Tabelle dinamiche

Spesso non si conosce a priori quanta memoria serve per memorizzare una struttura dati (tabella di dati ~ array, tabella hash, heap, stack, ecc.).

Può capitare quindi di allocare una certa quantità di memoria e poi accorgersi, durante l'esecuzione del programma, che tale memoria non è sufficiente.

In tal caso bisogna allocare una memoria maggiore, ricopiare il contenuto della vecchia memoria nella nuova e rilasciare la vecchia memoria.

1

E' anche opportuno, dopo aver rimosso molti elementi, ridurre la memoria allocando una memoria più piccola in cui memorizzare gli elementi rimasti.

Vedremo come sia possibile aggiungere e togliere un elemento dalla tabella in tempo ammortizzato costante ***O*(1)** benchè tali operazioni abbiano costo maggiore quando comportano espansione o riduzione della memoria.

Vedremo anche che questo si può fare garantendo che la memoria inutilizzata sia sempre inferiore ad una frazione costante della memoria allocata.

2

Supporremo che una tabella ***T*** abbia i seguenti attributi:

a) un puntatore ***pt*[*T*]** alla memoria riservata per memorizzare gli elementi.

b) due campi interi ***num*[*T*]** e ***size*[*T*]** che contengono rispettivamente il numero di elementi presenti nella tabella e la dimensione della tabella.

3

Le operazioni che vogliamo realizzare sono

1. ***Table*(*T*)**: costruisce una tabella vuota ***T***.
2. ***Insert*(*T, x*)**: inserisce l'oggetto ***x*** nella tabella ***T***.
3. ***Delete*(*T, x*)**: rimuove l'oggetto ***x*** dalla tabella ***T***.

4

Non ci cureremo di come gli elementi siano organizzati nella memoria allocata né di altri dettagli implementativi ma ci limiteremo ad assumere che siano definite le tre operazioni:

***Push*(*pt, x*)**	memorizza l'elemento ***x*** nella memoria puntata da ***pt***
***Pop*(*pt, x*)**	rimuove l'elemento ***x*** dalla memoria puntata da ***pt***
***Copy*(*pt1, pt, n*)**	ricopia nella memoria puntata da ***pt1*** ***n*** elementi memorizzati nella memoria puntata da ***pt***

5

Per allocare e rilasciare memoria utilizzeremo le due operazioni:

Allocate*(*n*)** che riserva memoria per ***n elementi e restituisce un puntatore a tale memoria.

Free*(*pt, n*)** che rilascia la memoria per ***n elementi puntata da ***pt***.

6

Espansione

Consideriamo dapprima il caso in cui vengono eseguite soltanto inserzioni di nuovi elementi e nessuna rimozione.

Definiamo come *fattore di carico* della tabella il rapporto α = ***num*** / ***size***.

Quando α = **1** la tabella è piena ed occorre *espanderla* allocando nuova memoria. Una euristica comune è raddoppiare la memoria, il che garantisce un fattore di carico $\alpha > 1/2$.

7

La definizione della funzione ***Table*(*T*)** che inizializza una tabella vuota è:

```
Table(T)
  size[T] ← 0
  num[T] ← 0
  pt[T] ← nil
```

Essa richiede un tempo costante.

8

La definizione della funzione ***Insert*(*x*)** è:

```
Insert(T, x)
  if size[T] = 0  then
    pt[T] ← Allocate(1)
    size[T] ← 1
  if num[T] = size[T] then
    pt1 ← Allocate(2 * size[T])
    Copy(pt1, pt[T], num[T])
    Free(pt[T], size[T])
    pt[T] ← pt1
    size[T] ← 2 * size[T]
  Push(pt[T], x)
  num[T] ← num[T] + 1
```

9

Per l'analisi della complessità di ***Insert*** possiamo assumere che le operazioni ***Allocate***, ***Free*** e ***Push*** richiedano tempo costante e che ***Copy*** richieda tempo proporzionale al numero di elementi copiati.

Consideriamo il costo di una sequenza di ***n*** operazioni ***Insert*** eseguite a partire da una tabella vuota.

Il costo della ***i***-esima ***Insert*** è **1** se non vi è espansione ed è ***i*** se vi è espansione (***i*** **- 1** per la ***Copy*** degli ***i*** **- 1** elementi precedenti più **1** per le altre operazioni).

