UNIVERSITÀ
DEGLI STUDI
DI UDINE

ALGORITMI E STRUTTURE DATI

PROGETTO PER IL LABORATORIO DI ASD
A.A. 2018/19

Relazione finale

Cognome e Nome:
Zamò Tobia

Matricola:
123364

E-mail:
zamo.tobia@spes.uniud.it

Corso di Laurea:
Tecnologie Web e
Multimediali

Indice

1 Introduzione

1.1 Problema da risolvere

Data la definizione di mediana pesata:

Definizione 1. *Si considerino n valori razionali positivi (pesi)* $w_1, \ldots, w_n$ *e si indichi con* W *la loro somma:* $W = \sum_{i=1}^{n} w_i$. *Chiamiamo* mediana (inferiore) pesata *di* $w_1, \ldots, w_n$, *il peso* w_k *tale che:*

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

Il problema da risolvere è implementare un algoritmo che prende in input una sequenza di numeri razionali positivi $w_1, ..., w_n \in Q^+$ (escluso lo zero) separati da una virgola e terminante con un punto. L'output di tale algoritmo sarà l'elemento che, tra quelli passati in input, soddisfa la definizione di mediana inferiore pesata, tale elemento verrà scritto a terminale.

1.2 Compilazione

I sorgenti in Java della soluzione al problema sono contenuti nella cartella del progetto Zamo_123364

Per compilare utilizzare i seguenti comandi nel teminale:
> javac Progetto.java
> java Progetto < input.txt > output.txt

All'interno della cartella MisurazioneTempi vi sono le sorgenti per la misurazione dei tempi di esecuzione (*vedi Sezione 4*).

2 Soluzione

2.1 Approccio iniziale al problema

Prima di procedere con la scrittura del codice ho eseguito alcuni test su carta per comprendere la definizione. La prima cosa che ho notato era che per individuare la mediana inferiore pesata a mano avrei avuto bisogno di un array ordinato in modo ascendente su cui poi applicare la definizione. L'unica ipotesi sugli input è che sono forniti *n valori razionali positivi* (chiamati pesi). Di conseguenza, all'interno del codice, la soluzione "semplice" sarebbe stata di ordinare l'array in ordine crescente con un algoritmo di ordinamento con complessità nel caso peggiore di $\Theta(nlogn)$ (mergesort o heapsort), calcolare W/2 (la somma di tutti i pesi in input diviso 2) e inizializzare due puntatori p_1, p_2 uno all'inizio dell'array e l'altro alla fine, facendoli scorrere attraverso due cicli while:

- Il primo ciclo esegue una sommatoria partendo dal primo elemento dell'array fino a quando la somma diventa maggiore o uguale a W/2. Ad ogni somma viene incrementato il valore di p_1.
- Il secondo svolge la stessa procedura ma partendo dall'ultimo, anche in questo caso p_2 viene incrementato ad ogni somma.

La mediana inferiore pesata w_k può infatti essere definita nel seguente modo:

$$\sum_{w_i<w_k} w_i < W/2 \tag{1}$$

$$\sum_{w_i>w_k} w_i \leq W/2 \tag{2}$$

Se i due puntatori si fermano sullo stesso elemento, allora l'elemento in tale posizione soddisfa la definizione e viene restituito in output. Altrimenti se p_1 e p_2 puntano a due elementi diversi, per la definizione di mediana **inferiore** pesata viene restituito l'elemento con puntatore p_1 (quello più a sinistra).

2.2 Soluzione deterministica lineare

La definizione di mediana inferiore pesata riportata nel paragrafo precedente può essere anche letta nel seguente modo: la somma delle due partizioni dell'array suddivise dalla mediana ponderata w_k devono essere tra di loro "bilanciate", ossia non devono superare il valore W/2.
Il problema può essere quindi risolto nel seguente modo. Viene inizialmente calcolato W/2 (chiamato target) il cui valore rimarrà invariato in tutte le chiamate ricorsive dell'algoritmo[1].
Se la lunghezza dell'array è ≤ 2, allora la mediana inferiore pesata può essere determinata attraverso ricerca esaustiva, nel caso la lunghezza sia uguale a 2 viene selezionato l'elemento di peso maggiore. In tutti gli altri casi si determina la mediana m utilizzando la selezione (Select) in tempo lineare nel caso peggiore descritta nel paragrafo 9.3 del libro di testo per poi partizionare l'array attorno a tale mediana.
All'interno del codice ho implementato l'algoritmo seguendo il libro di testo: Select determina il k-esimo elemento più piccolo di un array di input di n elementi, effettuando i seguenti passi:

