Soluzione Hash Table

Liste di collisione + Dizionario

> Si consideri una **tabella di hash con liste di collisione** che implementa un dizionario D in cui le chiavi sono stringhe di 4 cifre decimali che indichiamo con c1, c2, c3, c4 e i valori sono stringhe.
> La tabella di hash ha 7 bucket indicizzati da 0 a 6 e la funzione di hash è **h: (c1+c2*c3+c4) mod 7**.
>
> **1. (senza voto)**: scrivete sul foglio della risposta le cifre 0 0 7 0 5 0, poi la vostra data di nascita nel formato ANNO (4 cifre) MESE (2 cifre) GIORNO (2 cifre), poi la vostra matricola universitaria (7 cifre), poi la vostra matricola universitaria al contrario (7 cifre), e infine le cifre 0 1 7 0 [...]
>
> **2 (senza voto, [...])**: disegnate sul foglio della risposta uno schema come quello mostrato sotto [...] :
>
> | 0070 | 5020 | 0001 | 0112 | 3456 | 7765 | 4321 | 0170 |
> |---|---|---|---|---|---|---|---|
> | ma | co | gra | pe | le | re | ca | giu |
> | hash(0070) | Hash(5020) | Hash(0001) | ... | ... | ... | ... | ... |
>
> **3 (1/3 del punteggio):** Il dizionario D implementato dalla tabella di hash è inizialmente vuoto. **Disegnate** la tabella di hash che si ottiene dagli inserimenti delle stringhe “ma”, “co”, “gra”, “pe”, “le”, “re”, “ca”, “giu” associate alle chiavi su quattro cifre così come le avete calcolate al Passo 2, [...]. **Ricordate che in un dizionario non si possono ammettere chiavi duplicate**: nel caso ce ne fossero ignorate l'inserimento del duplicato e scrivete esplicitamente sul foglio “questa coppia chiave-valore è stata ignorata perché la chiave è duplicata”.
>
> **4 (1/3 del punteggio):** quali sono le due principali “buone proprietà” che una tabella di hash deve avere? Rispetto alle chiavi che avete ottenuto al Passo 2, la funzione di hash **h** applicata alle chiavi ottenute al Passo 2 ha queste due “buone proprietà” ? Motivate esaurientemente la risposta.
>
> **5 (1/3 del punteggio):** sotto l'ipotesi che la funzione di hash abbia tutte le “buone proprietà” che dovrebbe avere, qual è la complessità dell'operazione di **cancellazione** di un elemento data la sua chiave nel caso peggiore? [...]

1. Preparazione dati:

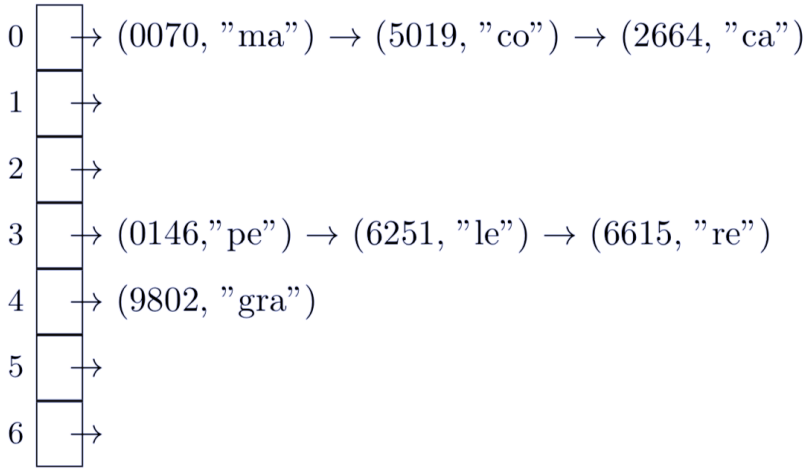
007050 19980201 4662516 6152664 0170

2. Calcolo funzione hash:

`h(x) = (c1 + c2 * c3 + c4) mod 7` quindi nel primo caso abbiamo: `h(0 + 0 * 7 + 0) mod 7 = 0 mod 7 = 0`

0070	5019	9802	0146	6251	6615	2664	0170
ma	co	gra	pe	le	re	ca	giu
hashVal = 0	hashVal = 0	hashVal = 4	hashVal = 3	hashVal = 3	hashVal = 3	hashVal = 0	hashVal = 0

3. Disegno la tabella di hash risultante



Sono state inserite tutte le chiavi in quanto **non ci sono duplicati!** (le tabelle di hash non accettano chiavi duplicate)

> Le chiavi sono “0070”, “5019”, ... mentre i valori duplicati sono quelli risultanti dalla funzione di hash: h(0070) == h(5019), ma si risolve con le liste di collisione

4. Buone proprietà di una fuzione di hash

- deve essere calcolabile in tempo costante
- deve essere uniformemente distribuita quindi deve minimizzare le collisioni. (Ad esempio se hai 100 chiavi ed hai a disposizione 10 celle, una buona funzione di hash associa 10 chiavi ad ogni cella –> fattore di carico $\alpha = \frac{n}{m} = \frac{100}{10} = 10\%$.)

> NON DEVE ESSERE INIETTIVA A MENO CHE NON SI PARLI DI FUNZIONE PERFETTA

Nel nostro caso rispettiamo la prima proprietà (quindi “h()” è calcolabile in tempo costante) ma NON la seconda, infatti la nostra funzione non è uniformemente distribuita in quanto su 8 celle disponibili solamente 3 sono riempite e 2 di queste contengono più di un elemento.

5. Complessità operazione delete() caso peggiore

La complesità delle operazioni è dettata dal fattore di carico, ossia il grado di riempimento di una tabella (quindi se abbiamo una tabella di dimensione 10 e vogliamo inserire 10 elementi il fattore di carico sarà di 1, mentre se nella stessa tabella inseriamo 100 elementi, ogni cella conterrà una lista lunga circa 10) .

Infatti con una funzione di hashing ottima avremo che ogni cella dell’array contiene 1 solo elemento e quindi tutte le operazioni avranno complessità $O(1)$.

Invece se la funzione “h()” non è ottima (ma soddisfa comunque entrambe le “buone proprietà”), la complessità delle operazioni sarà dettata dal grado di riempimento di ogni cella –> $O(1+\alpha)$ dove $\alpha = \frac{numero\ elementi}{dimensione\ hashtable}$.

Liste di Collisione

Operazioni	*Caso Migliore*	*Caso Peggiore*
insert(elem e, chiave k)	*Θ(1)*	*Θ(1+n/m)*
delete(chiave k)	*Θ(1)*	*Θ(1+n/m)*
search(chiave k) → elem	*Θ(1)*	*Θ(1+n/m)*

Se avessimo usato un' implementazione con **indirizzamento aperto**, tutte le operazioni avrebbero avuto:

- complessità migliore: $O(1)$
- complessità peggiore: $O(n)$