Azzolini Riccardo 2019-04-17

Tabelle hash

1 Hash statico con indirizzamento aperto

I dati vengono inseriti direttamente nella tabella, che deve quindi avere dimensione M maggiore o uguale al numero n di dati attesi.

Siccome a ogni indirizzo può essere memorizzato un solo dato, bisogna determinare come gestire le collisioni, cioè cosa fare se

- Insert(x): la posizione $H(x.chiave)$ è occupata;
- Member(x), Delete(x): la posizione $H(x.chiave)$ contiene $y \neq x$.

1.1 Strategie di gestione delle collisioni

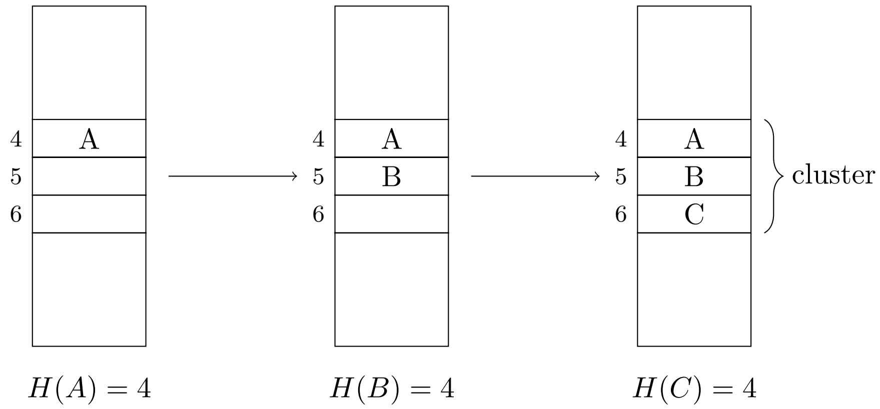
In caso di collisione, vengono sondate altre posizioni della tabella, $h_0(c), h_1(c), \dots$, dove

$$h_i(c) = \langle H(c) + F(i) \rangle_M$$

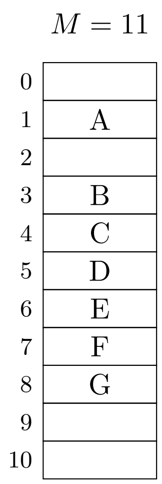
Esistono varie strategie, che si differenziano per la scelta della funzione F:

- **scansione lineare**: $F(i) = i$;
- **scansione quadratica**: $F(i) = i^2$;
- **hash doppio**: $F(i) = i \cdot H_2(c)$
  (in questo caso, $H(c)$ prende il nome di *hash primario*, mentre $H_2(c)$ si dice *hash secondario*).

La scansione lineare è semplice, ma tende a produrre **cluster** (sequenze di posizioni occupate) di grandi dimensioni, con conseguente degrado delle prestazioni perché ogni operazione può dover scorrere un intero cluster. Ad esempio:



La scansione quadratica e l'hash doppio tendono invece a formare cluster più frammentati (cioè più numerosi, ma di dimensioni minori), però non è garantito che trovino una posizione libera, anche se la tabella non è piena, perché non vengono scandite tutte le posizioni. Ad esempio:



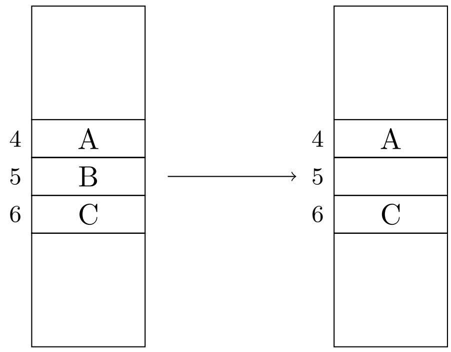
Inserimento di X: $H(X) = 3$
Scansione quadratica:
$3, 4, 7, 1, 8, 6, 6, 8, 1, 7, 4, 3, 4, 7, \ldots$

1.1.1 Ricerca ed eliminazione

Per le operazioni di ricerca ed eliminazione, si esegue la scansione (con la strategia scelta) finché non si trova l'elemento cercato o una posizione vuota, che indica invece l'assenza dell'elemento.

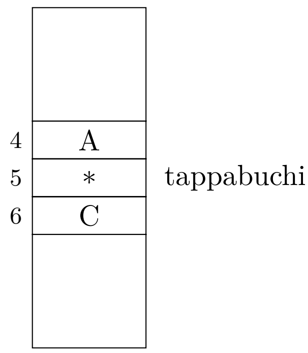
Sorge però un problema: quando si elimina un elemento all'interno di un cluster, viene creata una posizione libera, che rende irraggiungibili gli elementi successivi.

Ad esempio, in questo caso, con scansione lineare e $H(A) = H(B) = H(C) = 4$, dopo aver cancellato B non è più possibile reperire C perché la scansione si arresta in posizione 5:



Tale problema ha due possibili soluzioni:

- far scorrere indietro gli elementi successivi del cluster dopo l'eliminazione;
- ricorrere alla *cancellazione logica*, inserendo al posto dell'elemento rimosso un "tappabuchi", che fa risultare libera la posizione in caso di inserimento, ma non provoca l'arresto della ricerca.



