Analisi ammortizzata

Si considera il tempo richiesto per eseguire, nel caso pessimo, una intera sequenza di operazioni.

Se le operazioni costose sono relativamente meno frequenti allora il costo richiesto per eseguirle può essere ***ammortizzato*** con l'esecuzione delle operazioni meno costose.

Metodo dell'aggregazione

Si calcola la complessità $O(f(n))$ dell'esecuzione di una sequenza di n operazioni nel caso pessimo.

Il costo ammortizzato della singola operazione si ottiene dividendo per n tale complessità ottenendo $O(f(n)/n)$

In questo modo viene attribuito lo stesso costo ammortizzato a tutte le operazioni.

Illustriamo il metodo con due esempi:

operazioni su di una pila

Sia ***P*** una pila con operazioni:

Push*(*P*,*x*)** // aggiunge ***x alla pila ***P***
***Pop*(*P*)** // toglie il primo elemento dalla pila
Top*(*P*)** // restituisce il primo elemento di ***P
// (senza toglierlo)
Empty*(*P*)** // ritorna ***true se la pila è vuota

a cui aggiungiamo:

```
MultiPop(P, k)
  while not Empty(P) and k > 0
    Pop(P), k = k-1
```

che toglie i primi ***k*** elementi, oppure vuota la pila se ci sono meno di ***k*** elementi

Se la pila contiene m elementi il ciclo while viene iterato $\min(m,k)$ volte e quindi ***MultiPop*** ha complessità $O(\min(m,k))$.

Consideriamo una sequenza di n operazioni eseguite a partire dalla pila vuota.

L'operazione più costosa ***MultiPop*** richiede tempo $O(n)$ nel caso pessimo.

Moltiplicando per n otteniamo il limite superiore $O(n^2)$ per il costo della sequenza di n operazioni.

Il metodo dell'aggregazione fornisce un limite più stretto.

Un elemento può essere tolto dalla pila soltanto dopo che è stato inserito!

Quindi il numero di operazioni ***Pop***, comprese quelle interne alle ***MultiPop***, non può superare il numero totale di operazioni ***Push*** ed è quindi minore di ***n.***

Se dal tempo richiesto per eseguire ***MultiPop*** togliamo il tempo per le iterazioni del ciclo **while** rimane un tempo costante.

Quindi il tempo richiesto per eseguire l'intera sequenza di n operazioni è $O(n)$ più il tempo richiesto per eseguire tutte le iterazioni dei cicli **while** interni alle operazioni ***MultiPop*** presenti nella sequenza.

Siccome una iterazione richiede tempo costante e il numero totale di iterazioni è minore di n, l'esecuzione di tutte le iterazioni del ciclo **while** richiede tempo totale $O(n)$.

Il costo dell'intera sequenza di operazioni è quindi $O(n)$ e pertanto il costo ammortizzato di ciascuna operazione è $O(n)/n = O(1)$.

Incremento di un contatore binario

Implementiamo un contatore binario di ***k*** bit con un array di bit

$$\boxed{A[0..k\text{-}1]}$$

Un numero binario ***x*** registrato in ***A*** ha il bit meno significativo in ***A*[*0*]** e il più significativo in ***A*[*k*-1]** per cui

$$x = \sum_{i=0}^{k-1} A[i]2^i$$

Supponiamo che A venga usato per contare a partire da $x = 0$ usando l'operazione di incremento:

```
Increment(A)
  i = 0
  while i < k and A[i] == 1
    A[i] = 0
    i = i +1
  if i < k
    A[i] = 1
```

Una singola operazione di incremento richiede tempo $\boldsymbol{O(k)}$ nel caso pessimo il che fornisce un limite superiore $\boldsymbol{O(nk)}$ per una sequenza di $\boldsymbol{n}$ incrementi.

Possiamo però osservare che il tempo necessario ad eseguire l'intera sequenza è proporzionale al numero di bit che vengono modificati.

Quanti bit vengono modificati in totale?

