aumentare della dimensione dell'input nelle diverse misurazioni.



Osservazioni

5.1 Note sulla soluzione

Poiché l'algoritmo di sorting scelto è mergeSort si può affermare che la soluzione proposta è stabile[1] ma non inPlace[2].

5.1 Note sulla codice

Sono state importate le seguenti classi esterne per gestire la lettura dell'input:

- java.util.Scanner

1. preserva l'ordine relativo dei dati con chiavi uguali all'interno del file da ordinare.
2. non crea una copia dell'input per ordinarlo.