Relazione di progetto

Stefano Pessotto

Introduzione

In questa relazione si propone un'implementazione dell'algoritmo per il calcolo della mediana pesata basato sull'algoritmo $SELECT$ deterministico in place. Il documento analizza i vari algoritmi utilizzati, i problemi implementativi riscontrati e le prestazioni in rispetto ai risultati teorici ottenuti.

Simbologia

$\mathfrak{R}$	Insieme dei numeri reali
$\mathfrak{R}^+$	Insieme dei numeri reali positivi

Compilazione

Il programma è stato compilato e testato in un sistema Windows utilizzando l'ambiente MinGW. Tuttavia il programma è utilizzabile anche in ambiente Linux e compilabile con gli stessi comandi.
Il programma richiesto si troverà nella cartella *"Calcolo mediana pesata"* mentre il programma per il calcolo dei tempi si troverà nella cartella *"Calcolo dei tempi"*.

Programma per il calcolo della mediana pesata

Nella cartella sono presenti due file, *"weightedMedian.c"* e *"weightedMedian.h".*
Per compilare il programma, utilizzando gcc e supponendo sia incluso nelle variabili d'ambiente, bisognerà lanciare da shell il seguente comando :
"gcc weightedMedian.c -o weightedMedian".
Il programma *weightedMedian* ora può essere eseguito.
L'input del programma deve essere fornito tramite **STDIN** come una lista di numeri. Devono essere separati da virgole e terminati da un punto. L'output del programma viene stampato nello stream **STDOUT**.
L'output del programma è dato con una precisione fino alla decima cifra.

Un esempio :

```
C:\Users\Stefano\...>gcc weightedMedian.c -o weightedMedian


C:\Users\Stefano\...>weightedMedian.exe < test.txt
0.2
C:\Users\Stefano\...>
```

In questo caso l'input è la lista di numeri "0.1 , 0.35 , 0.05, 0.1, 0.15 , 0.05, 0.2 ." e l'output è il valore "0.2".

Programma per il calcolo dei tempi

Per compilare il programma per il calcolo dei tempi bisognerà lanciare la linea di comando la seguente pipeline :
"gcc weightedMedian.c timeMeasurement.c -o timeMeasurement".
Il programma sarà eseguibile attraverso il file *timeMeasurement*.
Il programma non riceve alcun input e fornisce l'output tramite **STDOUT**.
Il comportamento del programma è modificabile tramite i parametri del file header "*timeMeasurement.h*" :

1. Definire/Non definire il flag *DO_ALL* per decidere se comparare vari algoritmi per il calcolo della mediana pesata oppure misurare solo quello proposto nel progetto.
2. Modificare *ARRAY_START_SIZE* e *ARRAY_NEXT_SIZE_INCREMENT* per decidere da che dimensione iniziare a misurare e di quanto incrementare ad ogni iterazione.
3. Modificare *NUMBER_OF_ITERATIONS* per decidere quanti campioni si vogliono prendere.

Un esempio di compilazione ed esecuzione è il seguente :

```
C:\Users\Stefano\...>gcc weightedMedian.c timeMeasurement.c -o timeMeasurement.exe

C:\Users\Stefano\...>timeMeasurement.exe
Granularity     TMin(ns)
2500    124999
size    Deterministic (ns)      Deterministic (delta)
1000    460716  22694.926
2000    872610  11731.424
3000    1355365 60291.324
4000    1775397 68984.158
5000    2295365 107918.703
```

```
6000    2675010 125927.373
7000    3130830 76324.730
8000    3598683 124791.703
9000    4003170 129775.941
10000   4593760 195061.926

C:\Users\Stefano\Desktop\...>
```

I valori vengono stampati separati da tabulazioni, quindi copiando e incollando l'output in un editor per la gestione di fogli elettronici (Google Sheets, …) risulteranno incolonnati correttamente.

Attenzione : in caso di errore di compilazione per mancanza della definizione di *CLOCK_MONOTONIC* o di *clock_gettime* aggiungere il seguente argomento nella pipeline di compilazione con gcc : "`-D_POSIX_C_SOURCE=199309L`" [1] .

Il problema

Dato in input un vettore A di n elementi non ordinati rappresentanti dei pesi positivi nel dominio dei numeri reali ($A = \{ w_1, \ldots, w_n \} : \forall i \, w_i \in \Re^+$) denotiamo con $W = \sum_{i=1}^{n} w_i$ la somma degli elementi del vettore a cui riferiremo ora come peso complessivo di A.

La mediana pesata inferiore è il valore $w_k \in A : \sum_{i=1}^{k-1} w_i < \frac{W}{2} \leq \sum_{i=1}^{k} w_i$, ovvero il valore w_k che, sommato ai $k - 1$ valori che lo precedono in un ordinamento di A, permette di raggiungere o superare la metà del peso complessivo di A.

Analogamente possiamo definire il problema come il valore w_k in un ordinamento del vettore A tale che : $\sum_{i=1}^{k-1} w_i \leq \sum_{i=k+1}^{n} w_i < \frac{W}{2}$, ovvero il valore in posizione k di un ordinamento di A per cui la somma degli elementi alla sua sinistra e la somma degli elementi alla sua destra non raggiungono la metà del peso complessivo del vettore.

L'algoritmo

L'algoritmo implementato si basa sulla ricerca dicotomica del peso utilizzando l'algoritmo $SELECT$ per trovare l'elemento α che finirebbe in posizione centrale in un ordinamento di A limitato ad un determinato range $[min, max)$. Il vettore verrà poi partizionato tramite la procedura $PARTITION$ utilizzando tale elemento come pivot.

