

Supponiamo di dover trovare, in una tabella di grande dimensioni, tutti gli elementi che si ripetono più di un certo numero di volte. Per esempio, vogliamo trovare i prodotti visualizzati più frequentemente su Amazon, oppure le parole cercate più frequentemente su Google. Questo problema prende il nome di ricerca degli *heavy hitters* e, in astratto, richiede di trovare in una tabella di n interi tutti gli interi che si ripetono almeno n/k volte, dove $n \gg k$. Si noti che ci possono essere al più k heavy hitters e potrebbe non essercene neanche uno.

Partiamo da una versione più semplice di questo problema: vogliamo trovare nella tabella un numero che si ripete almeno $n/2$ volte sapendo che ce n'è uno. Chiaramente, questo valore deve corrispondere alla mediana di tutti i valori nella tabella e posso trovarlo in tempo $\mathcal{O}(n)$ con un algoritmo deterministico. Vediamo ora un semplicissimo algoritmo ad hoc che trova tale valore scandendo l'array una sola volta dall'inizio alla fine e usando una memoria ausiliaria sublineare (algoritmi di questo tipo si chiamano *streaming*).

Algorithm 1 (Boyer-Moore)

Require: Array A

```

1:  $c \leftarrow 0$                                 ▷ inizializza contatore maggioranza
2:  $v \leftarrow \text{NULL}$                           ▷ inizializza maggioranza corrente
3: for  $i = 1, \dots, n$  do
4:   if  $c = 0$  then                               ▷ nessuna maggioranza
5:      $v \leftarrow A[i]$ 
6:      $c \leftarrow c + 1$ 
7:   else if  $A[i] = v$  then                         ▷ incremento maggioranza corrente
8:      $c \leftarrow c + 1$ 
9:   else                                           ▷ decremento maggioranza corrente
10:     $c \leftarrow c - 1$ 
11:   end if
12: end for

```

Non è difficile verificare che quando l'algoritmo termina il valore corrente di v corrisponde al valore di maggioranza nella tabella. Ora ci chiediamo se esista una soluzione streaming anche per il problema di trovare gli heavy hitters. In realtà è possibile dimostrare che non esiste un algoritmo streaming che risolve il problema di ricerca degli heavy hitters con memoria ausiliaria sublineare (dimostrazione omessa).

Per riuscire a trovare una soluzione streaming, rilassiamo il problema originario introducendo una versione approssimata. Nel problema di ricerca di heavy hitters ε -approssimati (indicato ε -HH) abbiamo una tabella A di lunghezza n e due parametri k e ε . L'algoritmo deve produrre una lista di valori tale che:

1. ogni valore che compare in A almeno $\frac{n}{k}$ volte è nella lista,

2. ogni valore nella lista compare almeno $\frac{n}{k} - \varepsilon n$ volte in A .

La soluzione che proporremo userà memoria ausiliaria $\Theta(\frac{1}{\varepsilon})$. Questo evita di produrre una contraddizione col risultato di impossibilità quando $\varepsilon \rightarrow 0$.

Per risolvere il problema ε -HH utilizzeremo una struttura dati probabilistica chiamata count-min sketch. Questa struttura supporta due operazioni: $\text{Inc}(x)$ che incrementa il contatore associato al valore x e $\text{Count}(x)$ che ritorna il numero di volte che $\text{Inc}(x)$ è stato invocato. La struttura dati è composta da ℓ tabelle hash ciascuna di dimensione b . Siano $h_1 \dots, h_\ell : \{1, \dots, n\} \rightarrow \{0, \dots, b-1\}$ le funzioni hash associate alle ℓ tabelle. Ogni tabella hash comprime la tabella di n elementi in una di dimensione $b \ll n$. Le ℓ tabelle diverse servono a ridurre la probabilità di errore dovuto a collisione. Il codice per le due operazioni è estremamente semplice.

Algorithm 2 Count-Min Sketch

```

1: Crea matrice CMS[ $\ell$ ][ $b$ ]
2: procedure Inc( $x$ )
3:   for  $i = 1, \dots, \ell$  do
4:     Incrementa CMS[ $i$ ][ $h_i(x)$ ]
5:   end for
6: end procedure
7: procedure Count( $x$ )
8:   Ritorna  $\min_{i=1, \dots, \ell} \text{CMS}[i][h_i(x)]$ 
9: end procedure

```

Sia x un valore che compare almeno una volta nella tabella A e sia n_x il numero di occorrenze di x in A . Dato che $b \ll n$ ci saranno delle collisioni, ovvero $h(x) = h(y)$ con $x \neq y$. Questo significa che

$$n_x \leq \text{CMS}[i][h_i(x)] \quad i = 1, \dots, \ell.$$

Infatti $\text{Inc}(x)$ verrà chiamata esattamente n_x volte, ma —a causa delle collisioni— due chiamate $\text{Inc}(x)$ e $\text{Inc}(y)$ tali che $h_i(x) = h_i(y)$ incrementeranno lo stesso contatore. Quindi, dato che ogni $\text{CMS}[i][h_i(x)]$ sovrastima n_x è sensato utilizzare come valore di $\text{Count}(x)$ la più piccola di tali stime.

