

# Tavole di hash, una breve introduzione

Andrea Burattin

## 1 Tavole di hash

Una tavola di hash è una struttura dati che richiede memoria proporzionale al numero di chiavi presenti contemporaneamente nel dizionario. Ogni chiave  $k$  viene memorizzata in una delle  $m$  celle tramite una **funzione di hash**.

Dato che  $|U| > m$  (con  $U$  insieme elementi da mappare) esisteranno coppie di chiavi distinte la cui funzione punta alle medesime celle. In questo caso si ha una **collisione**. Le collisioni sono un effetto inevitabile nelle tavole di hash, è quindi necessario prevedere un meccanismo di gestione di queste situazioni.

### 1.1 Risoluzione delle collisioni con liste

Con questa tecnica, detta anche di chaining, la tavola è formata da puntatori a liste di valori. L'operazione di *Insert* viene eseguita in tempo  $O(1)$ ; *Search* richiede tempo proporzionale alla lista  $T[h(k)]$  e *Delete* richiede una *Search* +  $O(1)$ .

La *Search* richiede tempo  $O(n)$  nel caso che tutti i valori siano nella stessa lista (caso pessimo). La sua complessità si calcola in funzione del fattore di carico  $\alpha$ :

$$\alpha = \frac{n}{m} \quad 0 \leq \alpha \leq \frac{|U|}{m} \quad \text{dato che } 0 \leq n \leq |U|$$

**Definizione 1 (Hash uniforme semplice)** ogni chiave da inserire nella tavola di hash viene estratta casualmente da  $U$ . Inoltre, la funzione di hash, sia tale che ogni chiave abbia probabilità  $1/m$  di andare in una qualsiasi delle  $m$  celle.

### 1.2 Proprietà data l'ipotesi di hash uniforme semplice

- La lunghezza media di una lista è:

$$E[n_j] = \frac{1}{m} \sum_{i=1}^m n_i = \frac{n}{m} = \alpha$$

- La ricerca di una chiave **non presente** richiede tempo  $\Theta(\alpha + 1)$ ; la ricerca di una chiave **presente** richiede tempo  $\Theta(\alpha + 1)$
- Se  $n \leq cm$  per qualche  $c > 0$  allora  $\alpha = \Theta(1)$  e  $\Theta(\alpha + 1) = \Theta(1)$ .

## 2 Funzioni hash

Una buona funzione hash dovrebbe soddisfare l'ipotesi di hash uniforme semplice. Purtroppo questa dipende dalla distribuzione degli elementi e questa è quasi mai nota. Alcune metodologie per la generazione di funzioni hash vengono riportate di seguito.

## 2.1 Metodo della divisione

$$h(k) = k \bmod m + 1$$

Questa tecnica non funziona molto bene nel caso si prenda  $m = 2^p$ , quindi potenza di 2. È quindi bene prendere numeri primi molto distanti da  $2^p$ .

## 2.2 Metodo della moltiplicazione

$$h(k) = \lfloor m(kA \bmod 1) \rfloor + 1$$

$$0 < A < 1 \quad x \bmod 1 = x - \lfloor x \rfloor \quad (\text{parte frazionaria di } x)$$

La scelta di  $m$ , in questo caso, non è critica, funziona bene per qualsiasi valore di  $A$ , anche se per ragioni storiche si tende a preferire l'inverso del rapporto aureo:  $A = \frac{\sqrt{5}-1}{2}$ .

## 2.3 Randomizzazione di funzioni hash

Dato che nessuna funzione hash può evitare le collisioni, si può pensare di prenderne una da un insieme in maniera casuale. Questo approccio viene detto **hash universale**.

Un insieme di funzioni si dice universale se, per ogni coppia di chiavi  $k$  ed  $l$ , vi sono al più  $\frac{|H|}{m}$  funzioni tali che  $h(k) = h(l)$ , dove  $H$  è l'insieme delle funzioni ed  $m$  è il numero di chiavi.

In questa situazione, la lunghezza attesa della lista  $h(k)$  è al più  $\alpha$  se  $k$  non è presente, altrimenti è  $\alpha + 1$ .

### 2.3.1 Costruzione di un insieme universale di funzioni di hash

Sia  $1 \leq a \leq p$ ,  $0 \leq b \leq p$  con  $p$  preso come numero primo maggiore di ogni chiave  $K$ , allora un insieme universale di funzioni hash è identificato dalla seguente:

$$h_{a,b}(k) = ((aK + b) \bmod p) \bmod m$$

## 3 Risoluzione delle collisioni con indirizzamento aperto

Con questa tecnica, la funzione di hash, non individua una singola cella, ma una lista di celle. L'inserimento avviene nella prima cella libera incontrata durante l'ispezione. Per la realizzazione delle liste di ispezione ci sono tre tecniche, vediamole.

### 3.1 Ispezione lineare

Si ottiene dalla funzione  $h(k, i) = (h'(k) + i) \bmod m$ . L'ispezione inizia dalla cella  $h'(k) + 0$  e prosegue prossimamente su  $h'(k) + 1, h'(k) + 2, \dots, h'(k) + n$ . È quindi facile da implementare, tuttavia provoca molto rapidamente fenomeni di addensamento primario, ovvero il fenomeno per cui sequenze di celle già popolate tendono ad allungarsi.

### 3.2 Ispezione quadratica

Si ottiene dalla funzione  $h(k, i) = (h'(k) + c_1i + c_2i^2) \bmod m$ , con  $c_1$  e  $c_2$  costanti e  $c_2 \neq 0$ . A chiavi uguali corrispondono sequenze di ispezioni uguali, se  $h'(k) = h'(l)$ . Provoca addensamento secondario, dovuto al fatto che il valore iniziale determina la sequenza di ispezione. Per questa ragione ce ne sono solo  $m$  distinte.

### 3.3 Hash doppio

Si ottiene da due funzioni hash  $h_1$  e  $h_2$  ponendo  $h(k, i) = (h_1(k) + ih_2(k)) \bmod m$ . Affinché la sequenza di ispezione percorra tutta la tavola,  $h_2(k)$  deve essere relativamente primo con  $m$ .

Con l'hash doppio abbiamo  $\Theta(m^2)$  sequenze distinte e quindi bassi fenomeni di addensamento, in definitiva, questa tecnica è la più buona.

### 3.4 Proprietà dell'indirizzamento aperto

Il numero di celle ispezionate nella ricerca di una chiave

- **presente** è:  $\frac{m+1}{2}$  se  $\alpha = 1$ , al più  $\frac{1}{\alpha} \ln\left[\frac{1}{1-\alpha}\right]$  se  $\alpha < 1$ ;

- **non presente** è:  $m$  se  $\alpha = 1$ , al più  $\frac{1}{1-\alpha}$  se  $\alpha < 1$ ;

Conseguenza: il numero medio di celle ispezionate per inserire una nuova chiave è  $m$  se  $\alpha = 1$ , al più  $\frac{1}{1-\alpha}$  se  $\alpha < 1$ .

#### 3.4.1 Tavola valori in funzione di $\alpha$

$\alpha$	$1/\alpha \ln[1/(1-\alpha)]$
0.3	1.19
0.5	1.39
0.7	1.72
0.9	2.56
0.95	3.15