Dopo aver calcolato il peso delle due partizioni si può determinare se α sia il valore della mediana pesata (inferiore) oppure se iterare la ricerca nei range $[min, \frac{max}{2})$ o $(\frac{max}{2}, max]$.

La procedura Partition

Partition prende in input l'array A, un indice di inizio φ_0, un indice di fine φ_1 e un pivot α, garantendo in output l'indice Φ di A in cui finirebbe α in un ordinamento del vettore e mantenendo in A le seguenti proprietà :

1. $\forall i \in \{ \varphi_0, \ldots, \Phi - 1 \} \; A_i \leq \alpha$
2. $A_\Phi = \alpha$
3. $\forall i \in \{ \Phi + 1, \ldots, \varphi_1 \} \; A_i \geq \alpha$

Una possibile pseudocodifica è la seguente :

```
Partition( A, p, q, pivot )
   i ← p -1
   for( j ← p to q )
      if( A_j ≤ pivot )
         i ← i + 1
         swap(A, i, j )
      end
   done
   return i
end
```

Complessità

Sia $n = q - p$.

La procedura effettua una scansione di n elementi dell'array tramite un ciclo for. Quindi banalmente, il costo totale della procedura è $\Theta(n)$.

L'algoritmo Select

L'algoritmo di selezione deterministica è un algoritmo ricorsivo che mira a fornire il k-esimo elemento di un ordinamento di un vettore di dimensione n in tempo lineare.

L'idea di $SELECT$ è quella di suddividere gli elementi del vettore in blocchi di dimensione fissa e ordinare singolarmente ogni blocco. Ordinando poi i blocchi tra loro tramite i mediani possiamo trovare, nel mediano di questi blocchi, un pivot efficiente su cui effettuare successivamente la procedura $PARTITION$. Questo ci garantisce di scartare una quantità di elementi non trascurabile nel caso in cui il pivot non sia nella posizione cercata, ovvero $\approx \frac{1}{4}n$ degli elementi nel caso peggiore e $\approx \frac{3}{4}n$ nel caso migliore. La ricerca quindi prosegue ricorsivamente.

Nel caso in numero di elementi non sia un numero divisibile per la dimensione dei blocchi, l'elemento di dimensione massima si può aggiungere più volte per evitare di non considerare gli elementi dell'ultimo blocco.

Si può però effettuare un ulteriore accorgimento nel trovare il mediano dell'ultimo blocco :

- se questo ha un numero di elementi che raggiunge la metà della dimensione del blocco, allora il suo mediano sarà il massimo tra questi, perché eventuali valori del massimo aggiunti in questo blocco finirebbero alla fine e non nella posizione del mediano.
- se il numero di elementi è insufficiente per l'ottenimento del mediano si può sostituire quest'ultimo con l'ultima occorrenza del valore massimo nel vettore, perché :
    1. Aggiungendo il valore massimo alla fine dell'ultimo blocco questo risulterà il mediano, poiché tutti gli altri nel blocco sono minori o uguali a lui. (Vale solo nell'ipotesi che il numero degli elementi non sia sufficiente a determinare un mediano).
    2. L'ultima occorrenza del valore massimo sarà ultimo nel suo blocco (nel caso in cui si trovasse nell'ultimo blocco, questo sarà posizionato prima della metà), quindi non è mai mediano di un altro blocco.

Nel caso base e nell'ordinamento dei blocchi si può utilizzare un algoritmo di ordinamento veloce, come $INSERTIONSORT$, che su un input di dimensioni contenute opera con costanti moltiplicative ridotte.

Un'implementazione in place

La criticità nell'implementazione in place di $SELECT$ consiste principalmente nella ricorsione sugli elementi mediani dei blocchi. La soluzione utilizzata nell'implementazione di questo progetto consiste nello spostare gli elementi mediani in una porzione dell'array, ad esempio all'inizio, ed effettuare la chiamata ricorsiva solo su quella parte dell'array.
Inoltre non si rivelano necessarie modifiche all'array prima della chiamata ricorsiva finale, poiché con la procedura $PARTITION$ si spostano gli elementi su cui effettuare la ricorsione in una delle due sezioni dell'array.

Complessità nel caso di elementi distinti

Per valutare la complessità bisogna prima capire quanti elementi vengono scartati nel caso peggiore.
Supponiamo di avere n elementi $a_1, \ldots, a_n$ del vettore A distinti tra loro.
Li dividiamo in $\lceil \frac{n}{5} \rceil$ blocchi di dimensione 5 e, dopo averli ordinati, richiamiamo ricorsivamente la procedura sui mediani dei blocchi, ottenendo il mediano dei mediani a cui riferiremo come α.
Otteniamo la seguente relazione che lega un sottoinsieme di A:
Preso un elemento a_i se :

1. a_i precedeva il mediano del suo blocco
2. Il mediano del blocco di a_i precede, nell'ordinamento dei mediani, α

Allora possiamo dire che $a_i < \alpha$ e lo precede in un eventuale ordinamento.
Allo stesso modo diciamo che se :

1. a_i succedeva il mediano del suo blocco
2. Il mediano del blocco di a_i succede, nell'ordinamento dei mediani, α

Allora $a_i > \alpha$ e lo succede in un eventuale ordinamento.