1. Gli n elementi dell'array di input vengono divisi in $\lfloor n/5 \rfloor$ gruppi di 5 elementi ciascuno e (al massimo) un gruppo con i restanti $n \; mod \; 5$ elementi.
2. Viene trovata la mediana di ciascuno degli $\lceil n/5 \rceil$ gruppi, effettuando prima un ordinamento con InsertionSort degli elementi di ogni gruppo (al massimo 5) e poi scegliendo la mediana dalla lista ordinata degli elementi di ogni gruppo.
3. Select viene chiamato ricorsivamente per trovare la mediana (inferiore) delle $\lceil n/5 \rceil$ mediane trovate al passo 2
4. L'array viene partizionato attorno alla mediana delle mediane. Se i indica il numero di elementi nel lato basso della partizione, allora la mediana delle mediane è il $i-$esimo elemento più piccolo e il lato alto della partizione contiene $n - i$ elementi.
5. Se $i = k-1$ allora viene restituito quell'elemento che è la mediana delle mediane (nel mio caso restituisco la posizione di tale mediana che mi servirà per il calcolo della mediana pesata). Se $i > k - 1$ si utilizza Select ricorsivamente per trovare il k-esimo elemento più piccolo nel lato basso, altrimenti se $i < k-1$ il $k - i - 1$ esimo elemento più piccolo nel lato alto.

[1] Nell'interpretazione del problema, dal momento che tutti i pesi appartengono ai numeri razionali positivi Q^+, ho dato per scontato che lo zero sia escluso da tale dominio (altrimenti sarebbe indicato come Q_0^+). In ogni caso, all'interno dell'algoritmo, se la somma di tutti i pesi in input (diviso 2) è uguale a 0, allora viene restituito semplicemente 0 come risultato.

Nella chiamata iniziale la partizione sinistra è costituita da tutti gli elementi (pesi) $\leq m$ e la loro somma non può mai essere $\geq$ W/2.
Si supponga *without loss of generality* di aver partizionato gli elementi dell'array di lunghezza N con l'algoritmo Select sopra descritto:

CASO 1: N è dispari (ossia $N = 2n + 1$)
In questo caso la mediana è l'elemento di posizione $n + 1$
Avendo partizionato l'array con Select si ha:

$$\forall x \in Partizione_{sx}, \quad \forall y \in Partizione_{dx} \qquad x \leq mediana \leq y$$

Sommando su tutti gli elementi delle partizioni ottengo:

$$SOMMA_{sx} \leq n \times mediana \leq SOMMA_{dx}$$

da cui:

$$SOMMA_{sx} \leq SOMMA_{dx}$$

Allora:

$$SOMMA_{sx} + SOMMA_{sx} \leq SOMMA_{sx} + SOMMA_{dx} <$$
$$< SOMMA_{sx} + SOMMA_{dx} + mediana$$

Dal momento che:

$$SOMMA_{sx} + SOMMA_{sx} = 2 \times SOMMA_s x$$

$$SOMMA_{sx} + SOMMA_{dx} + mediana = W$$

Si ha che:

$$2 \times SOMMA_{sx} < W \Rightarrow$$

$$\Rightarrow SOMMA_{sx} < \frac{W}{2}$$

CASO 2: N è pari (ossia $N = 2n$)
In questo non si ha una mediana ben definita, nell'algoritmo viene presa in considerazione quella che, nell'array ordinato in modo crescente, ha posizione $array.length/2 + 1$.

SOTTOCASO: tutti gli elementi dell'array hanno lo stesso valore.
Qui banalmente $Somma_{sx} = SOMMA_{dx} = \frac{W}{2}$
Semplicemente tutto vale k, quindi:

$$Somma_{sx} = nk, \quad SOMMA_{dx} = nk, \quad W = 2nk$$

Se ci sono almeno due elementi diversi allora $SOMMA_{sx} < SOMMA_{dx}$. Infatti possiamo confrontare gli elementi delle due partizioni posto per posto:

$$\left.\begin{array}{c} x_{sx_1} \leq x_{dx_1} \\ \cdots \\ x_{sx_n} \leq x_{dx_n} \end{array}\right\} \begin{array}{c} \text{almeno una disuguaglianza è stretta} \\ \text{perché ci sono almeno due elementi diversi} \end{array}$$

Come nel caso precedente:

$$2 \times SOMMA_{sx} (= SOMMA_{sx} + SOMMA_{sx}) < W (= SOMMA_{sx} + SOMMA_{dx}) \Rightarrow$$

$$\Rightarrow SOMMA_{sx} < \frac{W}{2}$$

Dopo aver partizionato l'array con l'algoritmo Select, la ricerca della mediana inferiore pesata segue un procedimento simile alla ricerca binaria. Viene calcolata la somma di elementi della partizione sinistra (*leftSum*) e della partizione destra (*rightSum*). A queste vengono sommate rispettivamente due variabili: $leftWeight$ e $rightWeight$. Esse rappresentano i "vecchi" pesi delle partizioni: se la chiamata ricorsiva procede ad analizzare la partizione destra, allora $leftWeight$ prende il valore di $leftSum$ corrente, se invece procede ad analizzare la partizione sinistra, allora $rightWeight$ prende il valore di $rightSum$ corrente.

