Complementi di Algoritmi e Strutture Dati

Conteggio approssimato

Docente: *Nicolò Cesa-Bianchi* versione 1 aprile 2025

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

$c \leftarrow 0$ ▷ inizializza contatore maggioranza
$v \leftarrow$ NULL ▷ inizializza maggioranza corrente
for $i = 1, \ldots, n$ **do**
  if $c = 0$ **then** ▷ nessuna maggioranza
    $v \leftarrow A[i]$
    $c \leftarrow c + 1$
  else if $A[i] = v$ **then** ▷ incremento maggioranza corrente
    $c \leftarrow c + 1$
  else ▷ decremento maggioranza corrente
    $c \leftarrow c - 1$
  end if
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 con $\frac{1}{n} < \varepsilon < \frac{1}{k}$. 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.

L'algoritmo che proponiamo è probabilistico e usa memoria ausiliaria $\Theta\big(\frac{\ln n}{\varepsilon}\big)$.

Per risolvere il problema ε-HH utilizzeremo una struttura dati probabilistica chiamata count-min sketch. Questa struttura supporta due operazioni: $\textsc{Inc}(x)$ che incrementa il contatore associato al valore x e $\textsc{Count}(x)$ che ritorna il numero di volte che $\textsc{Inc}(x)$ è stato invocato. La struttura dati è composta da ℓ tabelle hash ciascuna di dimensione b. Siano $h_1 \ldots, h_\ell : \{1, \ldots, n\} \to \{0, \ldots, b-1\}$ 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 e per la routine principale SelectEl è estremamente semplice.

Algorithm 2 Count-Min Sketch

```
Crea matrice CMS[ℓ][b]
procedure Inc(x)
    for i = 1, ..., ℓ do
        Incrementa CMS[i][h_i(x)]
    end for
end procedure
procedure Count(x)
    Ritorna min_{i=1,...,ℓ} CMS[i][h_i(x)]
end procedure
procedure SelectEl(A, k)
    Crea lista vuota
    for t = 1, ..., n do
        Leggi il prossimo elemento x_t = A[t]
        Esegui Inc(x_t)
        if Count(x_t) ≥ n/k then
            Aggiungi x_t alla lista (solo se x_t non è già nella lista)
        end if
    end for
    Ritorna la lista
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

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

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

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

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

Quindi,

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

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

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

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

$$\begin{aligned}
\mathbb{P}\left(\textsc{Count}(x) \geq n_x + \varepsilon n\right) &= \mathbb{P}\left(\min_{i=1,\ldots,\ell} Z_i \geq n_x + \varepsilon n\right) \\
&= \prod_{i=1}^{\ell} \mathbb{P}\left(Z_i \geq n_x + \varepsilon n\right) \\
&= \mathbb{P}\left(\bigwedge_{i=1,\ldots,\ell} \Big(Z_i \geq n_x + \varepsilon n\Big)\right) \\
&\leq e^{-\ell} \ .
\end{aligned}$$

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

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

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

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

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

Quindi, se fissiamo $\delta = 0{,}01$, abbiamo che $b = \Theta\left(\frac{1}{\varepsilon}\right)$ e $\ell = \Theta(\log n)$. Quindi lo spazio totale utilizzato è $\Theta\left(\frac{1}{\varepsilon}\log n\right)$, ovvero logaritmico nella taglia della tabella A se ε non dipende da n.[1] La routine SelectEl soddisfa le seguenti proprietà:

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 $\frac{n}{k} - \varepsilon n$ volte in A.

[1] In realtà dobbiamo anche contare lo spazio utilizzato dalla lista che contiene gli heavy hitters. Questo sarà di ordine $\mathcal{O}\big(k/(1-\varepsilon k)\big)$.