Esercitazione 5
Algorithmi e Strutture Dati (Informatica)
A.A 2015/2016

Tong Liu

April 7, 2016

Liste trabocco (Separate Chaining)

Esercizio 1

[Libro 7.5] Un dizionario è realizzato con liste di trabocco. Alcune chiavi sono ricercate molto più frequentemente delle altre ma non si sa a priori quali siano queste chiavi. Quali accorgimenti si possono adottare per sperare di abassare il tempo di accesso a queste chiavi ?

Svolgimento

Soluzione del libro: Quando si accede ad una chiave, si può scambiarla di posto con quella che la precede nella lista di trabocco. In questo modo, quelle chiavi più frequntemente ricercate tenderanno di spostarsi verso l'inizio della lista e quindi richiederà un minore tempo di accesso.
Correzione: in lezione abbiamo discusso che questo approccio avrebbe un bug, immaginiamo che elementi 1 e 2 stianno in fondo della lista, per un stream di dati composto da {2,1,2,1,2,1,2 ...}, elementi 1 e 2 staranno sempre in fondo e l'efficienza non aumenta.
Un vosto collega ha proposto una modifica: usare una variabile associata ad ogni elemento per contare la lettura, e mantenere la lista ordinata con questa variabile. Costo di aggiornamento ad ogni lettura $O(n)$
Un altro collega ha proposto un miglioramento: per ogni elemento appena letto, lo sposta nella prima posizione della lista. Costo di aggiornamento ad ogni lettura $O(1)$.

Esercizio 2

[Libro 7.10] Supponiamo di mantenere ordinate le liste di trabocco con le loro chiavi. Quali sono i vantaggi e/o gli svantaggi di tale scelta in termini di efficienza ?

Svolgimento

Supponianmo che la lista di trabocco abbia n elementi.
Pro: Quando si cerca un elemento, è possibile fermarsi prima quando si è accorto l'elemento recuperato comincia di essere maggiore o uguale dell'elemento cercato, e quindi risparimia di fare scansione su intera lista in caso di assenza. Inoltre, se la ricerca è implementata con la tecnica divide et impera, il costo diventa $O(\log n)$ al posto di $O(n)$ nell'approccio classico.
Con: L'operazione di inserimento avrebbe un costo maggiore (caso pessimo $O(n)$) perché si deve scansionare la lista per confrontare il valore dell'elemento e trovare la posizione opportuna per l'inserimento.

Indirizzamento aperto (Open Addressing)

Esercizio 3

[Libro 7.2] Si consideri una tabella hash di dimensione $m = 11$ inizialmente vuota. Si mostri il contenuto della tabella dopo aver inserito nell'ordine, i valori 33, 10, 24, 14, 16, 13, 23, 31, 18, 11, 7. Si assuma che le collisioni siano gestite mediante indirizzamento aperto utilizzando come funzione di hash $h(k)$, definita nel modo seguente:

$$h(k) = (h'(k) + 3i + i^2) \mod m \tag{0.1}$$

$$h'(k) = k \mod m \tag{0.2}$$

Mostrare il contenuto della tabella al terminedegli inserimenti e calcolare il numero medio di accessi alla tabella per la ricerca di una chiave presente nella tabella.

Svolgimento

La tabella hash è composta nel modo seguente:

0	1	2	3	4	5	6	7	8	9	10
33	23	24	14	11	16	13	18		31	10

Il valore 7 non può essere inserito, perché la funzione di scan scorre 11 posizioni ($\geq m$) e non trova l'unica cella libera rimasta (la 8). Il numero di accessi medi è 1.1 per i valori che hanno trovato collocazione.

Esercizi Esami

Esercizio 4

[19/06/2015] Data una lista L di interi, si vuole togliere da L tutti gli elementi ripetuti e inserirli in una nuova lista M, matenendo in entrambe le liste l'ordine originario degli elementi

(p.e. se L = 2,1,5,4,5,2,7,4,5, il risultato è L = 1,7 e M =2,5,4,5,2,4,5). Si scriva in pesudocodice un algoritmo efficiente utilizzando gli operatori delle liste visti a lezione.