Se $arr.length > 2$, possono presentarsi 3 diversi casi:
CASO 1: la mediana m ricavata con l'algoritmo Select soddisfa la definizione di mediana inferiore pesata[2].

$$leftSum < target$$
$$\&\&$$
$$rightSum \leq target$$

m viene quindi restituita a terminale.

CASO 2: la mediana m NON soddisfa la definizione di mediana inferiore pesata e $leftSum \geq target$
In questo caso viene violata la condizione (1). La variabile $rightWeight$ prende il valore di $rightSum$ e la ricerca della mediana inferiore pesata prosegue nella partizione sinistra dell'array.

CASO 3: la mediana m NON soddisfa la definizione di mediana inferiore pesata e $rightSum > target$
In questo caso viene violata la condizione (2). In modo simile al Caso 2, $leftWeight$ prende il valore di $leftSum$ e la ricerca della mediana inferiore pesata prosegue nella partizione destra dell'array.

Tale procedimento ricorsivo continua fino a quando o viene trovata una mediana m che soddisfa le condizioni del Caso 1, o l'array viene ridotto a 2 elementi e la mediana inferiore pesata viene determinata attraverso ricerca esaustiva.

[2]Vedi sezione 2.1

3 Correttezza e calcolo della complessità

La correttezza e lo studio della complessità viene riportato solo sugli algoritmi risolventi il problema, dunque non vengono presi in considerazione gli aspetti di parsing dell'input.
Per dimostrare la correttezza di questo algoritmo, le seguenti precondizioni devono essere valide per ogni chiamata ricorsiva: la mediana inferiore pesata w_k è sempre presente nell'array delle chiamate ricorsive e il peso totale W degli elementi rimane invariato[3]. Tali precondizioni sono banalmente vere per la chiamata iniziale e nel caso base in cui l'algoritmo è composto da 1 o 2 elementi (in cui viene restituito direttamente il risultato). Si può dimostrare per induzione che tale presupposto è vero anche per le chiamate successive.
Si considerino i casi in cui la definizione di mediana inferiore pesata non è rispettata.
Ogni volta che viene calcolato $leftSum$ e $rightSum$, a questi vengono sommati rispettivamente i valori di $leftWeight$ e $rightWeight$.
Con $leftSum \geq target$ la mediana inferiore pesata w_k si trova, per definizione, nella partizione a sinistra di m: per mantenere il peso totale dell'array invariato (e di conseguenza anche il valore di $target$), $rightWeight$ prende il valore di $rightSum$ e la ricerca prosegue nella partizione sinistra.
Se $rightSum > target$, $leftWeight$ prende il valore di $leftSum$ e la ricerca prosegue nella partizione destra.
In entrambe le chiamate ricorsive, il subarray risultante contiene sia la mediana che la mediana pesata e il peso totale dell'array $leftSum + rightSum + m$ rimane invariato.
L'algoritmo termina sempre perché la dimensione dell'array viene decrementata di $n/2$ ad ogni chiamata ricorsiva, finché la mediana m soddisfa la definizione o la dimensione dell'array viene ridotta a 2 elementi, portando al caso base.

Per quanto riguarda la complessità abbiamo al più una chiamata sulla metà degli elementi più la mediana. La ricerca esaustiva iniziale su 1 o 2 elementi ha complessità $\Theta(1)$, determinare la mediana (inferiore) usando Select richiede $\Theta(n)$ e partizionare attorno alla mediana richiede $\Theta(n)$. Calcolare il peso di W/2 e delle due partizioni ha costo $\Theta(n)$.
L'equazione ricorsiva di tale algoritmo è:

$$T(n) = T(n/2) + \Theta(n)$$

[3]Se W rimane invariato, allora anche il valore del $target$ (W/2) rimane invariato

Risolvendo l'equazione utilizzando il metodo di sostituzione si ha che:

$$\begin{aligned} T(n) &= T(n/2) + \Theta(n) \\ &= T(n/2^2) + n/2 + n \\ &= T(n/2^3) + n/2^2 + n/2 + n \\ &\quad \ldots \\ &= T(n/2^k) + n/2^{k-1} + n/2^{k-2} + \quad \ldots \quad + n/2 + n \end{aligned}$$

