Problema del dizionario

Vogliamo in questa sezione discutere le mappe o dizionari, ossia strutture che "generalizzano" gli array nel senso che le chiavi ammettono anche tipi diversi "da quello intero" (e "non vanno out of bound").

Supponiamo che U sia il nostro universo delle chiavi (per esempio, gli interi), e denotiamo con $S \subseteq U$ l'insieme delle chiavi memorizzate.

Vogliamo dotare la struttura dei seguenti metodi:

- `find(x)` — 1 se $x \in S$, 0 altrim. } STATICO
- `insert(x)` — $S \leftarrow S \cup \{x\}$
- `remove(x)` — $S \leftarrow S \setminus \{x\}$

} DINAMICO

Se U è anche ben ordinato allora possiamo implementare:

- pred (x) — $\max\{y \in S \mid y \leq x\}$
- succ (x) — $\min\{y \in S \mid y \geq x\}$
- intervallo (a, b) — $\{y \in S \mid a \leq y \leq b\}$

$\rightsquigarrow x \in S \Rightarrow \text{pred}(x) = \text{succ}(x) = x$.

- count (a, b) = # intervallo (a, b).

Struttura	find	insert	delete
Array	$O(n)$	$O(1)$	$O(1)$
Array ord.	$O(\log(n))$	$O(n)$	$O(n)$
Linked list	$O(n)$	$O(1)$	$O(1)$ (con riferim.) $O(n)$ (con valore)
" ord.	$O(n)$	$O(n)$	"
Heap	$O(n)$	$O(\log(n))$ sift-up	$O(\log(n))$ sift-down
BST alt. h	$O(h)$	$O(h)$	$O(h)$
Hash	$O(1)$ medio	$O(1)$ medio	$O(1)$ medio

Studiamo questi hash, che ci permettono per l'appunto di implem effic i dizionari

Hash

→ $[m] := \{0, \dots, m-1\}$

Una **FUNZIONE HASH** (o di hashing) è una funz $h : U \to [m]$ dove $|U| \gg m$.

→ l'idea è quella di studiare U su un insieme molto più piccolo — è vero che per Pigeonhole vi saranno collisioni (ovvero h non è iniettiva), ma cercheremo di rendere la scoperta di collisioni improb. nel caso medio.

Esempi classici MD5, SHA-1, SHA-256, o fondamenti $h(x) = x \% m$ per gli interi; un esempio più moderno è quello di prendere p primo in $[m+1, 2m]$ (teorema di Bertrand)

UNIVERSAL HASH è usare $h(x) = ((ax+b) \% p) \% m$ con $a \in \mathbb{Z}_p^*$, $b \in \mathbb{Z}_p$.

→ con quest'ultimo hash la prob. di collis. è 1/m (uniforme)

→ un hash si dice perfetto se non ha coll. su S.

Per chiavi lunghe si usa la tecnica del FOLDING, si divide l'elem. fattorizz. e si effettua lo XOR del risultato dell'hashing sui singoli fattori

Per le stringhe si divide in caratteri $a_0 \dots a_{n-1}$ e si usa $h(x) = \sum_{i=0}^{n-1} (a_i \sigma^{n-i-1}) \% m$ con $\sigma = |\Sigma|$ (e.g. $\sigma = 131$, per l'ASCII). per ottenere

Un altro hashing famoso è quello di Karp-Rabin, che ammortizza a $O(1)$ per "l'hash successivo".

Se $h : U \to [m]$ e $|S| = n$, si definisce:

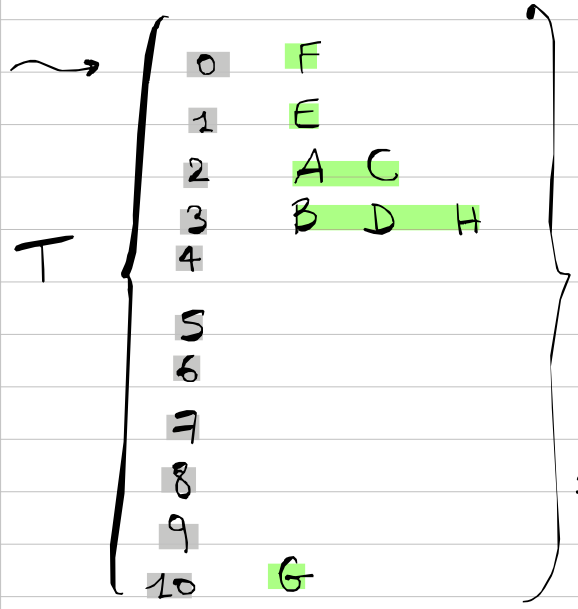
$$\alpha = \frac{n}{m}$$

su prob. unif.

come FATTORE DI CARICO, indica mediamente quanti elem. ci sono in una lista - elem. delle liste di trabocco.