$< \alpha$	$< \alpha$	$< \alpha$	$< \alpha$	$< \alpha$	$< \alpha$					
$< \alpha$	$< \alpha$	$< \alpha$	$< \alpha$	$< \alpha$	$< \alpha$					
$< \alpha$	$< \alpha$	$< \alpha$	$< \alpha$	$< \alpha$	α	$> \alpha$	$> \alpha$	$> \alpha$	$> \alpha$	$> \alpha$
					$> \alpha$	$> \alpha$	$> \alpha$	$> \alpha$	$> \alpha$	$> \alpha$
					$> \alpha$	$> \alpha$	$> \alpha$	$> \alpha$	$> \alpha$	$> \alpha$

A

Nella figura sono mostrati i vari blocchi ordinati internamente (dall'alto al basso in ordine crescente) ed esternamente (da sinistra a destra per mediano in ordine crescente). Notare che il pivot (α) si trova nella posizione centrale.

Quindi nella ricorsione possiamo scartare almeno $\frac{1}{4}$ (più accuratamente possiamo parlare del 30% [2]) degli elementi e applicare la ricorsione al massimo su $\frac{3}{4}n$.

Quindi le fasi di $SELECT$ sono :

- nel caso base vengono ordinati gli elementi (ad esempio quando ha meno di cinque elementi) e ritorno l'elemento in posizione corretta ⇒ $\Theta(1)$
- altrimenti :
    1. Divido in $\lceil \frac{n}{5} \rceil$ blocchi e ordino i blocchi interamente ⇒ $\Theta(1) * \lceil \frac{n}{5} \rceil$
    2. Se l'ultimo blocco non arriva alla posizione del mediano cerco il massimo ⇒ $\Theta(n)$
    3. Sposto gli elementi mediani in una regione del vettore ⇒ $\Theta(\lceil \frac{n}{5} \rceil)$
    4. Chiamo $SELECT$ ricorsivamente sui mediani in cerca del pivot α ⇒ $T(\lceil \frac{n}{5} \rceil)$
    5. Chiamo $PARTITION$ sul pivot ⇒ $\Theta(n)$
    6. Se la posizione del pivot è quella cercata, ritorno il pivot.
    7. Altrimenti richiamo $SELECT$ sulla parte del vettore (supponiamo di trovarci nel caso peggiore) ⇒ $T(\frac{3}{4}n)$

Quindi, l'equazione di complessità $T(n)$ risulta essere :

$$\begin{cases} \Theta(1) & se\ n < 5 \\ T(\lceil \frac{n}{5} \rceil) + T(\frac{3}{4}n) + \Theta(n) & se\ n \geq 5 \end{cases}$$

Dimostrazione

Limite inferiore :

$$\Omega(n) = \{ \exists n' > 0 \ \exists c > 0 \ : \ \forall n \geq n' \ T(n) \geq cn \}$$

caso base :

$n = 0$

$$T(1) = a \geq c \Rightarrow c \leq a$$

Passo induttivo

$Hyp : \ \forall n < m \ T(n) \geq cn$

$Ts \ : \ n = m \ T(n) \geq cn$

$T(n) = T(\lceil \frac{n}{5} \rceil) + T(\frac{3}{4}n) + bn \geq c\lceil \frac{n}{5} \rceil + c\frac{3}{4}n + bn = c(\lceil \frac{n}{5} \rceil + \frac{3}{4}n) + bn$

$c(\lceil \frac{n}{5} \rceil + \frac{3}{4}n) + bn \geq cn$?

$c(1 - \frac{1}{5} - \frac{3}{4}) \leq b$?

$c \leq 20b$

Limite superiore :

$$O(n) = \{ \exists n' > 0 \ \exists c > 0 \ : \ \forall n \geq n' \ T(n) \leq cn \}$$

caso base :

$n = 0$

$$T(1) = a \leq c \Rightarrow c \geq a$$

Passo induttivo

$Hyp : \ \forall n < m \ T(n) \leq cn$

$Ts \ : \ n = m \ T(n) \leq cn$

$T(n) = T(\lceil \frac{n}{5} \rceil) + T(\frac{3}{4}n) + bn \leq c\lceil \frac{n}{5} \rceil + c\frac{3}{4}n + bn = c(\lceil \frac{n}{5} \rceil + \frac{3}{4}n) + bn$

$c(\lceil \frac{n}{5} \rceil + \frac{3}{4}n) + bn \leq cn$?

$c(1 - \frac{1}{5} - \frac{3}{4}) \geq b$?

$c \geq 20b$

Quindi $T(n) = T(\lceil \frac{n}{5} \rceil) + T(\frac{3}{4}n) + \Theta(n) \in \Theta(n)$

Complessità nel caso di elementi ripetuti

Un comportamento anomalo dell'algoritmo $SELECT$ si verifica quando l'array è composto esclusivamente da elementi ripetuti. In questo caso il pivot risulta essere uno degli elementi ripetuti e, in base all'implementazione di $PARTITION$, il programma può presentare comportamenti molto diversi.

Si prenda in considerazione lo pseudocodice di $PARTITION$ proposto precedentemente.

Se la condizione di scambio implementata è del tipo $A_j \leq pivot$ allora tutti gli elementi del vettore verranno scambiati e l'indice del pivot ritornato dalla procedura risulterà essere l'ultimo elemento dell'array. Questo cambia l'equazione di complessità, che nel caso peggiore diventa $T(n) = T(n-1) + \Theta(n)$, ovvero $T(n) \in O(n^2)$.
Se invece la condizione di scambio è di minoranza stretta, quindi $A_j < pivot$, nessuno degli elementi uguali al pivot verrà scambiato e il pivot risulterà in prima posizione, cambiando l'equazione di complessità di nuovo in $T(n) \in O(n^2)$.