Vediamo cosa succede con un contatore di $\boldsymbol{k = 8}$ bit.

x	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	costo
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23
14	0	0	0	0	1	1	1	0	25
15	0	0	0	0	1	1	1	1	26
16	0	0	0	1	0	0	0	0	31

Si vede che:

$A[\mathbf{0}]$ viene modificato ogni volta

$A[\mathbf{1}]$ viene modificato ogni **2** volte

$A[\mathbf{2}]$ viene modificato ogni **4** volte

ed in generale:

$A[\boldsymbol{i}]$ viene modificato ogni $\mathbf{2}^{\boldsymbol{i}}$ volte

Il numero totale di bit modificati è

$$\sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$

La complessità di n operazioni di incremento a partire da $x = 0$ è $O(n)$.

Quindi la complessità ammortizzata di una operazione di incremento è $O(n)/n = O(1)$.

***Esercizio* 11**

Mostrare che se al contatore binario di k bit aggiungiamo anche una operazione ***Decrement*** che decrementa di una unità il valore del contatore allora una sequenza di n operazioni può costare $\Theta(nk)$

***Esercizio* 12**

Su di una struttura dati si esegue una sequenza di n operazioni.

L'operazione k-esima costa k quando k è una potenza di **2** e costa **1** negli altri casi.

Mostrare che le n operazioni hanno costo ammortizzato costante.

Metodo degli accantonamenti

Si caricano le operazioni meno costose di un costo aggiuntivo.

Il costo aggiuntivo viene assegnato come ***credito prepagato*** a certi oggetti nella struttura dati.

I crediti accumulati saranno usati per pagare le operazioni più costose su tali oggetti.

Il costo ammortizzato delle operazioni meno costose è il costo effettivo aumentato del costo aggiuntivo.

Il costo ammortizzato delle operazioni più costose è il costo effettivo diminuito del credito prepagato utilizzato.

Illustriamo questo metodo con i soliti due esempi.

operazioni su di una pila

Ricordiamo che i *costi effettivi* delle operazioni sulla pila sono:

Push	**1**
Pop	**1**
Top	**1**
Empty	**1**
MultiPop	**1+min(k,m)**

Push	**2**
Pop	**0**
Top	**1**
Empty	**1**
MultiPop	**1**

A tali operazioni attribuiamo dei *costi ammortizzati*

Quando effettuiamo una ***Push*** usiamo una unità di costo per pagare il costo effettivo dell'operazione mentre l'altra unità di costo la attribuiamo come credito prepagato all'oggetto inserito nella pila.

Quando eseguiamo una ***Pop*** paghiamo il costo dell'operazione utilizzando il credito attribuito all'oggetto che viene tolto dalla pila.

Quando eseguiamo una ***Multi-Pop*** l'unità di costo attribuita a tale operazione si usa per pagare il costo dell'operazione stessa escluse le iterazioni del ciclo **while**.

Le **min(*k*,*m*)** iterazioni del ciclo **while** vengono invece pagate utilizzando i **min(*k*,*m*)** crediti prepagati attribuiti agli oggetti che vengono tolti dalla pila.

incremento di un contatore binario

```
Increment(A)
  i = 0
  while i < k and A[i] == 1
    A[i] = 0
    i = i +1
  if i < k
    A[i] = 1
```

Il costo effettivo di una operazione ***Increment*** è pari al numero di bit modificati.

Tra questi vi è un certo numero $t \geq 0$ di bit **1** trasformati in **0** e al più un solo bit **0** trasformato in **1.**

Attribuiamo costo ammortizzato **2** ad ogni operazione ***Increment***.

Per eseguire ***Increment*** usiamo una delle due unità di costo per pagare l'eventuale bit **0** trasformato in **1** e l'altra unità di costo la attribuiamo come credito prepagato a tale bit.

In questo modo ogni bit **1** ha sempre un credito prepagato che possiamo usare per pagare la trasformazione dei ***t*** bit **1** trasformati in **0** dall'operazione ***Increment***.

***Esercizio* 13**

Realizzare un contatore binario che prevede, oltre all'operazione ***Increment***, anche una operazione ***Reset*** che azzera il contatore.

La complessità ammortizzata delle operazioni deve risultare costante.

Suggerimento: memorizzare la posizione del bit **1** più significativo.

