14. Tabelle Hash: Re-hashing

Abbiamo visto che nelle tabelle hash il numero di confronti di chiavi per effettuare una ricerca o un inserimento dipende dal *fattore di carico* $\alpha = n/m$, definito come rapporto tra il numero n di elementi memorizzati nella tabella e la capacità m della tabella stessa.

Ad esempio, se le collisioni vengono gestite utizzando la scansione quadratica o l'hashing doppio, il numero medio di passi che vengono effettuati per la ricerca nel caso di chiave non presente nella tabella è $\frac{1}{1-\alpha}$ (per $\alpha < 1$). Se $\alpha = 1/2$ (tabella piena al 50%) tale numero è 2, se $\alpha = 3/4$ (tabella piena al 75%) tale numero è 4. È importante notare che, a differenza delle altre tecniche utilizzate per i dizionari, in cui il numero di passi dipende da n (ad esempio $\Theta(\log n)$ per array ordinati o alberi bilanciati), in questo caso il numero di passi dipende da quanto è piena la tabella. Osserviamo inoltre che se la tabella è "molto piena", cioè α si avvicina a 1, il numero medio di passi per trovare un elemento cresce notevolmente.

Per avere buone prestazioni, oltre a scegliere una funzione hash che distribuisca in maniera uniforme le chiavi sugli indici, è dunque importante evitare che la tabella si riempia troppo. A tale scopo si può fissare un valore massimo α_{max} per il fattore di carico e sostituire la tabella con una tabella più grande ogni volta che α supera α_{max}, in modo da mantenere costante (in media) il numero di passi per un inserimento o una ricerca.

La sostituzione della tabella con una nuova tabella, detta *re-hashing*, è un'operazione dispendiosa in termini di tempo. Occorre inoltre una funzione hash adatta alla nuova tabella. Per risolvere questa seconda questione, la funzione hash può essere definita in modo parametrico rispetto alla dimensione della tabella, ad esempio potrebbe essere della forma:

$$h(x) = f(x) \bmod m$$

dove f è una funzione che restituisce un numero molto grande[1] e m è la dimensione della tabella. Variando la dimensione della tabella è sufficiente cambiare il parametro m per ottenere la nuova funzione.

Per quanto riguarda lo spostamento degli elementi dalla tabella vecchia a quella nuova, occorre scandire l'intera vecchia tabella e inserire ogni suo elemento nella nuova tabella, calcolandone la posizione in base alla nuova funzione hash e risolvendo eventuali collisioni. Pertanto il *re-hashing* richiede un numero di passi pari almeno alla dimensione della vecchia tabella, che deve essere scandita interamente.

[1] Anch'essa è una funzione hash, in Java si potrebbe usare il metodo `hashCode` fornito dalle librerie.

Il presente materiale integra *ma non sostituisce* il libro di testo consigliato.
Data pubblicazione: 20 dicembre 2019

Sebbene il re-hashing appaia molto dispendioso in termini di tempo, se effettuato in maniera appropriata ha un costo ammortizzato basso, come mostreremo ora. In particolare, supponiamo di procedere come segue:

- Fissiamo il valore massimo α_{max} del fattore di carico a 1/2.
- Ogni volta che il fattore di carico raggiunge α_{max}, al momento del successivo inserimento effettuiamo il re-hashing sostituendo la tabella con una nuova tabella di capacità doppia.

Immaginiamo ora di avere una tabella iniziale T_0 di dimensione m e di effettuare una serie di inserimenti, sostituendo, quando necessario, una tabella T_i con una tabella T_{i+1} mediante re-hashing, secondo la strategia indicata sopra. L'evoluzione delle tabelle è riportata nel seguente schema:

tabella	T_0	$\rightarrow$	T_1	$\rightarrow$	T_2	$\rightarrow$	$\cdots$	$\rightarrow$	T_k
capacità	m		$2m$		2^2m		$\cdots$		2^km
numero max elementi	$m/2$		m		$2m$		$\cdots$		$2^{k-1}m$

Calcoliamo il numero totale di operazioni di inserimento che si effettuano a causa dei re-hashing. A tale scopo, osserviamo che quando si effettua il re-hashing passando da una tabella T_i alla tabella T_{i+1}, tutti gli elementi presenti in T_i, che sono $2^{i-1}m$ (metà della capacità di T_i) devono essere inseriti in T_{i+1}. Sommando per tutte le operazioni di re-hashing otteniamo il numero totale di inserimenti dovuti a re-hashing, fino ad arrivare ad utilizzare la tabella T_k:

$$\sum_{i=0}^{k-1} 2^{i-1}m = m\sum_{i=0}^{k-1} 2^{i-1} = \frac{m}{2}\sum_{i=0}^{k-1} 2^i = \frac{m}{2}\left(2^k - 1\right) \tag{14.1}$$

Supponiamo ora che, partendo da T_0 inizialmente vuota, siano stati fatti in totale N inserimenti di elementi, effettuando, quando necessario, re-hashing.[2] Per contenere N elementi, la tabella finale T_k corrisponde al più piccolo intero k tale che $2^{k-1}m \geq N$, cioè tale che $k \geq \log_2 \frac{N}{m} + 1$. Pertanto $k = \lceil \log_2 \frac{N}{m} \rceil + 1 < \log_2 \frac{N}{m} + 2$. Sostituendo in (14.1) otteniamo che il numero totale di inserimenti dovuti al re-hashing è minore di:

$$\frac{m}{2}\left(2^{\log_2 \frac{N}{m}+2} - 1\right) = \frac{m}{2} \cdot 4 \cdot \frac{N}{m} - \frac{m}{2} < 2N \tag{14.2}$$

Da (14.2) possiamo dunque concludere che se effettuiamo N operazioni di inserimento in una tabella hash, a causa del re-hashing effettuiamo in totale $O(N)$ ulteriori inserimenti. Se ogni operazione di inserimento utilizza un numero di passi costante, il numero di passi totali tenendo conto anche del re-hashing è $O(N)$. Pertanto, dividendo per il numero N di inserimenti "effettivi", otteniamo che il tempo ammortizzato è $\frac{O(N)}{N} = O(1)$. Pertanto, anche effettuando il re-hashing. nel modo indicato sopra, il costo medio delle operazioni di inserimento resta costante.

[2] Con N indichiamo gli inserimenti effettivi, cioè senza contare quelli dovuti al re-hashing.