La modifica di Partition

Per risolvere il problema è stato deciso di differenziare gli elementi uguali al pivot dagli altri.
Se un elemento è uguale al pivot allora viene consultato un flag che indica se questo dev'essere spostato nella parte destra o sinistra del pivot. Questo flag viene negato ogni volta che viene letto.
Fa eccezione a questa regola il pivot stesso che vogliamo risulti sempre nella parte sinistra del vettore. Se fosse nella parte destra del vettore, al raggiungimento della sua posizione nel contatore che mantiene la grandezza della parte sinistra, potrebbe essere scambiato con un elemento che risulta essere minore del pivot e posizionato nella parte a destra del vettore. Questo richiederebbe di controllare ad ogni iterazione che il pivot non sia uno dei due elementi (ed eventualmente aggiornarlo) e di modificare lo scambio finale, che sposta il pivot nella posizione corretta, in modo da considerare la partizione in cui si trova il pivot.
In questo modo riusciamo a distribuire equamente gli elementi uguali al pivot nelle due metà del vettore.

Correttezza

$SELECT(k, A, start, end)$ termina.

Dimostrazione :

Se il numero di elementi è minore di 5 allora, per la terminazione di $InsertionSort$ il programma termina.

Altrimenti :

Viene eseguito un ciclo for per ordinare ogni blocco. Le iterazioni sono esattamente $\lceil \frac{end - start}{5} \rceil$.

Per la ricerca del massimo viene eseguito un ciclo for che esegue esattamente $end - start$ iterazioni.

Viene eseguito un while per spostare i mediani di ogni blocco. Ad ogni iterazione l'indice viene incrementato, quindi è riscrivibile come ciclo for. Questo ciclo esegue esattamente $\lceil \frac{end - start}{10} \rceil$ volte (ovvero metà del numero di blocchi).

La chiamata ricorsiva sui mediani dei blocchi termina per ipotesi induttiva, visto che la dimensione dell'input decresce in $\lceil \frac{end - start}{5} \rceil$.

$PARTITION$ termina per la dimostrazione di terminazione vista in classe.

Se la posizione del pivot è corretta il programma termina, altrimenti esegue una ricorsione su una partizione del vettore scartando il pivot. Quindi la dimensione dell'input decresce e, per ipotesi induttiva, queste chiamate ricorsive terminano.

$SELECT(k, A, start, end)$ termina ritornando l'elemento che risulterebbe in posizione $A[start + k]$ in un ordinamento di A.

Dimostrazione per induzione sul numero di elementi :

Caso base : Meno di cinque elementi.

L'array viene ordinato e viene ritornata la posizione $A[start + k]$.

Ipotesi induttiva : La correttezza vale per m elementi.

Tesi induttiva : La correttezza vale per $m + 1$ elementi.

Passo induttivo :

Invariante primo ciclo for : All'$i - esima$ iterazione gli elementi del blocco i sono ordinati.

Dimostrazione : Ad ogni iterazione ordino gli elementi da $(start + i * 5)$ a $(start + (i + 1) * 5)$. Partendo i da 0 alla prima iterazione ordino $A[start, .., start + 5]$, ed alla $k - esima$ iterazione ordino $A[start + k * 5, ..., start + (k + 1) * 5]$, che corrisponde al $k - esimo$ blocco. All'ultima iterazione x ($x = \lceil \frac{end - start}{5} \rceil$) ordino $A[start + (x * 5), .., end]$, considerando che l'ultimo blocco può avere meno di cinque elementi.

Invariante secondo ciclo for : All'i-esima iterazione $maxValue$ contiene il puntatore all'ultima occorrenza del massimo in $A[start, .., i]$.

Dimostrazione : Supponiamo di essere all'iterazione k. Analizzando l'elemento in posizione $A[k]$ possiamo avere tre situazioni :

1. $A[k] < *maxValue$, quindi l'elemento massimo rimane quello puntato dalla variabile.
2. $A[k] = *maxValue$, quindi l'elemento in $A[k]$ è una nuova occorrenza del massimo. Aggiorno il puntatore spostandolo in $A[k]$.
3. $A[k] > *maxValue$, quindi $A[k]$ è un nuovo massimo. Aggiorno il puntatore spostandolo in $A[k]$.

Inizio con $j = 1$, $i = (posizione\ mediano\ del\ blocco\ centrale\ nel\ vettore)$, $m = (numero\ del\ blocco\ centrale)$.

Invariante del ciclo while : $A[i+j]$ contiene il mediano del $(m+j)-esimo$ blocco. $A[i-j]$ contiene il mediano del $(m-j)-esimo$ blocco (se esiste).

Dimostrazione : Alla $k-esima$ iterazione :

a. Controllo se il mediano del blocco $i+m$ esiste.
    i. Se esiste lo sposto in posizione $A[i+k]$.
        1. Se sto spostando il massimo allora aggiorno il suo puntatore. Il massimo potrebbe essere l'elemento $A[i+k]$, ma non l'elemento mediano del blocco $i+m$.
    ii. Se non esiste allora, come spiegato precedentemente, il mediano del blocco $A[i+k]$ è l'ultima occorrenza del massimo. Quindi sposto in $A[i+k]$ questo elemento.
b. Incremento il contatore del numero di mediani a destra di i.
c. Controllo se il mediano del blocco $i-m$ esiste.
    i. Se esiste lo sposto in posizione $A[i-k]$ e incremento il contatore del numero di mediani a sinistra di i.
        1. Se sto spostando il massimo allora aggiorno il suo puntatore. Il massimo potrebbe essere l'elemento $A[i-k]$, ma non l'elemento mediano del blocco $i-m$.

Effettuo la chiamata ricorsiva sulla sezione del vettore dove sono stati spostati i mediani cercando quello intermedio. Per ipotesi induttiva ottengo il mediano dei mediani.

Richiamo $PARTITION$ che, per la correttezza vista in classe, ritorna l'indice della posizione del pivot in un ordinamento del vettore spostando gli elementi minori o uguali del pivot alla sua sinistra e gli elementi maggiori o uguali del pivot alla sua destra.

