

Hierarchical hash tree o *HH-tree*

Luigi Foscari

Si vuole creare una struttura che permette le operazioni di inserimento, cancellazione e ricerca su una o più chiavi¹. I valori da inserire nella struttura appartengono ad un insieme \mathcal{X} per cui esistono

- k feature map $f_i : \mathcal{X} \mapsto \mathcal{H} \quad i = 1, \dots, k$ che estraggono le chiavi dai dati.
- Un intero b .
- Una famiglia di hash universale H tale che $\forall h \in H, h : \mathcal{H} \mapsto \{0, \dots, b-1\}$.

La struttura è organizzata come un albero, le foglie sono liste di collisione, chiamate anche *bucket*, i nodi sono una tripla composta da una tabella di dimensione b , una chiave e una funzione di hash. Nelle celle della tabella sono presenti puntatori ai figli. La struttura è inizializzata con una singola foglia.

La procedura di inserimento per un elemento $x \in \mathcal{X}$ in un nodo y , inizialmente la radice, avviene nel seguente modo

- Se y è una foglia, x è aggiunto in testa alla lista.
- Se y è un nodo interno con feature map f e funzione di hash h , si ripete l'inserimento ricorsivamente nell'elemento puntato dalla posizione $h(f(x))$ della tabella.

Quando una foglia raggiunge un limite massimo m viene **scissa**, l'operazione di scissione controlla tutti gli elementi nel bucket e trova la combinazione di funzione di hash e feature map che descrive meglio l'insieme; viene poi creata una tabella di foglie di taglia b e ogni elemento x del bucket è inserito nella foglia puntata dalla posizione $h(f(x))$ della tabella.

Se $m = b$ allora in fase di scissione è possibile inserire ogni elemento in una foglia differente. Ciò non fornisce tuttavia garanzie sugli inserimenti successivi e implica costruire una funzione di hash ad hoc ad ogni scissione, che è un'operazione più costosa di prelevare una funzione dalla famiglia.

L'intera struttura dati può fare a meno delle feature map se si utilizzano funzioni di hash che hanno come dominio \mathcal{X} , ma questa opzione è meno interessante

¹Intese come valori che caratterizzano i dati, non per forza uniche, sono ammessi i duplicati. Esiste un nome migliore?

perché l'estrazione di feature permette di operare in modo più espressivo sui dati. Esempio pratico: l'universo contiene informazioni su libri, una feature che si è scelto di estrarre è la data di pubblicazione, si possono quindi cercare tutti i testi usciti nel medesimo anno in tempo efficiente.

Senza scendere nei dettagli la struttura dati dell'albero è espressa ricorsivamente in questo modo (notazione ML)

```
type 'a tree =  
  | Leaf of 'a list  
  | Node of 'a tree array
```

Ricerca

La ricerca in questo albero può avvenire per zero o più chiavi, nel caso in cui non ci siano chiavi si tratta di una visita dell'albero.

Nel caso in cui sia presente almeno una chiave è possibile, nei nodi che la comprendono, calcolare il valore della funzione di hash per quella chiave e continuare la ricerca solo nel sottoalbero indicato da questa posizione.

Nel caso in cui sono presenti tutte le chiavi il tempo di ricerca è pari al tempo di inserimento, più la ricerca lineare nella lista di collisione, che però avrà taglia limitata.

Scissione

Dato un bucket di elementi bisogna trovare la combinazione di funzione di hash e feature map tra quelle a disposizione che ottiene i valori più sparsi in una tabella, è possibile utilizzare l'indice di Gini normalizzato² I , indichiamo con x i valori nel campione (in questo caso il risultato dell'applicazione di f su ogni elemento del bucket) e con ϕ le frequenze relative:

$$\{x_1, \dots, x_m\} \rightarrow \{\phi_1, \dots, \phi_t\} \quad I = \left(1 - \sum_{i=1}^t \phi_i\right) \frac{t}{t-1}$$

Un algoritmo ingenuo che utilizza questo indice è il seguente

²Un altro indice di dispersione è l'eterogeneità, ma a causa dei logaritmi è più difficile da calcolare, ulteriori considerazioni vanno fatte

```

Input:  $B$  bucket,  $f_1, \dots, f_k$  feature map,  $H$  famiglia universale di hash
Output: Una feature map
Sia  $M$  una matrice  $k \times |B|$ 
Estrai una funzione  $h$  di hash a caso da  $H$ 
for  $i = 0, \dots, k - 1$  do
    for  $j = 0, \dots, |B| - 1$  do
         $M[i][j] \leftarrow h(f_i(B_j))$ 
    end
end
Sia  $G$  un array di taglia  $k$ 
for  $i = 0, \dots, k - 1$  do
     $G[i] \leftarrow$  indice di Gini normalizzato della  $i$ -esima riga di  $M$ 
end
if  $G$  non è valido then
    Ripeti l'algoritmo dall'inizio
end
 $r \leftarrow$  indice dell'elemento più grande di  $G$ 
return  $f_r$ 

```

La scelta tiene conto anche della funzione di hash perché se valutassi solo la feature map ci sarebbe il rischio che la funzioni di hash (anche se poco probabile) incanali l'input sempre nella stessa foglia. La funzione di hash è scartata se G non soddisfa una serie di requisiti, ad esempio

- Deve esistere almeno un valore diverso da 0.

Prestazioni

Definiamo un HH-tree T con

- Dimensione massima iniziale della lista di collisione m .
- k feature map f_1, \dots, f_k calcolabili in tempo costante.
- Una famiglia di funzioni di hash universali H di taglia $\geq k$ calcolabili in tempo costante e un intero b che indica la dimensione delle tabelle di hash.

Il tempo di inserimento o cancellazione di un elemento $x \in \mathcal{X}$ è O grande della profondità dell'albero, perché nel caso peggiore l'inserimento è fatto nella foglia più lontana dalla radice, ma questa ultima operazione è a tempo costante e il calcolo degli indici $h(f(x))$ è sempre costante.

Una metrica spesso utilizzata per valutare queste strutture dati è il numero di accessi necessari per effettuare una ricerca, nel caso in cui ci siano tutte le chiavi il numero di accessi è pari alla profondità dell'albero. Essendo le tabelle

contenitori di soli puntatori, si potrebbero inserire una accanto all'altra nello stesso blocco o in pochi blocchi e lasciare le liste di collisione in un altro blocco.

Bisogna capire come si evolve la profondità dell'albero in base al numero di elementi inseriti n e i parametri m e b .

Prossimi passi

- Fare una ricerca in letterature di strutture dati simili e fare dei paragoni. Ad esempio
 - B-albero
 - Tabella di hash
 - ?
- Analizzare le metriche utilizzate da queste strutture. Ad esempio (considerati in media)
 1. Numero di accessi per una ricerca con tutte le chiavi.
 2. Numero di accessi per un inserimento.
 3. Dimensione delle liste di collisione.
 4. ?
- Definire un dataset e valutare nel seguente modo
 1. Definire un sottoinsieme delle permutazioni del dataset³.
 2. Per ogni elemento di questo sottoinsieme inserirlo per intero in un HH-tree e valutare le metriche.
 3. Ripete l'intero procedimento per diversi valori di m e b .

³Per un dataset abbastanza grande il numero di permutazioni sarebbe enorme