Laboratorio di Algoritmi e Strutture Dati I

Esercitazione 10

martedì 19 maggio

Lezione 10: hash table

Una hash table è una struttura dati astratta in cui gli elementi memorizzati sono accessibili mediante l'applicazione di una funzione sulla chiave degli elementi stessi.

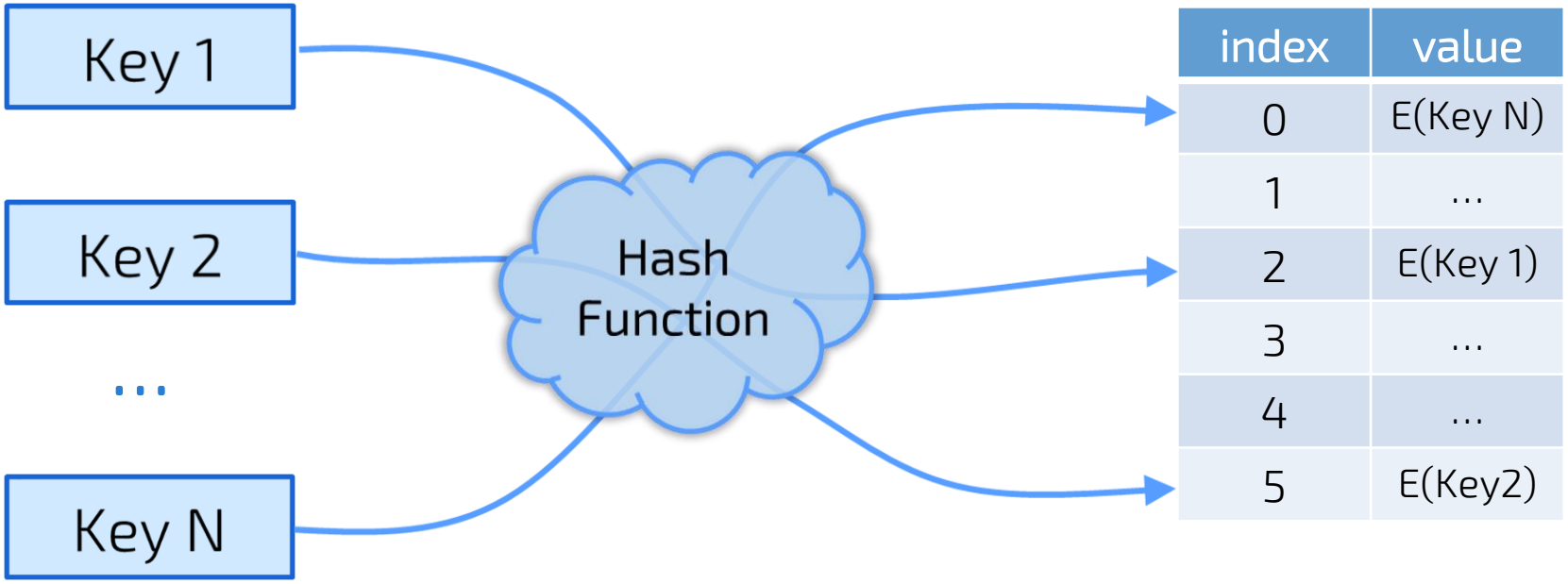
Definiamo:

- U = universo di tutte le possibili chiavi;
- T = tabella hash;
- m = dimensione della tabella T;
- n = numero di elementi effettivi da memorizzare in T;
- k = una generica chiave;
- h(k) = il risultato dell'applicazione di una certa funzione h ad una generica chiave k.

La regola generale delle hash table è : **T[h(k)] = elemento con chiave k .**

Lezione 10: hash table

T[h(k)] = elemento con chiave k



Notazione: E(k) indica l'elemento che ha k come chiave.

Lezione 10: perché le tavole hash

Le operazioni fondamentali (di base) che solitamente si effettuano su una struttura dati sono, tra le altre:

- Inserimento
- Ricerca
- Cancellazione

La complessità computazionale di tali operazioni cambia in base alla struttura dati utilizzata:

- Liste o Array: $O(n)$
- Alberi Binari di Ricerca non bilanciati: $O(n)$
- Alberi Binari di Ricerca bilanciati (AVL): $O(\log n)$
- Tavole Hash: $O(1)$ "teoricamente"
  - Perché?

Lezione 10: funzione hash

Una **funzione hash** (h) è una funzione che, data una certa chiave k appartenente all'universo U, restituisce la posizione h(k) della tabella in cui l'elemento con chiave k viene memorizzato

Idealmente, dati k_1 e k_2 (con $k_1 \neq k_2$), allora $h(k_1) \neq h(k_2)$.

Problema: la dimensione della tabella T (indicata con m) non potrà, per ovvi motivi, essere grande quanto l'intero universo U

Nella realtà, può capitare che, dati k_1 e k_2 (con $k_1 \neq k_2$), allora $h(k_1) = h(k_2)$.

