

# Analisi dell'HH-tree

Luigi Foscari

## 1 Parametri e metriche

I parametri della struttura dati sono

- $m$  la dimensione massima delle foglie.
- $b$  la taglia delle tabelle di hash nei nodi.
- $n$  il numero di elementi inseriti (spesso sconosciuto a priori).

Le metriche analizzate sono

- *Usage* o utilizzo è la percentuale  $\in [0, 1]$  di riempimento delle foglie rispetto ad  $m$ .
- *Depth* è la profondità  $\in \mathbb{N}$  dell'albero.

## 2 Relazione tra i parametri e le metriche

Le seguenti analisi hanno mostrato come gli unici valori *sensati* per  $m$  e  $b$  sono quelli minori di  $n$  e che conviene allontanarsi da valori troppo bassi per entrambi.

È necessario distinguere due implementazioni, nella prima, chiamata *rigida*, il parametro  $m$  rimane inalterato in tutta la struttura, nella seconda, chiamata *adattiva*,  $m$  è incrementato di 1 per ogni figlio, quindi una foglia a profondità  $t$  avrà una lunghezza massima  $m + t$ .

### 2.1 Implementazione rigida

L'implementazione in questione presenta una condizione sui parametri, nel caso in cui  $b = 1$  e  $m < n$  è inevitabile una ricorsione infinita alla prima scissione, perchè non è possibile inserire tutti gli elementi in un solo bucket, ma non è neanche possibile utilizzare più di un bucket. Perciò  $b > 1$  o, meno efficientemente,  $m \geq n$ .

- Riguardo a  $b$  possiamo dire che
  - Se  $b = 2$  l'albero è binario, considerando che l'hashing è universale il valore atteso di utilizzo delle foglie è  $1/2$ , la profondità è
$$\log_2 \left( \frac{m}{2} + 1 \right) - 1.$$
  - Se  $b \geq n$  in valore atteso la profondità è 1 e l'utilizzo è  $n/b$ .
- Riguardo ad  $m$  possiamo dire che
  - Se  $m = 1$  ogni foglia contiene al massimo un elemento, quindi in valore atteso l'utilizzo è  $1/b$  e la profondità è al massimo  $n$ .
  - Se  $m = n$  l'utilizzo è sempre uguale a  $m/b$ , mentre la profondità è sempre 1.

## 2.2 Implementazione adattiva

- Riguardo a  $b$  possiamo dire che
  - Se  $b = 1$  l'utilizzo è sempre 1, perchè c'è una scissione ad ogni inserimento dopo l' $m$ -esimo, mentre la profondità è  $n - m + 1$ .
  - Se  $b = n$  l'utilizzo ha un valore atteso pari a  $n/m$  e la profondità 1, ipotizzando che l'hashing utilizzato sia universale.
- Riguardo ad  $m$  possiamo dire che
  - Al crescere di  $m$  da 1 ad  $n$  la profondità e l'utilizzo tendono a scendere.
  - Dal momento in cui  $m = n$  non avvengono scissioni, perciò la profondità sarà sempre 1 e l'utilizzo al 100%.

## 3 Valutazione struttura dati

Le seguenti valutazioni valgono per l'implementazione rigida. Consideriamo inoltre  $n > m$ , per cui la radice è sempre un nodo e ha  $b$  figli.

Consideriamo una struttura dati con solo  $n = m + 1$  elementi quindi è presente solo la radice e  $b$  foglie. Una foglia è scissa, e quindi diventa nodo, se contiene più di  $m$  elementi. La probabilità che in un inserimento una foglia venga scelta rispetto ai suoi fratelli è  $1/b$ , quindi il numero di scissioni è descritto bene da una variabile aleatoria binomiale  $X \sim B(1/b, n)$ . Abbiamo che

$$E[X] = \frac{n}{b}$$

Generalizzando se consideriamo un nodo e le sue  $b$  foglie vuote, dopo i primi  $n$  inserimenti c'è da aspettarsi di avere effettuato  $n/b$  scissioni.