Assumendo che $n/2^k = 1$
Si ha che $n = 2^k \Rightarrow k = log(n)$

Allora:

$$\begin{aligned} T(n) &= T(1) + n(1/2^{k-1} + 1/2^{k-2} + \quad \ldots \quad + 1/2 + 1) \\ &= 1 + n(1+1) \\ &= 1 + 2n \end{aligned}$$

Da cui si ricava la complessità dell'algoritmo: $T(n) = \Theta(n)$

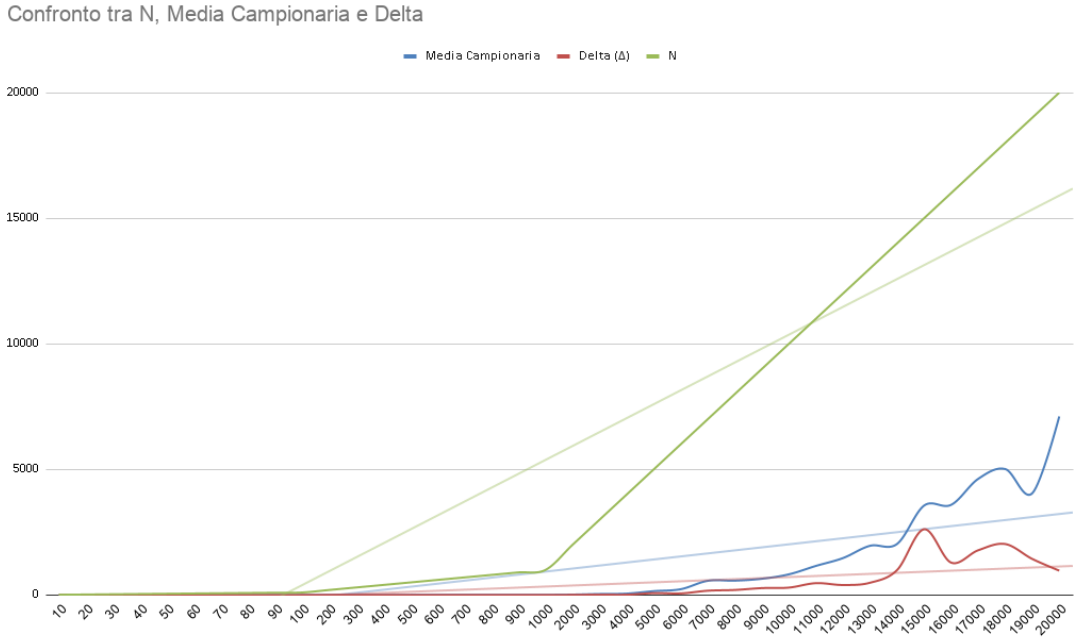
4 Misurazione dei tempi di esecuzione

I test sono stati eseguiti su Windows 10 con processore Intel Core i7-4770 da 3.50Ghz, 8 GB di memoria RAM. Le misurazioni sono state eseguite con input generato in modo pseudocasuale in base ai seguenti parametri:

- Test effettuati su array di dimensione da 10 a 20000 elementi, con numero di misurazioni $c = 10$
- Errore massimo pari al 5%
- $\alpha = 0,05$ e di conseguenza $z\alpha = 1,96$

Qui sotto sono riportati la tabella dei risultati dei primi 25 campioni e i grafici dei risultati ottenuti.

N	Media Campionaria	Delta(Δ)
10	0.6916666666666665	0.16784437562793597
20	1.575	0.21968181963591754
30	1.5	0.16974098244537492
40	1.475	0.18914254252429602
50	1.525	0.16177422853691806
60	1.55	0.18070307487991719
70	1.425	0.12102107490064075
80	1.575	0.1557244391351177
90	1.45	0.1935344968164006
100	1.55	0.15182095012618205
200	1.65	0.14201549487666754
300	1.75	0.19600000381469732
400	1.775	0.18914254252429646
500	2.175	0.1704467692004385
600	2.5	0.20788939771494225
700	3.0	0.25928363353070055
800	3.25	0.3323341756412718
900	3.975	0.35638869923727784
1000	4.875	0.6763065265130495
2000	15.825	2.9765635294542365
3000	39.05	11.997858108982143
4000	55.175	10.9269013332963
5000	152.875	74.74263134219774
6000	230.575	62.42211840184495
7000	564.875	170.1996993948312
. . .	. . .	. . .


Confronto tra N, Media Campionaria e Delta
Media Campionaria
Delta (Δ)
N
20000
15000
10000
5000
0
10
20
30
40
50
60
70
80
90
100
200
300
400
500
600
700
800
900
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
11000
12000
13000
14000
15000
16000
17000
18000
19000
20000