10

Siccome ***size*[*T*]** è sempre una potenza di **2** e l'espansione si ha quando ***num*[*T*]** = ***i*** **- 1** è uguale a ***size*[*T*]** il costo della ***i***-esima ***Insert*** è:

$$c_i = \begin{cases} i & \text{se } i-1 \text{ è una potenza di 2} \\ 1 & \text{altrimenti} \end{cases}$$

e il costo totale della sequenza di ***n Insert*** è:

$$C(n) = \sum_{i=1}^{n} c_i = n + \sum_{j=0}^{\lfloor \log_2 (n-1) \rfloor} 2^j = n + 2^{\lfloor \log_2 (n-1) \rfloor + 1} - 1 < 3n$$

Insert ha quindi costo ammortizzato ***O*(3*n*)/*n* = *O*(1).**

11

Il metodo degli accantonamenti mostra perchè il costo ammortizzato debba essere **3**.

1. una unità di costo viene spesa subito per l'inserimento dell'elemento stesso,
2. una viene attribuita come credito all'elemento stesso per pagare un suo successivo ricopiamento e
3. la terza viene attribuita come credito ad un (eventuale) altro elemento ricopiato prima e rimasto privo di credito.

Al momento dell'espansione ogni elemento ha una unità di costo per pagarsi il ricopiamento.

12

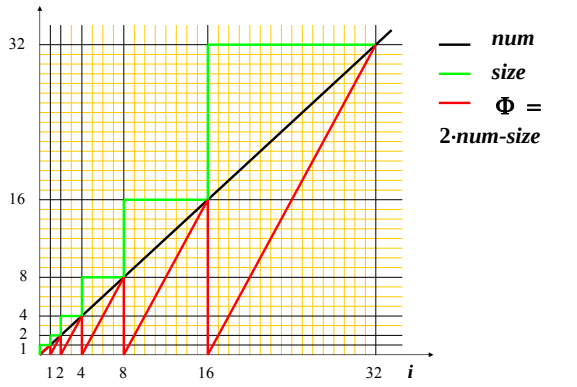
Possiamo usare il metodo del potenziale scegliendo una funzione potenziale **Φ** che vale **0** (all'inizio e) subito dopo un ricopiamento e che cresce tra una espansione e la successiva, raggiungendo la dimensione della tabella quando questa è piena.

Una tale funzione è: $\Phi = 2 \cdot num - size$

Subito dopo il ricopiamento: $\Phi = 2 \cdot \frac{1}{2} size - size = 0$

Quando la tabella è piena: $\Phi = 2 \cdot size - size = size$

13



14

Il costo ammortizzato di un inserimento senza espansione è:

$$\begin{aligned}\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= 1 + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\ &= 1 + 2 \cdot (num_{i-1} + 1) - size_{i-1} - 2 \cdot num_{i-1} + size_{i-1} \\ &= 3\end{aligned}$$

con espansione $\hat{c}_1 = 2$ (tabella vuota) e per ***i* > 1**:

$$\begin{aligned}\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= 1 + num_{i-1} + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\ &= 1 + num_{i-1} + 2 \cdot (num_{i-1} + 1) - 2 \cdot num_{i-1} - 2 \cdot num_{i-1} + num_{i-1} \\ &= 3\end{aligned}$$

15

Espansione e contrazione

Delete*(*x*)** si realizza con una ***Pop*(*pt, x*)** che rimuove l'elemento ***x dalla tabella.

Ad evitare uno spreco eccessivo di memoria è opportuno contrarre la tabella quando il fattore di carico **α = *num* / *size*** diventa troppo piccolo.

Questo si fa allocando una memoria più piccola, ricopiando i ***num*** elementi presenti nella tabella, e rilasciando la vecchia memoria.

16

La strategia ovvia è dimezzare la memoria quando il fattore di carico **α** diventa **1/2**.

Questo ci assicura un fattore di carico **α > 1/2** anche in presenza di operazioni ***Delete***.

Purtroppo, con questa strategia il costo ammortizzato delle operazioni non è più costante.

17

Consideriamo una successione di $n = 2^k$ **(*k*>3)** operazioni delle quali le prime 2^{k-1} sono ***Insert*** mentre le altre 2^{k-1} sono una ripetizione di 2^{k-3} gruppi di quattro operazioni: una ***Insert*** seguita da due ***Delete*** seguite da una ***Insert***.

L'esecuzione di ogni gruppo comporta una espansione da ***size*** = 2^{k-1} a 2^k di costo 2^{k-1} ed una contrazione da ***size*** = 2^k a 2^{k-1} anch'essa con costo 2^{k-1}.

18

Il costo della sequenza di 2^{k-3} gruppi è quindi pari a $2^{k-3}(4+2^{k-1}+2^{k-1}) = O(2^{2k}) = O(n^2)$.

Quindi il costo ammortizzato di una operazione è $O(n^2) / n = O(n)$.

Il problema è che dopo l'espansione (che porta α ad **1/2** e consuma tutti i crediti) non si eseguono rimozioni sufficienti ad accumulare crediti per la successiva contrazione.

