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 mostrano 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 prodondità t avrà una lunghezza massima $m + t$.

2.1 Implementazione rigida

L'implementazione in question presenta una condizione sui parametri, nel caso in cui $b = 1$ e $m < n$ è invevitable 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, $m \geq n$.

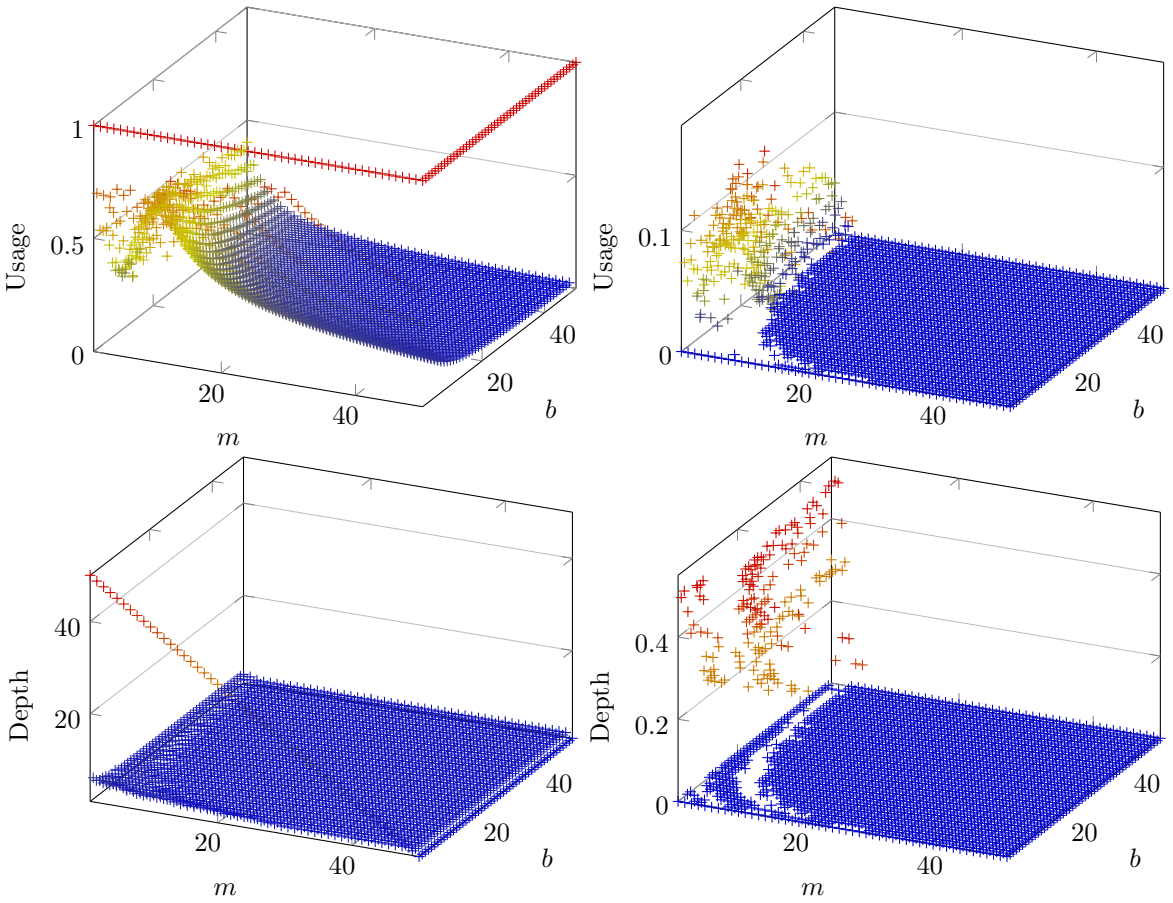
- Riguardo a b possiamo dire che
  - Se $b = 2$ l'albero è binario, considerando che l'hashing è universale il valore atteso di utilizzo delle foglie è $1/2$, La profondità è
    $$\log_2 \left(\frac{m}{2} + 1 \right) - 1.$$
  - Se $b \geq n$ in valore atteso la profondità è 1 e l'utilizzo è n/b.
- Riguardo 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.

