Hierarchical hash tree o *HH-tree*

Luigi Foscari

Questo documento è una bozza della struttura dati.

Si vuole creare una struttura che permette le operazioni di inserimento, cancellazione e ricerca su una o più chiavi[1]. 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, ad ogni nodo corrisponde una lista di collisione, chiamata anche *bucket*, oppure una tripla composta da una tabella di dimensione b, una chiave e una funzione di hash. Nelle celle della tabella sono presenti puntatori a nodi figli. La struttura è inizializzata con un singolo nodo composto da solo un bucket.

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

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

Se un bucket in un nodo t raggiunge un limite massimo m viene **scisso**, l'operazione di scissione controlla tutti gli elementi nel bucket e trova la chiave che descrive meglio l'insieme; scelta una feature map f viene estratta una funzione di hash dalla famiglia H e creata una tabella di taglia b, ogni elemento x nel bucket è inserito nella tabella in posizione $h(f(x))$.

In fase di inizializzazione viene deciso un valore m, per ogni nodo figlio m è incrementato di 1. Questo per evitare il caso in cui ci siano scissioni infinite.

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

[1] Intese come valori che caratterizzano i dati, non per forza uniche, sono ammessi i duplicati

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 chiave tra quelle a disposizione che ottiene i valori più sparsi in una tabella, è possibile utilizzare l'indice di Gini normalizzato[2] I, indichiamo con x i valori nel campione (in questo caso il risultato di $f(x)$ per ogni x nel bucket) e con ϕ le frequenze relative:

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

Un algoritmo ingenuo che utilizza questo indice è il seguente

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

Input: B bucket, $f_1, \ldots, f_k$ feature map
Output: Una feature map
Sia M una matrice $k \times |B|$
for $i = 0, \ldots, k-1$ **do**
  for $j = 0, \ldots, |B|-1$ **do**
    $M[i][j] \leftarrow f_i(B_j)$
  end
end
Sia G un array di taglia k
for $i = 0, \ldots, k-1$ **do**
  $G[i] \leftarrow$ indice di Gini normalizzato della i-esima riga di M
end
$r \leftarrow$ indice dell'elemento più grande di G
return f_r

Prestazioni

Definiamo un HH-tree T con

- Dimensione massima iniziale della lista di collisione m.
- k feature map $f_1, \ldots, f_k$ 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.