Per la radice questo vale senza eccezioni, però per un nodo a profondità maggiore  $n$  è diverso, perchè bisogna considerare solo i valori che effettivamente lo raggiungono. In particolare un nodo a profondità  $l$  ha probabilità  $1/b^l$  di essere selezionato, rispetto a tutti i nodi al suo livello e a quelli sopra. Quindi in generale al livello  $l$  dopo  $n$  inserimento bisogna aspettarsi  $n/b^l$  scissioni, chiamiamo queste variabili aleatorie  $X_l$ .

Se definiamo  $S$  la variabile aleatorie che descrive il numero di scissioni in base a  $b$ , il numero totale di scissioni dopo  $n$  inserimenti è

$$E[S] = \sum_{l=0}^{\infty} E[X_l] = \sum_{l=0}^{\infty} \frac{n}{b^l} = n \sum_{l=0}^{\infty} \left(\frac{1}{b}\right)^l = \frac{n}{1 - 1/b}$$

Ma questo non ha senso perchè al crescere di  $n$  il numero di scissioni dovrebbe calare. C'è un errore. Ad esempio per  $b = 2$  devo aspettarmi  $2n$  scissioni per ogni  $n$  elementi inseriti, che non ha senso.

Proviamo a non usare la binomiale, ma l'ipergeometrica, dopo  $n$  inserimento voglio che almeno  $m$  siano stati effettuati in una foglia precisa, e questo avviene con probabilità  $1/b$ , quindi  $X \sim H(b, 1, n)$ . Questa rappresentazione è analoga perchè  $E[X] = n/b$ .

## 4 Strutture dati analoghe

Un HH-tree con  $k$  mappe di feature su un universo  $\mathcal{X}$  permette di fare ricerca su più campi, espressi come feature degli elementi di  $\mathcal{X}$ . Vediamo ora due strutture dati naïve che permettono di effettuare le stesse operazioni. Assumiamo sempre di avere a disposizione le mappe di feature per estrarre i valori per le ricerche.

### 4.1 Forma lineare

Gli elementi inseriti  $S \subseteq \mathcal{X}$  sono conservati in una lista  $A$ . La ricerca, come la cancellazione, è effettuata elemento per elemento, calcolando i valori necessari per verificare se l'elemento è quello che si cerca e richiede tempo  $O(|S|)$ . L'inserimento è costante  $O(1)$ . Non c'è una grande differenza tra la ricerca su una singola chiave o su molteplici.

## 4.2 Tabelle di hash

Gli elementi inseriti  $S \subseteq \mathcal{X}$  sono conservati in un array  $A$  e sono create  $k$  tabelle di hash di taglia  $b$  con bucket, ad ognuna è associata una delle mappe.

- L'inserimento di un valore  $x \in \mathcal{X}$  è effettuato aggiungendo all'array  $x$  e per ognuna delle tabelle di hash calcolare la posizione di  $x$  e inserire nella lista di collisione la posizione di  $x$  all'interno dell'array.
- La ricerca su un insieme di chiavi è definita nel seguente algoritmo

**Data:**  $A$  insieme degli elementi,  $C$  chiavi,  $T$  tabelle  
**Output:** Risultato della ricerca su  $c$  chiavi  
Sia  $R$  una copia di  $A$   
**for**  $i = 1, \dots, |T|$  **do**  
    **for**  $j = 1, \dots, |C|$  **do**  
        Rimuovi da  $R$  ogni elemento che non compare in  $T[i][j]$   
    **end**  
**end**  
**return**  $R$

- La cancellazione di effettua rimuovendo l'indice dell'elemento da ogni tabella di hash e l'elemento da  $A$ .

Quindi l'inserimento è lineare su  $k$ , mentre la ricerca con un insieme di chiavi  $C$  è  $O(|T||C| + n) = O(|T|k + n)$ <sup>1</sup>. Lo spazio occupato è invece  $O(n + |T|n)$ . Inoltre l'inserimento occupa spazio  $O(n)$ .

---

<sup>1</sup> $|C| \leq k$