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.

1

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 quindi 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.

2

operazioni su di una pila

Sia ***P*** una pila di interi con le solite 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

ed una ulteriore operazione:

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

che toglie dalla pila i primi ***k*** elementi, oppure vuota la pila se essa contiene meno di ***k*** elementi.

3

Se la pila contiene ***m*** elementi il ciclo while è 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.

4

Il metodo dell'aggregazione fornisce un limite più stretto.

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

Di conseguenza il numero totale di operazioni ***Pop***, comprese quelle eseguite nelle operazioni ***MultiPop***, non può superare il numero totale di operazioni ***Push*** ed è quindi minore di ***n***.

5

Se dal tempo richiesto per eseguire ***MultiPop*** togliamo il tempo per eseguire 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 del ciclo while delle operazioni ***MultiPop*** presenti nella sequenza.

6

Siccome una singola iterazione richiede tempo costante e il numero totale di iterazioni è minore di ***n*** anche 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)**.

7

incremento di un contatore binario

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

***A*[0..*k*-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

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

8

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 do
    A[i] ← 0, i ← i +1
  if i < k then
    A[i] ← 1
```

9

Una singola operazione di incremento richiede tempo ***O*(*k*)** nel caso pessimo il che fornisce un limite superiore ***O*(*nk*)** per una sequenza di ***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?

Vediamo cosa succede con un contatore di ***k*** = 8 bit.

10

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

11

Si vede che

***A*[*0*]** viene modificato ad ogni incremento del contatore,

***A*[*1*]** viene modificato ogni due incrementi,

***A*[*2*]** ogni 4 incrementi ed in generale

***A*[*i*]** viene modificato ogni 2^i incrementi.

12

Dunque 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 quindi ***O*(*n*)** e di conseguenza la complessità ammortizzata di una operazione di incremento è ***O*(*n*)/*n* = *O*(1)**.

13

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 **Θ(*nk*)**.

14

Esercizio 12.
Su di una certa struttura dati viene eseguita una sequenza di ***n*** operazioni. L'operazione ***i*-**esima costa ***i*** quando ***i*** è una potenza di **2** mentre ha costo **1** negli altri casi. Mostrare che tali operazioni hanno costo ammortizzato costante.

15

Metodo degli accantonamenti

Si caricano le operazioni meno costose di un costo aggiuntivo che 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.

Illustriamo questo metodo con i soliti due esempi.

16

operazioni su di una pila

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

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

A tali operazioni attribuiamo i seguenti *costi ammortizzati*:

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

17

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.

18

Quando eseguiamo una ***MultiPop*** le **min(*k*,*m*)** iterazioni del ciclo while vengono pagate utilizzando i **min(*k*,*m*)** crediti prepagati attribuiti uno a ciascun oggetto che viene tolto dalla pila.

Ogni operazione ha costo (ammortizzato) costante!

19

incremento di un contatore binario

```
Increment(A)
  i ← 0
  while i < k and A[i] = 1 do
    A[i] ← 0, i ← i +1
  if i < k then
    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**.

20

Costi ammortizzati

trasformazione $0 \rightarrow 1$	2
trasformazione $1 \rightarrow 0$	0

Quando eseguiamo $0 \rightarrow 1$: una delle due unità di costo è effettiva e l'altra è attribuita come credito prepagato al bit **1**.

Quindi ogni bit **1** nel contatore ha un credito prepagato, che si può usare per pagare interamente le operazioni $0 \rightarrow 1$.

21

Quindi ogni ***Increment*** ha costo ammortizzato **2**, e una sequenza di ***n*** operazioni costerà ***O(n).***

22

Esercizio 13.

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

Fare in modo che la complessità ammortizzata delle operazioni risulti costante.

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

23

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 quando si esegue una ***Push***, prima di eseguire l'operazione viene scaricata su disco una parte degli ***m*** elementi.

Se una operazione ***Pop*** toglie l'ultimo elemento in memoria e ci sono degli altri elementi registrati su disco, dopo l'operazione se ne ricarica una parte in memoria.

24

Metodo del potenziale

Si associa alla struttura dati ***D*** un potenziale **Φ(*D*)** tale che le operazioni meno costose incrementino il potenziale mentre quelle più costose portino ad una diminuzione del potenziale della struttura.

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

25

In altre parole, se indichiamo con D_i la struttura dati dopo l'esecuzione della ***i***-esima operazione e con c_i il costo effettivo della ***i***-esima operazione allora il costo ammortizzato è:

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

Il costo ammortizzato di una sequenza di ***n*** operazioni è:

$$\hat{C} = \sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} [c_i + \Phi(D_i) - \Phi(D_{i-1})]$$
$$= C + \Phi(D_n) - \Phi(D_0)$$

26

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

In caso contrario la variazione di potenziale negativa relativa all'esecuzione di tutta la sequenza deve essere compensata da un aumento adeguato del costo ammortizzato delle operazioni.

Illustriamo anche questo metodo con i soliti esempi.

27

operazioni su di una pila

Come funzione potenziale **Φ(*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
Multi_Pop	min(k,m)	-min(k,m)	0

28

Osserviamo inoltre che all'inizio quando la pila è vuota $\Phi(P_0) = 0$ mentre alla fine $\Phi(P_n) \geq 0$ per cui la differenza di potenziale corrispondente all'esecuzione di tutta la sequenza di operazioni è non negativa.

29

incremento di un contatore binario

Scegliamo come funzione potenziale **Φ(*A*)** il numero bit **1** presenti nel contatore.

Ricordiamo che il costo effettivo di una operazione ***Increment*** è pari al numero di bit modificati e che tra questi vi è un certo numero $t \geq 0$ di **1** trasformati in **0** e al più un solo **0** trasformato in **1** per cui:

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

30

Osserviamo che l'esecuzione dell'intera sequenza di operazioni comporta una differenza di potenziale non negativa.

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

Con il metodo del potenziale possiamo calcolare il costo ammortizzato dell'incremento di un contatore binario di ***k*** bit anche quando non si parte da **0** ma da un valore qualsiasi.

31

In questo caso la differenza di potenziale relativa ad una sequenza di ***n*** incrementi può risultare negativa ma pur sempre in modulo minore o uguale di ***k***.

Un incremento del costo ammortizzato di ***k*/*n*** unità di costo è quindi sufficiente a compensare la differenza di potenziale negativa.

Il costo ammortizzato di ***Increment*** è quindi $O(1+k/n)$ che nel caso in cui $k = O(n)$ si riduce ad $O(1)$.

32

Esercizio 15.

Realizzare una coda ***Q*** di tipo FIFO utilizzando due normali pile P_1 e 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.

33