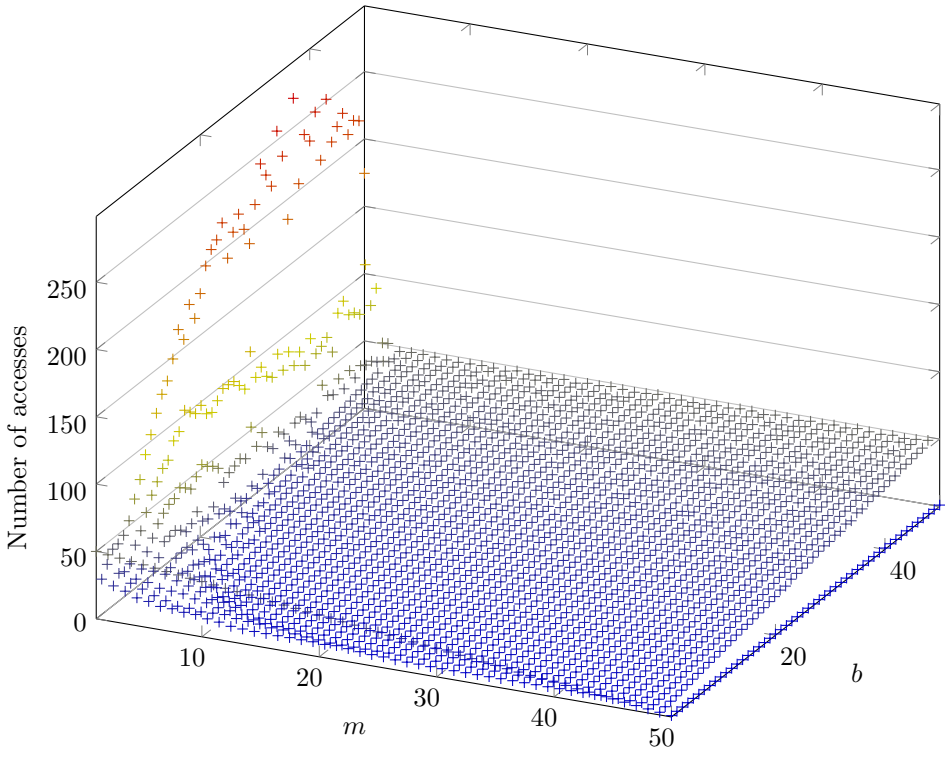
2.2 Implementazione adattiva

- 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 MAGIC Gamma Telescope Dataset

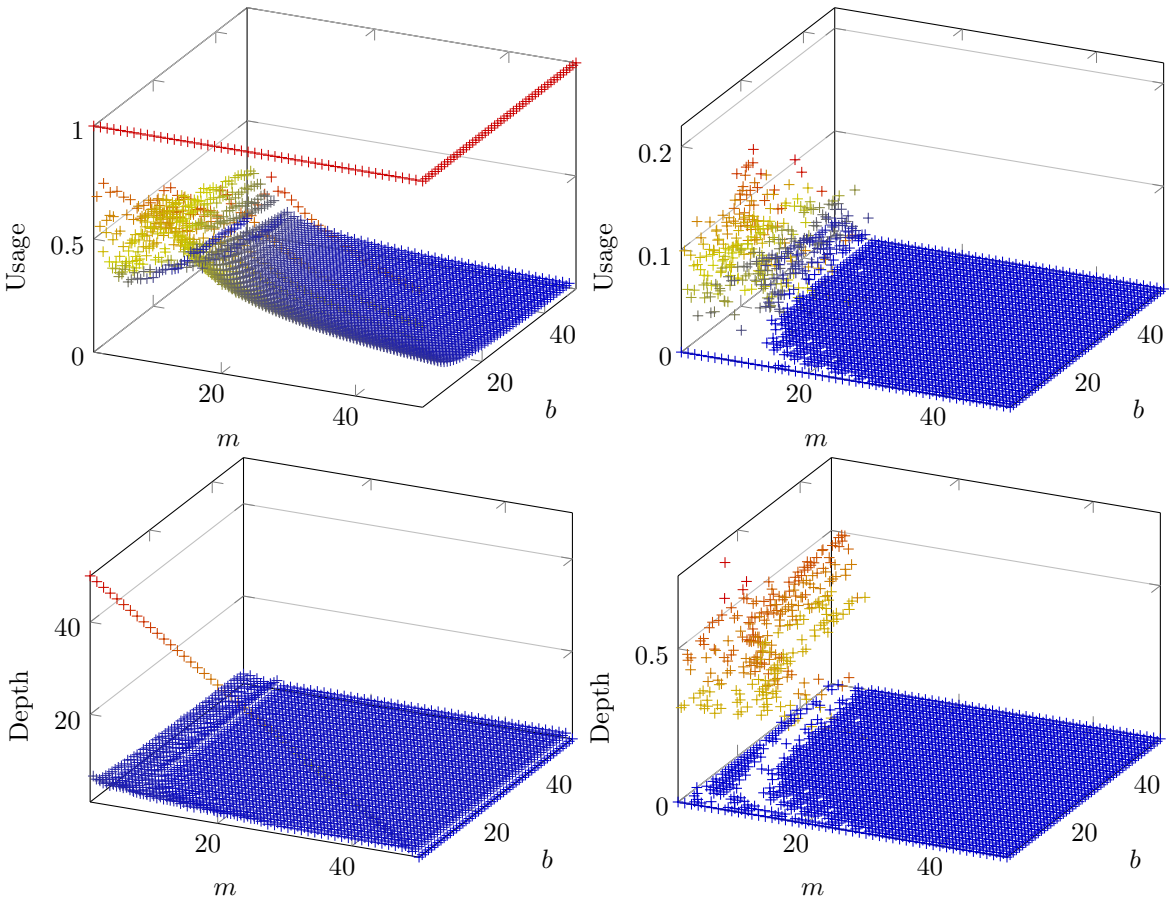
È stato utilizzato un subset di $n = 50$ righe del `MAGIC Gamma Telescope Dataset`. Per valori di m e b tra 1 e n sono stati calcolati utilizzo e profondità medi su 10 permutazioni. Questi sono i risultati.