Analizziamo ora la probabilità di errore del count-min sketch quando le funzioni hash $h_1 \dots, h_\ell$ sono estratte a caso e in modo indipendente da una famiglia universale \mathcal{H} di funzioni hash. Usiamo la notazione H_1, \dots, H_ℓ per indicare che le funzioni sono variabili casuali opportunamente definite. Dato x siano Z_1, \dots, Z_ℓ le variabili casuali $Z_i = \text{CMS}[i][H_i(x)]$ dove la probabilità è rispetto all'estrazione di H_i da \mathcal{H} . Allora

$$Z_i = n_x + \sum_{y \neq x} n_y \mathbb{I}\{H_i(y) = H_i(x)\}.$$

Ora, dato che \mathcal{H} è una famiglia universale,

$$\mathbb{P}(H_i(x) = H_i(y)) \leq \frac{1}{b} \quad i = 1, \dots, \ell.$$

Quindi,

$$\mathbb{E}[Z_i] = n_x + \sum_{y \neq x} n_y \mathbb{P}(H_i(y) = H_i(x)) \leq n_x + \sum_{y \neq x} \frac{n_y}{b} \leq n_x + \frac{n}{b}.$$

Introduciamo le variabili casuali non negative $X_i = Z_i - n_x$. Scegliendo $b = \frac{\varepsilon}{e}$ abbiamo che $\mathbb{E}[X_i] \leq \frac{\varepsilon n}{e}$. Applicando la disuguaglianza di Markov alle X_i otteniamo quindi

$$\mathbb{P}\left(Z_i \geq n_x + e \frac{\varepsilon n}{e}\right) = \mathbb{P}\left(X_i \geq e \frac{\varepsilon n}{e}\right) \leq \frac{1}{e}.$$

Ora, dato che le funzioni hash H_1, \dots, H_ℓ sono indipendenti, anche le Z_1, \dots, Z_ℓ sono indipendenti e quindi, in particolare, gli eventi $Z_i \geq n_x + \varepsilon n$ ($i = 1, \dots, \ell$) sono indipendenti. Questo implica che

$$\mathbb{P}(\text{Count}(x) \geq n_x + \varepsilon n) = \mathbb{P}\left(\min_{i=1, \dots, \ell} Z_i \geq n_x + \varepsilon n\right) = \prod_{i=1}^{\ell} \mathbb{P}(Z_i \geq n_x + \varepsilon n) \leq e^{-\ell}.$$

Per capire i prossimi passaggi ricordiamo che, per qualsiasi insieme di eventi A_1, \dots, A_N vale che

$$\mathbb{P}(\exists i : A_i) = \mathbb{P}(A_1 \cup \dots \cup A_N) \leq \sum_{i=1}^N \mathbb{P}(A_i).$$

Dato che vogliamo conteggi corretti con alta probabilità per ogni x nella tabella A di lunghezza n ,

$$\begin{aligned} \mathbb{P}(\exists x \in A : \text{Count}(x) \geq n_x + \varepsilon n) &= \mathbb{P}\left(\bigcup_{x \in A} (\text{Count}(x) \geq n_x + \varepsilon n)\right) \\ &\leq \sum_{x \in A} \mathbb{P}(\text{Count}(x) \geq n_x + \varepsilon n) \\ &\leq n e^{-\ell} \\ &\leq \delta \end{aligned}$$

per $\ell \geq \ln \frac{n}{\delta}$.

Quindi, se fissiamo $\delta = 0,01$ e scegliamo $\varepsilon = \frac{1}{2k}$, $b = 2ek$, $\ell = \Theta(\log n)$ e includiamo nella lista ogni elemento x per cui $\text{Count}(x) \geq \frac{n}{k}$ abbiamo che:

1. ogni valore che compare almeno $\frac{n}{k}$ volte in A è nella lista,
2. con probabilità almeno del 99%, ogni valore nella lista compare almeno $\frac{n}{2k}$ volte in A .

Lo spazio totale utilizzato è $\Theta(k \log n)$, ovvero logaritmico nella taglia della tabella A .