Calcolo del maggioritario in tempo lineare

Alessio Orlandi

1 marzo 2010

1 Introduzione

Queste note servono ad illustrare la tecnica di Boyer e Moore per il calcolo del maggioritario in tempo lineare. Il problema è così definito:

Input: Una sequenza di n interi detta A.
Output: Se esiste un valore j tale che almeno $\lfloor n/2 \rfloor + 1$ celle di A sono uguali a j, allora il valore j (il maggioritario esiste) . Altrimenti, risposta negativa (il maggioritario non esiste).

Nella sua formulazione iniziale, il problema riguarda il maggioritario in una elezione politica. Si supponga di avere n persone votanti e m candidati. Ognuno dei votanti esprime preferenza per un solo candidato, e si vuole trovare se esiste una maggioranza. Nel nostro caso, m è potenzialmente grande quanto lo spazio degli interi contenuti in un *int*.

Anche all'epoca, si era alla ricerca di una soluzione veloce, che permetesse di controllare i voti soltanto un numero lineare di volte, esempio usando $2n$ operazioni totali.

2 La soluzione

Boyer e Moore diedero il seguente pseduocodice per calcolare il maggioritario. Esso si divide in due parti, di cui solo la prima è descritta sotto. In una prima fase, si ricerca un *potenziale vincitore* all'interno dei candidati, avente le seguenti caratteristiche: se esiste una maggioranza all'interno di A, il potenziale vincitore ha la maggioranza; se la maggioranza non esiste, allora è un candidato qualunque. In questo modo l'algoritmo richiede una seconda fase di verifica, ovvero una ulteriore scansione dei dati che confermi se il potenziale vincitore ha la maggioranza o meno. Quest'ultima parte è semplice da progettare, mentre la prima richiede un lavoro maggiore.

A grandi linee, la soluzione opera cosí: si mantengono due variabili p e k, la prima che contiene un potenziale vincitore (sostiuito durante la scansione dei dati) e il secondo un contatore che indica quanti elementi sono uguali al vincitore attuale. Inizialmente p e k sono posti a zero. Iterando su ogni elemento della sequenza di voti A, si ottiene una ed una sola delle seguenti situazioni (i indica il passo dell'algoritmo)

1. $k = 0$: $A[i]$ diventa il nuovo potenziale vincitore p e $k = 1$.
2. $k > 0 \wedge A[i] = p$: si dice che p e $A[i]$ *concordano*, e si pone $k = k + 1$.
3. $k > 0 \wedge A[i] \neq p$: si dice che p e $A[i]$ *discordano* e si pone $k = k - 1$.

Al termine, p contiene il potenziale vinicitore. Chiaramente, quest'ultima frase suona misteriosa, in quanto è necessario dimostrare formalmente che l'algoritmo funziona.

3 Correttezza

Siccome la tecnica esposta prevede una seconda fase di verificha dell'effettiva esistenza della maggioranza, è di nostro interesse dimostrare la correttezza della procedura di Sezione 2 solo nel caso in cui tale maggioranza esista sul serio. In particolare, supponiamo ai fini della dimostrazione di conoscere già il valore maggioritario e chiamarlo m. Ad ogni passo i dell'algoritmo, denotiamo con k_i e p_i i valori di k e p al termine di tale iterazione. Inoltre, dato x tale che $0 \leq x < n$, denotiamo con A_x il prefisso della sequenza di voti $A[0]A[1]A[2] \dots A[x]$.

La dimostrazione si basa sul concetto di discordanze e concordanze con un candidato. Siamo interessati a dimostrare una properietà che riguarda i punti *critici*: un'iterazione i è definita critica se $k_i = 0$. In particolare, vogliamo provare che

Lemma 1. *Se $k_i = 0$, allora A_i non ha una maggioranza.*

da cui poi dedurremo

Lemma 2. *Se $k_i > 0$ e A_i ha una maggioranza, allora p_i è il maggioritario di A_i.*

Dimostrazione del Lemma 1. Sia $0 < c_0 < c_1 < \cdots < c_t$ la sequenza di punti critici nell'esecuzione dell'algoritmo, tali che $k_{c_j} = 0$ per ogni j. Procederemo per induzione su di essi, dimostrando quindi che

1. A_{c_0} non ha una maggioranza (caso base).

2. Supponendo che il Lemma valga fino a $j-1$, allora A_{c_j} non ha maggioranza (caso induttivo).

Il caso base è ricavabile dal concetto di discordanza. Se al passo c_0 si ha $k_{c_0} = 0$ vuol dire che durante i primi c_0 passi dell'algoritmo sono stati eseguiti tanti incrementi di k quanti decrementi. Notiamo quindi anche che tutti i punti critici sono posizioni pari. Il tutto implica che per ogni elemento $A[x]$ in A_{c_0} concorde con p_{c_0}, allora ne esiste un altro discorde. Quindi, ci possono essere al più $c_0/2$ elementi uguali a p_{c_0} e altrettanti uguali ad un qualsiasi elemento diverso da p_{c_0}. Ovvero, ogni elemento all'interno di c_0 compare al più $c_0/2$ volte in A_{c_0}. Siccome la maggioranza si ottiene con $c_0/2+1$ elementi almeno, non esiste alcuna maggioranza in A_{c_0}.