Se la posizione è corretta termino restituendo il puntatore all'elemento.

Se la posizione non è quella cercata viene effettuata una chiamata ricorsiva in base alla differenza tra la posizione cercata e il pivot su una parte del vettore. La dimensione dell'input termina dato che viene scartato il pivot, dunque, per ipotesi induttiva, questa chiamata termina con successo.

Quindi $SELECT(k, A, start, end)$ ritorna l'elemento in posizione $A[start + k]$.

Mediana pesata tramite ricerca dicotomica

L'algoritmo di mediana pesata applica la ricerca dicotomica dell'elemento utilizzando la metà del peso complessivo come target in modo ricorsivo.
All'inizio della procedura viene calcolato il peso complessivo e il primo target da cercare (ovvero la metà del peso complessivo).
Ad ogni chiamata ricorsiva viene ricercato l'elemento centrale dell'array in un range $[min, max)$ tramite $SELECT$ (all'inizio prenderà il valore $[0, n)$) e, dopo aver richiamato $PARTITION$ sul pivot ottenuto (che si troverà in posizione δ), calcola il peso della partizione precedente al pivot (che chiameremo ε_0) ed il peso che otterrebbe quella partizione aggiungendo il pivot (che chiameremo ε_1).
Le condizioni da rispettare per l'uscita dalla ricorsione sono che ε_0 non abbia raggiunto il target prefissato e invece ε_1 l'abbia raggiunto o lo superi. Nel caso il vettore sia composto da un solo elemento, la mediana pesata è banalmente l'unico elemento presente.
Nel caso di una ricorsione a sinistra del vettore (quindi ε_0 ha superato il target) il range del vettore viene modificato in $[min, \delta)$. Nel caso di ricorsione a destra (quindi ε_0 ha superato il target), il range del vettore diventa $(\delta, max]$ ed al nuovo target viene sottratto ε_1.

Errore nei dati

I numeri reali sono stati rappresentati in notazione double-precision floating-point [3].
Nella notazione macchina non tutti i numeri sono rappresentabili, questo porta un errore sui dati che si ripercuote, in certe situazioni amplificato, nelle operazioni macchina.
Nell'algoritmo di ricerca dicotomica per il calcolo della mediana pesata, le ripetute somme di valori soggetti ad errori possono provocare risultati errati nei confronti per frazioni insignificanti rispetto al valore dei dati.
Ad esempio, nell'implementazione effettuata senza la correzione poi proposta, inserendo i dati di input "0.90 , 7.60 , 6.30 , 6.40 , 5.30 , 2.30 , 2.90 , 2.60 , 1.50 , 7.80 ." nella seconda ricorsione per il calcolo della mediana, il confronto (<) tra gli elementi che compongono la somma della partizione precedente al pivot e il target ha successo, anche se dovrebbe fallire.
Analizzando i dati la somma dei valori risulta essere "11.6" mentre il target vale "11.600000000000001". Questo errore è stato introdotti nei dati ed amplificato nelle operazioni di somma e sottrazione.

La soluzione utilizzata consiste nell'effettuare un downcasting di queste variabili prima dei confronti, rimuovendo errori che appaiono nelle ultime cifre.

Complessità

Sia n la dimensione del vettore.

Gli elementi determinanti nella stima asintotica della complessità dell'algoritmo sono:

1. Il calcolo iniziale del target (eseguito una volta sola), con costo $\Theta(n)$.
2. Le chiamate a $SELECT$ e $PARTITION$, con costo computazionale $\Theta(n)$.
3. La chiamata ricorsiva su un input con dimensione $\frac{n}{2}$.

Quindi l'equazione di complessità risulta essere :

$\Theta(1) \qquad se\ n = 1$

$T(\frac{n}{2}) + \Theta(n) \quad se\ n > 1$

Risoluzione con albero delle chiamate ricorsive :

#	dimensione	costo
1.	dimensione n	costo bn
2.	dimensione $\frac{n}{2}$	costo $b\frac{n}{2}$
...	...	...
i.	dimensione $\frac{n}{2^i}$	costo $b\frac{n}{2^i}$
...	...	…
x.	dimensione 1	costo a

$\frac{n}{2^x} = 1 \ \Leftrightarrow x = log_2(n)$

$$T(n) = \sum_{i=0}^{log_2(n)-1} (b * \frac{n}{2^i}) + a = b * n * \sum_{i=0}^{log_2(n)-1} \frac{1}{2^i} + a = b * n * \sum_{i=0}^{log_2(n)-1} (\frac{1}{2})^i + a$$

La sommatoria risulta essere una serie geometrica convergente ad una costante, che chiamiamo m. Otteniamo quindi che :

$T(n) = b * n * m + a = \Theta(n)$.

Correttezza

La procedura $weightedMedianRec(A, p, q, target)$ termina.

Dimostrazione per induzione sul numero di elementi n :

Caso base : $n = 1$. La procedura ritorna l'unico elemento nel vettore.

Ipotesi induttiva : La procedura termina per n elementi.

Tesi induttiva : La procedura termina per $n + 1$ elementi.

Passo induttivo :

La procedura richiama $SELECT$ cercando l'elemento in posizione $\frac{q-p}{2}$. Per la dimostrazione di terminazione di $SELECT$ questo termina.

L'elemento ottenuto viene utilizzato come perno per la procedura $PARTITION$ che termina per la dimostrazione vista a lezione restituendo la posizione del pivot, che chiameremo α.