1.2 Prestazioni

Il costo delle operazioni dipende da:

- il *fattore di carico* $\alpha = \frac{n}{M}$ ($0 \leq n \leq M \implies 0 \leq \alpha \leq 1$, con $\alpha = 0$ se la tabella è vuota e $\alpha = 1$ quando è piena);
- la dimensione dei *cluster*;
- la gestione delle collisioni.

Teorema: Con la scansione lineare, il numero medio di sondaggi eseguiti da una ricerca in una tabella con fattore di carico α è:

- $\min\left(M,\ \dfrac{1}{2}\left(1 + \dfrac{1}{1-\alpha}\right)\right)$ in caso di *successo*;

- $\min\left(M,\ \dfrac{1}{2}\left(1+\dfrac{1}{(1-\alpha)^2}\right)\right)$ in caso di *insuccesso*;

α	$\frac{1}{2}$	$\frac{2}{3}$	$\frac{3}{4}$	$\frac{9}{10}$
successo	1.5	2.0	3.0	5.5
insuccesso	2.5	5.0	8.5	55.5

Teorema: Con l'hash doppio, il numero medio di sondaggi eseguiti da una ricerca in una tabella con fattore di carico α è:

- $\min\left(M,\ \dfrac{1}{\alpha}\ln\left(\dfrac{1}{1-\alpha}\right)\right)$ in caso di *successo*;
- $\min\left(M,\ \dfrac{1}{1-\alpha}\right)$ in caso di *insuccesso*;

α	$\frac{1}{2}$	$\frac{2}{3}$	$\frac{3}{4}$	$\frac{9}{10}$
successo	1.4	1.6	1.8	2.6
insuccesso	1.5	2.0	3.0	5.5

La scansione quadratica ha prestazioni intermedie.

Siccome, nel caso medio, il numero di sondaggi effettuati non è proporzionale alla dimensione della tabella o al numero di dati contenuti, la complessità in media è $O(1)$.

Indicativamente, mantenendo $\alpha \le \frac{3}{4}$ si ottiene un buon compromesso tra spazio occupato e prestazioni. Comunque, α deve sicuramente essere inferiore a $\frac{9}{10}$, altrimenti il costo di ciascuna operazione si avvicina a quello delle liste concatenate.

2 Hash dinamico con concatenazioni separate

Si crea inizialmente una tabella di piccole dimensioni. Poi:

- quando, in seguito a un inserimento, si ha $\alpha = 1$ (cioè ogni lista contiene in media 1 elemento), si crea una nuova tabella di dimensione $2M$ (cioè doppia), vi si trasferiscono i dati, e si elimina la tabella precedente;
- quando, con una cancellazione, il fattore di carico scende ad $\alpha = \frac{1}{2}$, si trasferiscono i dati in una nuova tabella di dimensione $\frac{M}{2}$.

In questo modo, è possibile mantenere un fattore di carico ottimale, e quindi ottenere buone prestazioni, senza bisogno di una stima della quantità di dati.

Osservazione: Quando cambia la tabella, cambia anche la funzione di hash, dato che si ha un range di indirizzi diverso.

2.1 Prestazioni

Il costo **ammortizzato** di n inserimenti è $O(1)$ *per ciascuno*, cioè $\Theta(n)$ in totale.

Dimostrazione: Si considera una tabella di dimensione iniziale M. L'inserimento di ciascuno dei primi $M-1$ dati ha costo $O(1)$. Poi, all'inserimento numero M, si crea una nuova tabella di dimensione $2M$ e vi si copiano gli M dati, con costo complessivo M.

All'inserimento numero 2^iM si passa da una tabella di dimensione 2^iM a una di dimensione $2^{i+1}M$, con costo 2^iM.[1] L'ultima creazione di una tabella avviene all'inserimento numero 2^kM, con k tale che:

$$2^{k-1}M < n \le 2^kM \implies k = \left\lceil \log_2 \frac{n}{M} \right\rceil$$

Il costo totale è allora:

$$M + \sum_{i=0}^{\lceil \log_2 \frac{n}{M} \rceil} 2^iM \le M + M \sum_{i=0}^{\lceil \log_2 n \rceil} 2^i \approx M(1+2n) = \Theta(n)$$

2.2 Scelta delle soglie di ridimensionamento

In alcuni casi, se si alternano inserimenti ed eliminazioni, ogni operazione può avere costo $\Theta(M)$:

1. si effettua un inserimento, dopo il quale α diventa 1, e perciò viene creata una tabella grande il doppio, avente quindi $\alpha = \frac{1}{2}$;
2. se, subito dopo, si elimina un dato, viene ricreata immediatamente una tabella con le dimensioni precedenti e il fattore di carico torna ad $\alpha = 1$.

Per risolvere questo problema è sufficiente cambiare le soglie alle quali si crea una nuova tabella, in modo che non sia più sufficiente una singola eliminazione a causare un ridimensionamento dopo un inserimento (e viceversa): ad esempio, si può dimezzare la tabella quando $\alpha = \frac{1}{4}$, invece che $\frac{1}{2}$.

Lo stesso ragionamento permette anche l'implementazione di *hash dinamici con indirizzamento aperto*: siccome bisogna evitare assolutamente di raggiungere $\alpha = 1$, si raddoppia la dimensione quando (ad esempio) $\alpha = \frac{3}{4}$.

[1] Il costo $2^{i-1}M$ degli inserimenti che portano, ogni volta, il fattore di carico da $\frac{1}{2}$ a 1 si può ignorare, essendo minore del costo di trasferimento degli elementi alla tabella successiva, 2^iM.