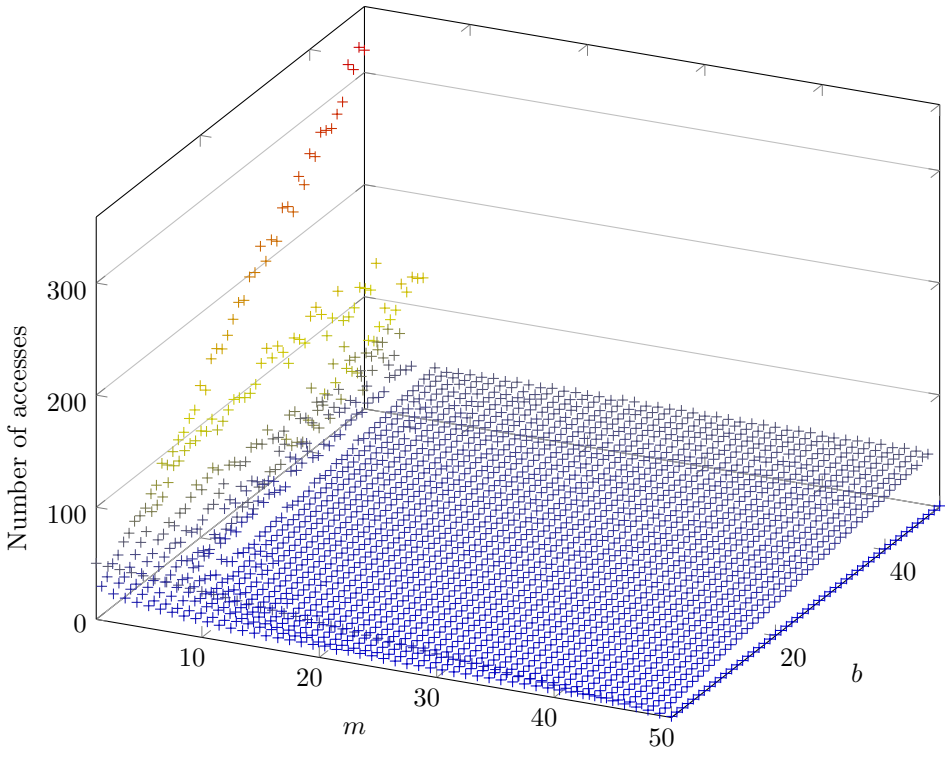




4 `Cloud DataSet`

È stato utilizzato un subset di $n = 50$ righe del `Cloud DataSet`. Per valori di m e b tra 1 e n sono stati calcolati utilizzo e profondità medi su 10 permutazioni. Questi sono i risultati.





5 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 $\mathcal{X}$. È possibile ottenere una struttura dati quasi analoga tramite l'utilizzo di molteplici tabelle di hash: gli elementi inseriti $S \subseteq \mathcal{X}$ sono conservati in un array A, sono create inoltre k tabelle di hash di taglia b, ad ognuna è associata una mappa.

- 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 $i = 1, \ldots, |T|$ **do**
  for $j = 1, \ldots, |C|$ **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 $O(|T||C| + n) = O(|T|k + n)$[1]. Lo spazio occupato è invece $O(n + |T|n)$. Inoltre l'inserimento occupa spazio $O(n)$.

[1] $|C| \leq k$