***Esercizio* 14**

Realizzare una pila ***P*** con operazioni di costo ammortizzato costante avendo a disposizione memoria per al più ***m*** elementi.

Se la memoria è piena prima di eseguire una ***Push***, si scarica su disco una parte degli ***m*** elementi.

Se dopo una operazione ***Pop*** la memoria resta vuota ma ci sono altri elementi su disco, se ne ricarica una parte in memoria.

Metodo del potenziale

Si associa alla struttura dati ***D*** un potenziale **Φ(*D*)** tale che la modifica della struttura dati dovuta alle operazioni meno costose comporti un aumento del potenziale mentre le operazioni più costose lo facciano diminuire.

Il costo ammortizzato è quindi la somma algebrica del costo effettivo e della variazione di potenziale.

Se $\boldsymbol{D_i}$ è la struttura dati dopo l'esecuzione della $\boldsymbol{i}$-esima operazione e $\boldsymbol{c_i}$ è il costo effettivo dell'operazione allora il costo ammortizzato è:

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

e il costo ammortizzato di una sequenza di $\boldsymbol{n}$ operazioni è:

$$\hat{C} = \sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} [c_i + \Phi(D_i) - \Phi(D_{i-1})]$$

$$= C + \Phi(D_n) - \Phi(D_0)$$

Se la variazione $\boldsymbol{\Phi(D_n)-\Phi(D_0)}$ del potenziale relativa all'esecuzione di tutta la sequenza non è negativa allora il costo ammortizzato $\boldsymbol{\hat{C}}$ è una maggiorazione del costo reale $\boldsymbol{C}$.

Altrimenti un valore $\boldsymbol{\Phi(D_n)-\Phi(D_0)}$ negativo deve essere compensato con un aumento adeguato del costo ammortizzato delle operazioni.

Illustriamo anche questo metodo con i soliti due esempi.

operazioni su di una pila

Come funzione potenziale $\Phi(P)$ prendiamo il numero m di elementi contenuti nella pila P per cui:

Operazione	costo effettivo	differenza di potenziale	costo ammortizzato
Push	1	1	2
Pop	1	-1	0
Top	1	0	1
Empty	1	0	1
MultiPop	1+min(k,m)	-min(k,m)	1

Osserviamo che all'inizio la pila è vuota e quindi $\mathbf{\Phi(\boldsymbol{P}_0) = 0}$ mentre alla fine $\mathbf{\Phi(\boldsymbol{P}_n) \geq 0}$.

Quindi la differenza di potenziale relativa a tutta la sequenza di operazioni non è mai negativa.

incremento di un contatore binario

Scegliamo come funzione potenziale $\Phi(A)$ il numero di bit **1** presenti nel contatore

Increment cambia $t \geq 0$ bit **1** in **0** e al più uno **0** in **1**

Quindi:

Operazione	costo effettivo	differenza di potenziale	costo ammortizzato
Increment	1+t	-t+1	2

Una sequenza di operazioni comporta una differenza di potenziale non negativa.

Infatti all'inizio il contatore vale **0** e tutti i bit sono **0** e quindi $\mathbf{\Phi}(A_0) = \mathbf{0}$ mentre alla fine $\mathbf{\Phi}(A_n) \geq \mathbf{0}$.

Con il metodo del potenziale si può calcolare il costo ammortizzato anche quando il contatore non parte da **0**.

In questo caso $\Phi(D_n) - \Phi(D_0)$ può essere negativa ma pur sempre in modulo minore o uguale di k.

Possiamo quindi compensarla aggiungendo la quantità k/n al costo ammortizzato di ***Increment***.

Il costo ammortizzato di ***Increment*** è quindi $O(1+k/n)$ che, se $k = O(n)$, si riduce ad $O(1)$.

***Esercizio* 15**

Realizzare una coda ***Q*** di tipo **FIFO** utilizzando due normali pile $\boldsymbol{P_1}$ e $\boldsymbol{P_2}$ e le relative operazioni ***Push*** e ***Pop***.

Le operazioni ***PushQ*** e ***PopQ*** di inserimento ed estrazione dalla coda devono richiedere tempo ammortizzato costante.