Occorre quindi aspettare che sia stata rimossa almeno una metà degli elementi, ossia che α diventi **1/4**.

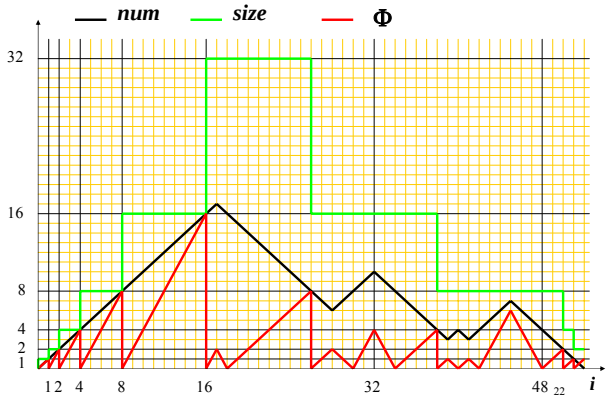
La definizione della funzione ***Delete*(*x*)** è:

```
Delete(T, x)
  if num[T] ≤ size[T] / 4 then
    pt1 ← Allocate(size[T] / 2)
    Copy(pt1, pt[T], num[T] )
    Free(pt[T], size[T] )
    pt[T] ← pt1
    size[T] ← size[T] / 2
  Pop(pt[T], x)
  num[T] ← num[T] - 1
```


Usiamo il metodo del potenziale scegliendo una funzione potenziale Φ che vale **0** sia all'inizio che subito dopo una espansione o contrazione e che cresce fino a raggiungere il numero di elementi presenti nella tabella quando il fattore di carico α aumenta fino ad **1** o diminuisce fino ad **1/4**.

Una funzione di questo tipo è

$$\Phi = \begin{cases} 2 \cdot num - size & \text{se } \alpha \geq 1/2 \\ size/2 - num & \text{se } \alpha \leq 1/2 \end{cases}$$



Se $\alpha \geq 1/2$ il costo ammortizzato di un inserimento senza espansione è:

$$\begin{aligned} \hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= 1 + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\ &= 1 + 2 \cdot (num_{i-1} + 1) - size_{i-1} - 2 \cdot num_{i-1} + size_{i-1} \\ &= 3 \end{aligned}$$

con espansione $\hat{c}_1 = 2$ (tabella vuota) e per $i > 1$:

$$\begin{aligned} \hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= 1 + num_{i-1} + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\ &= 1 + num_{i-1} + 2 \cdot (num_{i-1} + 1) - 2 \cdot num_{i-1} - 2 \cdot num_{i-1} + num_{i-1} \\ &= 3 \end{aligned}$$

Se $\alpha < 1/2$ non vi è sicuramente espansione e il costo ammortizzato di un inserimento è:

$$\begin{aligned} \hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= 1 + (size_i / 2 - num_i) - (size_{i-1} / 2 - num_{i-1}) \\ &= 1 + size_{i-1} / 2 - (num_{i-1} + 1) - size_{i-1} / 2 + num_{i-1} \\ &= 0 \end{aligned}$$

Se $\alpha \leq 1/2$ il costo ammortizzato di una rimozione senza contrazione è:

$$\begin{aligned}\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= 1 + (size_i/2 - num_i) - (size_{i-1}/2 - num_{i-1}) \\ &= 1 + size_{i-1}/2 - (num_{i-1} - 1) - size_{i-1}/2 + num_{i-1} \\ &= 2\end{aligned}$$

mentre con contrazione è:

$$\begin{aligned}\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= 1 + num_{i-1} + (size_i/2 - num_i) - (size_{i-1}/2 - num_{i-1}) \\ &= 1 + num_{i-1} + num_{i-1} - (num_{i-1} - 1) - 2 \cdot num_{i-1} + num_{i-1} \\ &= 2\end{aligned}$$

25

Se $\alpha > 1/2$ non vi è sicuramente contrazione e il costo ammortizzato di una rimozione è:

$$\begin{aligned}\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= 1 + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\ &= 1 + 2 \cdot (num_{i-1} - 1) - size_{i-1} - 2 \cdot num_{i-1} + size_{i-1} \\ &= -1\end{aligned}$$

26

Esercizio 16.

Assumere che la contrazione della tabella dinamica venga effettuata quando $\alpha = 1/3$ invece che quando $\alpha = 1/4$ e che invece di ridurre la sua dimensione ad **1/2 *size*** essa venga ridotta a **2/3 *size***.

Calcolare il costo ammortizzato delle operazioni usando la funzione potenziale:

$$\Phi = |2\,num - size|$$

27