Tramite un ciclo for di esattamente $\alpha - p$ iterazioni viene calcolato il peso della partizione a sinistra del pivot e a questo viene aggiunto il peso del pivot.

Se i due pesi calcolati rispecchiano la definizione di mediana pesata allora viene ritornato l'elemento utilizzato come pivot.

Se il pivot non è l'elemento cercato viene effettuata una ricorsione in una delle due parti del vettore, scartando il pivot. Essendo il pivot scartato la dimensione diminuisce e, per ipotesi induttiva, la chiamata ricorsiva termina.

La procedura ricorsiva di supporto $weightedMedianRec(A, p, q, target)$ ritorna il valore $w_k \in A[p, .., q-1]$ che sommato ai precedenti permette di raggiungere o superare il valore specificato in target.

Viene dimostrata la correttezza per induzione sul numero di elementi :

Caso base : $m = 1$ ($q = p + 1$), ovvero è presente un solo elemento.

Il valore $A[p]$ è banalmente il valore della mediana pesata.

Ipotesi induttiva : L'algoritmo ritorna il valore corretto con un numero di elementi $< m$.

Tesi induttiva : L'algoritmo ritorna il valore corretto con un numero di elementi $= m$.

Analizzando i passi eseguiti dall'algoritmo vediamo che :

1. Per la correttezza di $SELECT$ vista in classe otteniamo l'elemento che finirebbe in posizione $\frac{q-p}{2}$ in un ordinamento di A. Chiamiamo questa posizione Ω.
2. Per la correttezza di $PARTITION$ vista in classe partizioniamo il vettore A in base al pivot ottenuto precedentemente, restituendo come valore una posizione in cui finirebbe il pivot in un ordinamento di A.
3. Calcoliamo la somma della partizione sinistra al pivot (ω_0) e a questa ci sommiamo il valore del pivot (ω_1).

4. Se $\omega_0 < target$ e $\omega_0 \geq target$ allora il pivot è il valore corretto.
5. Altrimenti, per effettuare la chiamata ricorsiva confrontiamo i valori ottenuti con il target :
   a. Se ω_0 ha già superato il target allora il valore da ritornare si troverà nella partizione di sinistra. Effettuiamo la chiamata ricorsiva $weightedMedianRec(A, \Omega + 1, q, target)$. Per ipotesi induttiva la chiamata termina correttamente.
   b. Altrimenti significa che ω_1 è ancora minore del target, quindi effettuiamo la chiamata ricorsiva $weightedMedianRec(A, p, \Omega - 1, target - \omega_1)$ che, per ipotesi induttiva, ritorna correttamente. Dal target è importante rimuovere gli elementi della partizione scartata poichè non vengono piú considerati nelle operazioni di somma per ottenere ω_0 e ω_1.

La procedura $weightedMedian(A, size)$ termina.

Dimostrazione :

La procedura esegue un ciclo for con $size$ iterazioni per determinare il peso complessivo del vettore. Viene calcolato il target ricercato dividendo il peso complessivo per due.

Viene richiamata la procedura $weightedMedianRec(A, 0, size, target)$ che, per la dimostrazione di terminazione di $weightedMedianRec$ precedente, termina.

Vogliamo inoltre dimostrare che la procedura $weightedMedian(A, size)$ ritorna il valore $w_k \in A[0, .., size - 1]$ che rappresenta il valore della mediana pesata.

Dimostrazione :

Analizzando i passi di $weightedMedian$ vediamo che :

1. Il target viene calcolato sommando tutti gli elementi del vettore e dividendo il valore per due. Quindi risulta essere la metà del peso complessivo del vettore.
2. Viene ritornato il valore della chiamata $weightedMedianRec(A, p, q, target)$.

Per la dimostrazione precedente la chiamata ricorsiva ritorna il valore $w_k \in A$ tale che la somma degli elementi che lo precedono non raggiunge il target e sommando questo valore, il target viene raggiunto o superato. Essendo il target la metà del peso complessivo, possiamo concludere che il valore ritornato rispecchia la definizione di mediana pesata.

Studio dei tempi

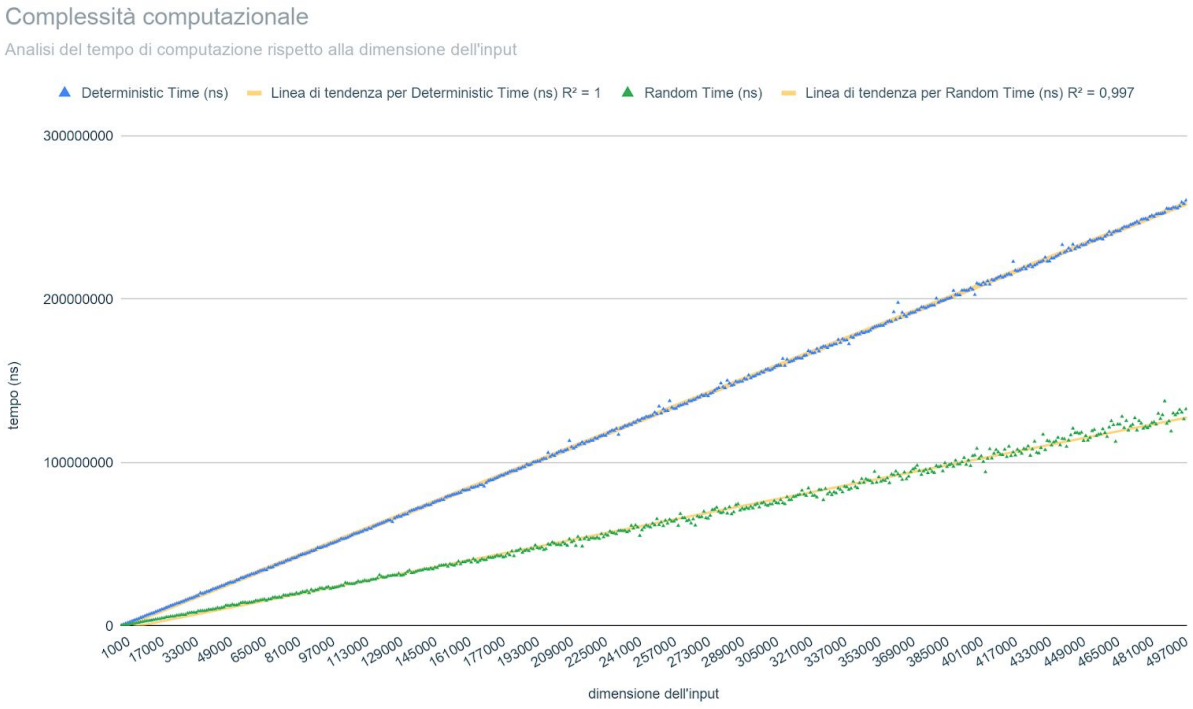
Per studiare i tempi di esecuzione del programma sono stati implementati gli algoritmi descritti negli appunti nel sito personale del docente.