Esempio

$x \in S$	B	A	D	C	F	G	E	H
$h(x)$	3	2	3	2	0	10	1	3



$m = 8$ $m = 11$

Liste contenute di trabocco

- find(x): scan T[h(x)]
- insert(x): T[h(x)].push(x)
- delete(x):
  - scan T[h(x)] until T[h(x)][i] = x
  - swap i ed end
  - pop

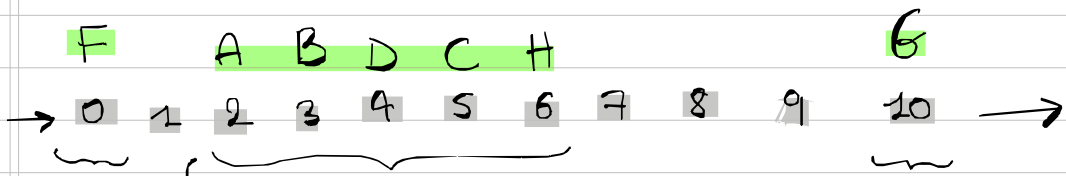
Il costo medio delle op. è $O(1+\alpha)$, mentre il costo pessimo è chiaram. $O(n)$ (se gli elem. non T) vengono spalmati uniform. su

Un'alternativa alle liste di trabocco è l'**indirizzamento aperto**.

Esempio

$x \in S$	B	A	D	C	F	G	H
$h(x)$	3	2	3	2	0	10	3

$m = 7$ $m = 11$



indirizzamento aperto

- **Inserimento**: parte da $T[h(x)]$ e si inserisce nel primo posto libero in modo circolare verso destra
- **Ricerca**: parte da $T[h(x)]$ e scorre fino a trovare l'elem o fino a quando trovi un posto libero (i.e. non c'è l'elem)
- **Cancell**: cancellazione logica (come negli AVL)

L'inserimento in indirizz. aperto è $O((1-\alpha)^{-1})$. Se $\tilde{T}(n,m)$ è il nr. medio di prob. eseguite per inserire una chiave, allora: ($n \geq m$)

$$\begin{cases} \tilde{T}(n,m) = \overbrace{\frac{n}{m}}^{p}\left(\underbrace{1+\tilde{T}(n-1,m-1)}_{\text{"resto il resto"}}\right) + \overbrace{\left(1-\frac{n}{m}\right)}^{1-p} \cdot 1 \\ \tilde{T}(0,m) = 1 \end{cases}$$

in part. $\widetilde{T}(n, m) = 1 + \frac{n}{m}\widetilde{T}(n-1, m-1)$, da cui:

$$\widetilde{T}(n, m) = 1 + \frac{n}{m}\left(1 + \frac{n-1}{m-1}\widetilde{T}(n-2, m-2)\right) =$$

$$= \dots =$$

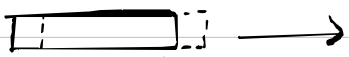
$$= 1 + \frac{n}{m}\left(1 + \frac{n-1}{m-1}\left(\dots\left(1 + \frac{1}{m-n+1}\cdot 1\right)\dots\right)\right) \leq$$

$$\leq \frac{m}{m-n} = (1-\alpha)^{-1} \rightsquigarrow O((1-\alpha)^{-1}).$$

Robin-Karp

→ tramite la tecnica del ROLLING HASH si possono comparare velocemente due stringhe per trovare una come sottostringa dell'altra.

stringa



sottostringa

si calcolano gli hash della sottostringa e della prima porzione della stringa. Se combaciano, si comparano le stringhe. Si passa poi alla successiva porzione, aggiornando l'hash. Se combacia, si comparano le stringhe, altrimenti si ripete.