Il caso induttivo funziona similarmente. Si supponga di essere al j-esimo passo, per cui si sa che $A_{c_{j-1}}$ non ha alcuna maggioranza (per ipotesi induttiva). Si analizzino ora le posizione che stanno tra c_{j-1} e c_j: esattamente come per il caso base, questa porzione di A non ha alcuna maggioranza. In effetti, sappiamo che nella prima porizione $A_{c_{j-1}}$, non esiste alcun elemento con frequenza maggiore di $c_{j-1}/2$ (altrimenti si avrebbe una maggioranza). Per lo stesso motivo, sappiamo che nella porzione mancante $A[c_{j-1}+1]A[c_{j-1}+2]\cdots A[c_j]$ nessun elemento appare per più di $(c_j - c_{j-1})/2$ volte. Unendo le due considerazioni, se si considera A_{c_j}, nessun elemento può apparire più di $(c_{j-1}/2) + (c_j - c_{j-1})/2 = c_j/2$ volte, e quindi non può esserci alcuna maggioranza.

Per induzione, il Lemma vale ogni volta che $k_i = 0$. □

Utilizzando il risultanto appena dimostrato, possiamo ottenere la dimostrazione del Lemma 2:

Dimostrazione del Lemma 2. Per fissare le idee, supponiamo che i sia maggiore del punto critico c_j. Se k_i è strettamente maggiore di zero, allora vuol dire che, a destra di c_j esistono almeno k_i posizioni in A che contengono il valore p_i. Creiamo ora il seguente array fittizio B: prendiamo A_i e cancelliamo esattamente le k_i posizioni più a destra in cui troviamo p_i. Se ad esempio $p_i = 4$, $k_i = 2$ e $A_i = \{4, 0, 1, 3, 7, 8, 4, 4, 4, 2\}$, allora costruiremo $B = \{4, 0, 1, 3, 7, 8, 4, 2\}$. Ora supponiamo di aver lanciato il nostro algoritmo su B e chiamiamo con x il valore di k al termine dell'esecuzione. E' immediato rendersi conto che, mancando le ultime k_i occorrenze concordanti con p_i, $x = 0$. Quindi, per effetto del Lemma 1, B non ha una maggioranza.

Notiamo ora che B ha lunghezza $i - k_i$, per cui ogni elemento dentro B non può apparire dentro B (e quindi dentro A_i) per più di $\lfloor (i - k_i)/2 \rfloor$ volte.

Per terminare la dimostrazione, notiamo che per assunzione A_i deve avere una maggioranza, ovvero un elemento che compare almeno $\lfloor i/2 \rfloor + 1$ volte. Ma per quanto dimostrato sopra, tutti gli elementi di A_i hanno al più $\lfloor (i - k_i)/2 \rfloor \leq \lfloor i/2 \rfloor$ occorrenze, a meno di p_i, per il quale abbiamo ancora k_i posizioni da reintegrare rispetto a B. Quindi, non possiamo che concludere che p_i sia il maggioritario di A_i. □

In totale, per dimostrare che il nostro algoritmo è corretto basta combinare i Lemmi di cui sopra: per assunzione data, A ha una maggioranza. Quindi, $k_n > 0$, altrimenti per il Lemma 1 si avrebbe una contraddizione. Ma allora, per il Lemma 2, p_n è esattamente l'elemento maggioritario di A_n, ovvero A. □

4 Complessità

La complessità è semplice da calcolare. Durante la fase di ricerca della maggioranza, ogni elemento di A viene considerato soltanto una volta, e quindi vengono eseguite $c'n$ operazioni, per una qualche costante c' che indica il numero di operazioni macchina per elemento. Durante la fase di verifica dell'esistenza della maggioranza, si scandisce il vettore A conteggiando gli elementi uguali a p_n, considerandoli ognuno una volta soltanto. Anche in questa seconda fase, quindi, vengono eseguite $c'n$ operazioni, per una qualche altra costante c''.

In totale, quindi, esiste una costante $c \geq c' + c''$ per cui vengono eseguite al più cn operazioni per l'esecuzione dell'algoritmo. In altre parole, la dipendenza tra n e il tempo di esecuzione dell'algoritmo è lineare.

Anche dal punto di vista spaziale si può ricorrere a una simile misurazione, per cui lo spazio dell'algoritmo è nuovamente lineare.