Questo fenomeno viene chiamato **collisione**

Lezione 10: gestione delle collisioni

- Eliminare del tutto le collisioni scegliendo una funzione hash **perfetta** (spoiler: babbo natale non esiste)

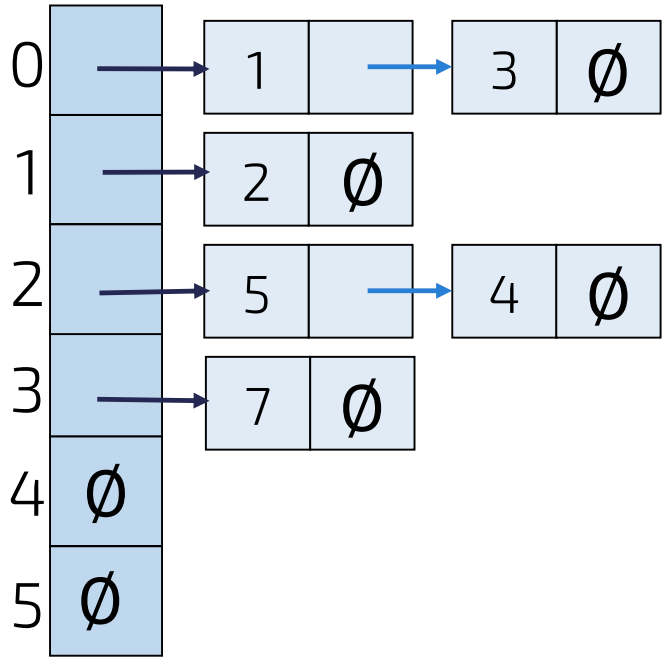
- Utilizzare una "**buona**" funzione hash, per minimizzare il numero delle collisioni, e gestirle se si presentano:

  - **Liste di collisione**: una generica locazione T[i] della tabella non contiene più un unico valore bensì il puntatore alla lista di elementi collidenti in i.
  - **Indirizzamento aperto:** se una cella è già occupata si cerca la prima cella libera. Tutti gli elementi sono memorizzati nella tabella.

Come influisce questo sulla complessità computazionale delle operazioni di base?

Lezione 10: lista delle collisioni

- Tutti gli elementi collidenti in una data posizione **i** vengono inseriti nella stessa posizione della tabella, in una lista concatenata.
- La locazione T[i] della tabella non contiene più un unico valore bensì il puntatore alla **lista** di elementi collidenti in i.



In questo esempio:

- h(1) = 0, h(3) = 0 (collisione)
- h(2) = 1 (perfetto)
- h(5) = 2, h(4) = 2 (collisione)
- h(7) = 3 (perfetto)

Esercitazione 10: parcheggio multipiano

Esercitazione 10: parcheggio multipiano

Vogliamo simulare la gestione di un parcheggio multipiano.

Abbiamo:

- Un parcheggio di NUM_PIANI piani
- Delle auto caratterizzate da delle targhe (salvate come stringa) e dall'orario di arrivo. NB: **la targa è la chiave**.
- Una funzione hash (che prende in input la targa) e ci restituisce il piano in cui l'auto deve essere parcheggiata.

```
typedef struct
{
   char targa[MAX_TARGA];
   int  ora;
   int  min;
}Auto;
```

```
typedef struct nodo
{
   Auto info;
   struct nodo* link;
}Nodo;

Nodo* parcheggio[NUM_PIANI]
```

Esercitazione 10: funzione hash

Abbiamo bisogno di implementare una funzione hash che prenda in input la targa dell'auto e restituisca in output il piano in cui parcheggiarla:

```
algoritmo hash_function(char targa[]) → int
```

- #Hint1: ricordatevi che un char può essere trattato come un intero.
- #Hint2: il numero restituito non può, ovviamente, superare il numero di piani presenti nel parcheggio.
- NB: supponiamo, per semplificare le cose, di non avere un limite massimo di auto per piano.

Esempio per un parcheggio di 7 piani:

- hash_code("PI555TA") = 6
- hash_code("CA220TO") = 2
- hash_code("EG000AL") = 5
- hash_code("GO000LE") = 5

Esercizio 10: progetto base

- Andate su eLearning e copiate il codice del progetto base per l'esercitazione odierna
- Create un nuovo progetto su Clion e incollateci il codice
- Troverete già definite:
  - Le strutture **Nodo** e **Auto**
  - La funzione *carica_auto_test(...)* che inserisce nel parcheggio 11 auto di test.
    - NB: funzionerà solo dopo aver definito una funzione hash e implementato la funzione di inserimento.
  - I prototipi delle funzioni necessarie.
    - Potete aggiungerne altre.

Esercitazione 10: funzione hash

Implementare:

- La *funzione hash* (da voi ideata e definita).
- Una funzione *acquisisci_auto(...)* che chiede in input (da terminale) le informazioni relative ad un'auto e le inserisce in una struttura di tipo Auto.
- Una funzione *inserisci_auto(...)* che permette di inserire una nuova auto, già acquisita, in base alla funzione hash applicata sulla targa. NB: le auto nelle liste di collisione NON sono memorizzate in maniera ordinata (quindi inserimento in … ?)
- Una funzione *stampa_piano(...)* che prende in input un numero di piano e stampa le informazioni di tutte le auto parcheggiate nel piano.
- Una funzione *stampa_parcheggio(...)* che stampa la disposizione delle auto nei vari piani (sfruttando la funzione precedente).
- Una funzione *ricerca_auto(...)* che permette di cercare un'auto nel parcheggio (prendendo in input il suo numero di targa) e restituisce il puntatore a nodo (Nodo*) che la rappresenta.
- Una funzione *elimina_auto(...)* che permette di eliminare un'automobile dal parcheggio.
- Un menu che consente all'utente di scegliere quale operazione effettuare.

Esercitazione 10: funzione hash

Esempio precedente (parcheggio di 7 piani):

stampa_piano(5)

```
Floor #5 : GO000LE  -  EG000AL
```

stampa_parcheggio()

```
Floor #0 :
Floor #1 :
Floor #2 : CA220TO
Floor #3 :
Floor #4 :
Floor #5 : GO000LE  -  EG000AL
Floor #6 : PI555TA
```

NB: questa disposizione varia a seconda della funzione hash implementata!

end().