1. Viene calcolata la granularità tramite l'$ALGORITMO\ 4$. Per ottenere il tempo si è utilizzata la funzione C clock_gettime(), una system call virtuale che non passa in spazio kernel, diminuendo i tempi di risposta. Il primo argomento della funzione utilizzato è $CLOCK\_MONOTONIC$, che premette di avere il tempo trascorso da un evento fisso in nanosecondi (come il boot del sistema o l'avvio del programma). Il metodo più accurato (a livello software) per ottenere il tempo consiste nell'accedere al TSC (registro a cui accede il kernel), ma comporterebbe problemi di portabilità ed avrebbe comunque una bassa precisione a causa del mismatch della frequenza di aggiornamento con quella del processore nei sistemi windows. [4]

   Il sistema mediamente ottiene una granularità di $1300 - 1500\ ns$.
2. Viene ricavato $tMin\left(\frac{granularità}{errore\ richiesto}, \text{ dove } errore\ richiesto = 0.02\right)$.
3. Viene eseguito l'$ALGORITMO\ 9$ dove :
   a. Viene calcolato il tempo medio netto tramite l'$ALGORITMO\ 7$ delle procedure:
      i. $PREPARA$: genera i numeri casuali tramite l'$ALGORITMO\ 8$ presente negli appunti. [5]
         1. Il seme utilizzato viene generato casualmente con la funzione rand, il cui seme è inizializzato dal tempo.
         2. Il seme è compreso tra $\{1,\ 2147483646\}$ (Vedi documento nella nota 4).
      ii. P: esegue la ricerca della mediana pesata tramite l'algoritmo spiegato precedentemente.
   b. Si calcola la media dei tempi ottenuti per $C\ (C = 5)$ ripetizioni e l'errore Δ. La procedura termina quando $\Delta < \frac{1}{20}$ del tempo medio ottenuto.

Grafici

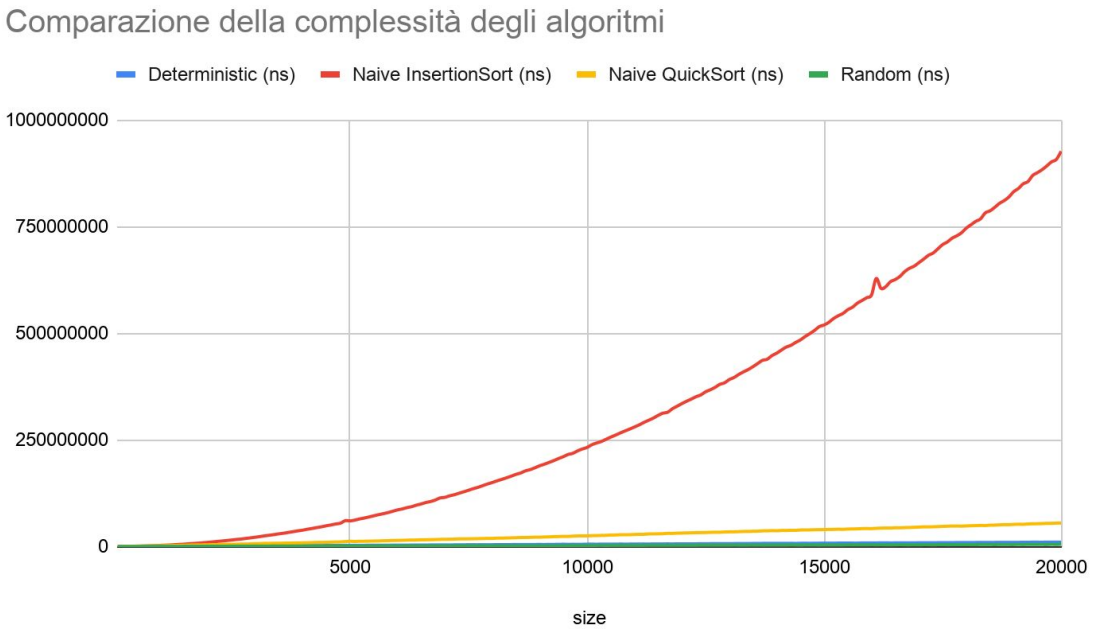
Il seguente grafico mostra il tempo di computazione nell'asse delle ordinate, rispetto alla dimensione dell'input nell'asse delle ascisse, per gli algoritmi di ricerca della mediana pesata con selezione deterministica o selezione pseudo casuale del mediano.
La dimensione dell'input varia da 1000 a $500\,000$ con una distanza di 1000 tra un campione e il successivo.
Nel grafico si possono valutare le linee di tendenza delle misurazioni che risultano lineari, ottenendo un coefficiente di determinazione $R^2 = 1$, che indica un'ottima corrispondenza tra i dati misurati empiricamente e la linea di tendenza che approssima la complessità. [6]
L'algoritmo che seleziona il mediano in maniera pseudo casuale presenta mediamente un andamento lineare nonostante la complessità sia, nel caso peggiore, quadratica. Possiamo notare come questo presenti prestazioni migliori rispetto all'algoritmo di selezione deterministico, date le contenute costanti moltiplicative.