Spesso come hash per $c_0 c_1 c_2 \dots c_{k-1}$ si usa: (RABIN-KARP FINGERPRINT)

$$c_0 \sigma^{k-1} + c_1 \sigma^{k-2} + \dots + c_{k-1} \pmod{p}$$

(p: primo)

dove $\sigma = |\Sigma|$ e Σ è l'alfabeto usato.

Per aggiornarlo è sufficiente computare:

$$H = \sigma(H - c_0 \sigma^{k-1}) + c_k \pmod{p}$$

calcolato con la post-exp.

Un'altra alternativa è il cuckoo hashing, che sfrutta due funzioni hash invece che una sola:

Cuckoo hashing

- $O(1)$
  - delete (caso pessimo)
  - find
- $O(1)$ ammortizzato
  - insert (medio)

- delete: elimina x in $T[h_2(x)]$ e $T[h_2(x)]$
- find: controlla $T[h_2(x)]$ e $T[h_2(x)]$

- **insert**: se libero, metto X in $T[h_1(X)]$; altrim. metto X in $T[h_2(X)]$ se libero. Se entrambi sono occupati inizio il seguente ciclo:
  - $i = h_1(X)$ [o $h_2(X)$]
  - while $T[i] \neq$ vuoto:
    - $tmp = T[i]$
    - $T[i] = X$
    - $X = tmp$
    - $i \leftarrow \{h_1(X), h_2(X)\} \setminus \{i\}$

Se il ciclo viene eseguito più di n volte, si effettua il REHASHING

REHASHING: si scelgono nuove h_1 e h_2, si riempie da capo T.

Analisi dell'inserimento

Si possono verificare tre situazioni:

(a.) la cella trovata è vuota ($O(1)$)

(b.) la cella non è vuota e il ciclo termina con conv. di lungh. l ($O(1+l)$)

(c.) il ciclo non termina $\rightsquigarrow$ REHASHING

Per (b.) è possibile stimare la lunghezza media $\tilde{l}$ di un cammino supponendo $m > 2nc$ con $c > 2$.

In tal caso — per induzione — $P(h_1(x) \overset{l}{\rightsquigarrow} J) \leq \frac{1}{c^l} \frac{1}{m}$

"si arrivi da $h_1(x)$ a J con cammino lungo l"

$\frac{1}{c^l}$: prob. di scegliere $h_1(x)$

$\left(\frac{n}{m}\right)$ è la prob. a ogni passo di trovare la cella occupata e si usa $m > 2nc$ per la stima...

Dunque:

$$\tilde{l} \leq \sum_{l \geq 1} l \cdot \frac{1}{c^l m} \leq \frac{1}{m} \frac{c}{(c-1)^2} = O(1)$$

è una serie derivata...

Quindi in media ci vuole $O(1 + \tilde{l}) = O(1)$.

Nel caso (c), possiamo stimare la probabilità di fare un rehashing:

$$P(\text{rehashing}) \leq P(\text{esista un ciclo}) \leq$$

$$\leq \sum_{i=1}^{m} P(i \to i) \leq$$

$$P(\bigcup_i X_i) \leq \sum_i P(X_i)$$

$$\leq \sum_{i=1}^{m} \sum_{l \geq 1} P(i \xrightarrow{l} j) \leq$$

$$\leq \sum_{i=1}^{m} \sum_{l \geq 1} \frac{1}{c^l} \frac{1}{m} \overset{(*)}{=}$$

$$= \frac{1}{m} \sum_{i=1}^{m} \frac{1}{c-1} = \frac{1}{c-1} = \rho$$

$\rightsquigarrow$ quindi $P(t \text{ rehashing}) \leq \rho^t$

$$\#\text{ medio di rehashing} = \sum_{t \geq 1} t \rho^t \overset{(**)}{=}$$

$$= \frac{\rho}{(1-\rho)^2} = O(1)$$

$$(*) \quad \sum_{l \geq 1} \frac{1}{c^l} = \frac{c}{c-1} - 1 = \frac{1}{c-1}.$$

$$(**) \quad \sum_{t \geq 1} t \rho^t = \rho \sum_{t \geq 1} t \rho^{t-1} = \rho \left(\frac{1}{1-\rho} \right)' =$$

$$= \frac{\rho}{(1-\rho)^2} \quad (\text{dacché } c > 2, \ \rho < 1 \ldots)$$