Svolgimento

Per svolgere questo problema in $O(n)$, si utilizza una tabella hash H grande sufficiente, poi l'algoritmo scorre ogni elemento L_i di L per $i = 1,\ldots,n$, memorizzando in H coppie chiave-valore (L_i, C_i) dove C_i è il numero di occorenze di L_i in L. Quindi, si scorre L una seconda volta e per ogni L_i si controlla il corrispondente valore C_i in H: se $L_i > 1$, li viene rimosso da L ed inserito in M; altrimenti proseguo la scansione. Se H è sufficientemente grande(e.g. $2n$) allora il costo di ricerca è mediamente $O(1)$, quindi il costo complessivo dell'algoritmo è $O(n)$ nel caso medio. Per dettagli si vede lo Pseudocodice 1.
Un vostro collega ha proposto di unire passo 2 e 3 dell'Algoritmo 1, cioè quando legge per la seconda volta lo stesso valore, si cancella già dalla lista L e lo inserisce nella lista M. Questo avrebbe un difetto perché i numeri ripetuti non veranno completamente eliminati dalla lista L quindi il terzo passo è obbligatoriamente eseguire un'altra volta.

Esercizio 5

[15/07/2014] Si descriva la struttura di dati dizionario, con le operazione tipicamente utilizzate, e la sua realizzazione efficiente tramite tabella hash. In quali casi la realizzazione con tabella hash è da preferirsi rispetto ad altre realizzazioni?

Svolgimento

Il termine dizionario rappresenta il concetto matematico di relazione univoca $R : D \to C$ fra gli elementi di un insieme D (il dominio) e gli elementi di un insieme C (il codominio). Gli elementi del dominio prendono spesso il nome di chiavi e gli elementi del codominio valori, si parla così di associazione chiave-valore. Le operazioni ammesse sono(vedi in Pseudocodice 2):
La sua realizzazione può essere realizzata con le liste di trabocco.
Assumiamo α rappresenta il fattore di carico, che è la lunghezza media di ciascuna lista di trabocco. In caso di successo, mediamente viene scandita metà lista, mentre in caso di insuccesso viene scandita un'intera lista. In entrambi i casi, è necessario un accesso al vettore A che contiene il puntatore alla lista. $I(\alpha) = 1 + \alpha$ e $S(\alpha) = 1 + \frac{\alpha}{2}$.

Richiamo i costi di operazioni di tabella hash, lookup(), insert(), remove(), tutti $O(1)$ min e scansione, $O(m+n)$ in caso di liste trabocco. $O(\log n)$ per alberi, e spesso $O(n)$ per liste.

Quindi Qualora l'obiettivo sia ottenere realizzazioni efficienti delle operazioni di base, e non sia necessario implementare le operazioni min, scansione e ordinamento.

Ref: P 61, P 142, P 149 del libro testo.

Algorithm 1 Procedura elimina duplicati

```
1:  procedure REMOVE-DUPLICATES(List L, List M)
2:      % Passo 1, calcolare la lunghezza di L che servirà per dichiarare la tabella hash
3:      Pos p ← L.head()
4:      Integer n ← 0
5:      while not L.finished(p) do
6:          n ← n + 1
7:          p ← L.next(p)
8:      end while
9:      Hash H ← Hash(2n)
10:
11:     % Passo 2, caricare la tabella hash
12:     p ← L.head()
13:     while not L.finished(p) do
14:         Integer l ← L.read(p)
15:         Integer c ← H.lookup(l)
16:         if c = nil then
17:             H.insert(l, 1)
18:         else
19:             H.insert(l, c + 1)
20:         end if
21:         p ← L.next(p)
22:     end while
23:
24:     % Passo 3, filtrare i numeri
25:     p ← L.head()
26:     while not L.finished(p) do
27:         Integer l ← L.read(p)
28:         Interger c ← H.lookup(l)
29:         if l ≠ 1 then
30:             M.insert(M.tail(), l)
31:             L.remove(p)
32:         else
33:             p ← L.next(p)
34:         end if
35:     end while
36: end procedure
```

Algorithm 2 Operazioni dizionari

```
1: procedure OPERAZIONI DIZIONARI
2:     % Restituisce il valore associatoalla chiave k se presente, nil altrimenti
3:     ITEM lookup(ITEM k)
4:
5:     % Associa il valore v alla chiave k
6:     insert(ITEM k, ITEM v)
7:
8:     % Rimuove l'associazione della chiave k
9:     remove(ITEM k)
10: end procedure
```