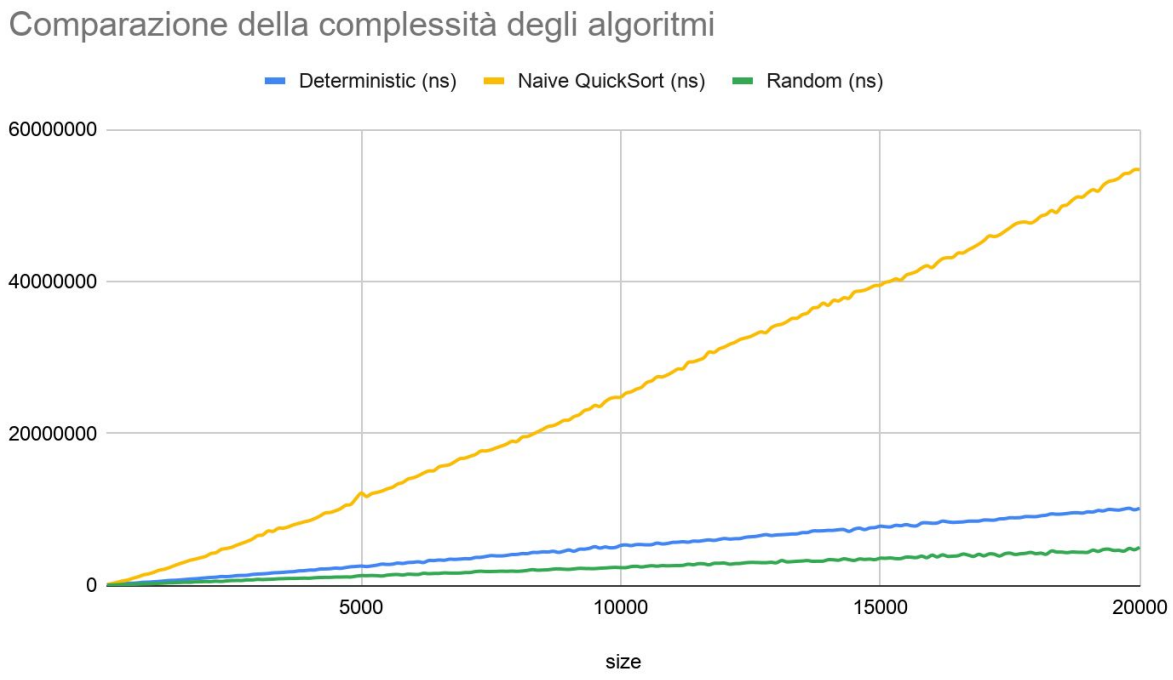
Questo grafico invece compara le due soluzioni del grafico precedente con la soluzione naive implementata tramite $INSERTIONSORT$ ($O(n^2)$) e tramite $QUICKSORT$ ($\Theta(nlogn)$).
Si può notare come l'andamento di $INSERTIONSORT$ rispecchi una funzione quadratica nonostante nel caso migliore la complessità risulti $\Omega(nlogn)$.

I campioni sono stati presi dalla dimensione 100 alla dimensione 20000 con uno spaziamento di 100 tra ogni elemento.

La differenza tra l'algoritmo con selezione deterministica e con selezione pseudo casuale non è purtroppo percettibile con grandezze di input così basse.



Rimuovendo il grafico dell'algoritmo naive implementato con $INSERTIONSORT$ si può notare meglio la differenza tra gli algoritmi che operano mediamente in $\Theta(n)$ e l'algoritmo naive implementato con $QUICKSORT$, che opera in $\Theta(nlog)$.



Note

1. https://stackoverflow.com/questions/50167494/how-to-know-to-which-value-i-should-define-posix-c-source
2. Sia k il blocco in cui si trova il pivot. Abbiamo :
    a. k blocchi in cui due elementi sono minori del pivot
    b. $k-1$ blocchi in cui i mediani sono minori del pivot

    Quindi vengono scartati $2k + (k - 1) = 3k - 1$ elementi.

    Avendo $\frac{n}{5}$ blocchi, nel caso in cui il mediano dei mediani si trovi nel blocco centrale (Ovvero $\frac{n}{10}$) si scartano circa $\frac{n}{5} + (\frac{n}{10} - 1) = \frac{3}{10}n - 1$ elementi.

    La ricorsione viene quindi applicata ad un massimo di $\frac{7}{10}n$ elementi.

3. https://en.wikipedia.org/wiki/Double-precision_floating-point_format
4. Si consigliano le seguenti letture :
    a. https://stackoverflow.com/questions/6498972/faster-equivalent-of-gettimeofday
    b. https://stackoverflow.com/questions/16740014/computing-time-in-linux-granularity-and-precision
5. http://www.firstpr.com.au/dsp/rand31/p1192-park.pdf
6. Per ulteriori informazioni vedere:
    a. https://it.wikipedia.org/wiki/Coefficiente_di_determinazione