Algoritmi e Strutture Dati

Analisi ammortizzata

Alberto Montresor and Davide Rossi

Università di Bologna

21 novembre 2024

Sommario

Introduzione

Analisi ammortizzata

Una tecnica di analisi di complessità che valuta il tempo richiesto per eseguire, nel caso pessimo, una sequenza di operazioni su una struttura dati

- Esistono operazioni più o meno costose.
- Se le operazioni più costose sono poco frequenti, allora il loro costo può essere ammortizzato dalle operazioni meno costose

Importante differenza

- Analisi caso medio: Probabilistica, su singola operazione
- Analisi ammortizzata: Deterministica, su operazioni multiple, caso pessimo

Metodi per l'analisi ammortizzata

Metodo dell'aggregazione

- Si calcola la complessità $T(n)$ per eseguire n operazioni in sequenza nel caso pessimo
- Tecnica derivata dalla matematica

Metodo degli accantonamenti

- Alle operazioni vengono assegnati costi ammortizzati che possono essere maggiori/minori del loro costo effettivo
- Tecnica derivata dall'economia

Metodo del potenziale

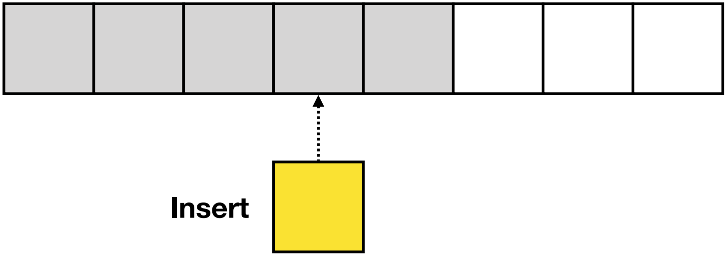
- Lo stato del sistema viene descritto con una funzione di potenziale
- Tecnica derivata dalla fisica

Sommario

Vettori dinamici – Espansione

Sequenze implementate tramite vettori dinamici

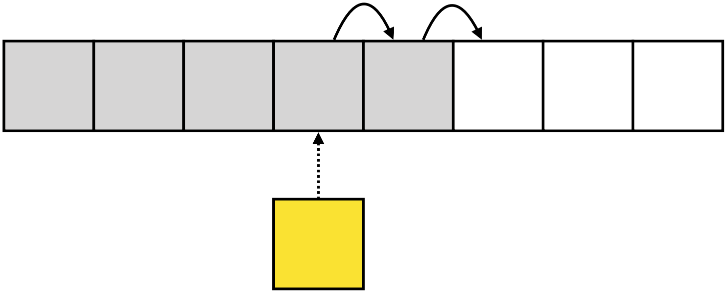
- Si alloca un vettore di una certa dimensione detta capacità
- L'inserimento di un elemento "in mezzo" ha costo $O(n)$
- Inserire un elemento "in fondo" alla sequenza (append) ha costo $O(1)$



Vettori dinamici – Espansione

Sequenze implementate tramite vettori dinamici

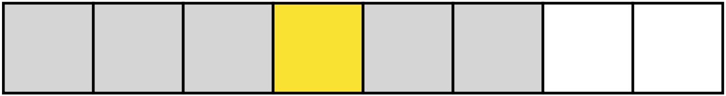
- Si alloca un vettore di una certa dimensione detta capacità
- L'inserimento di un elemento "in mezzo" ha costo $O(n)$
- Inserire un elemento "in fondo" alla sequenza (append) ha costo $O(1)$



Vettori dinamici – Espansione

Sequenze implementate tramite vettori dinamici

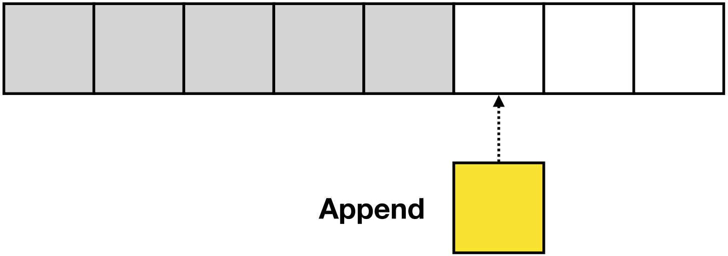
- Si alloca un vettore di una certa dimensione detta capacità
- L'inserimento di un elemento "in mezzo" ha costo $O(n)$
- Inserire un elemento "in fondo" alla sequenza (append) ha costo $O(1)$



Vettori dinamici – Espansione

Sequenze implementate tramite vettori dinamici

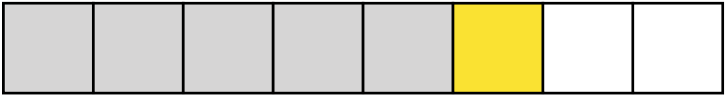
- Si alloca un vettore di una certa dimensione detta capacità
- L'inserimento di un elemento "in mezzo" ha costo $O(n)$
- Inserire un elemento "in fondo" alla sequenza (append) ha costo $O(1)$



Vettori dinamici – Espansione

Sequenze implementate tramite vettori dinamici

- Si alloca un vettore di una certa dimensione detta capacità
- L'inserimento di un elemento "in mezzo" ha costo $O(n)$
- Inserire un elemento "in fondo" alla sequenza (append) ha costo $O(1)$



Vettori dinamici – Espansione

Problema

- Non è noto a priori quanti elementi entreranno nella sequenza
- La capacità selezionata può rivelarsi insufficiente.

Soluzione

- Si alloca un vettore di capacità maggiore, si ricopia il contenuto del vecchio vettore nel nuovo e si rilascia il vecchio vettore
- Esempi: `java.util.Vector`, `java.util.ArrayList`

Vettori dinamici in Java

```java
private Object[] buffer = new Object[INITSIZE];

// Raddoppiamento
private void doubleStorage() {
  Object[] newb = new Object[2*buffer.length];
  System.arraycopy(buffer,0, newb,0, buffer.length);
  buffer = newb;
}

// Incremento fisso
private void incrementStorage() {
  Object[] newb = new Object[buffer.length+INCREMENT];
  System.arraycopy(buffer,0, newb,0, buffer.length);
  buffer = newb;
}
```

Vettori dinamici – Espansione

Domanda

Qual è il migliore fra i due?

- Raddoppiamento
- Incremento costante

Vettori dinamici – Espansione

Domanda

Qual è il migliore fra i due?

- Raddoppiamento
- Incremento costante

Dalla documentazione Java: ArrayList.add():

`As elements are added to an ArrayList, its capacity grows automatically. The details of the growth policy are not specified beyond the fact that adding an element has constant amortized time cost.`

Vettori dinamici in Java – Quale approccio?

```
private Object[] buffer = new Object[INITSIZE];

// Raddoppiamento - Utilizzato in ArrayList (1.2)
private void doubleStorage() {
  Object[] newb = new Object[2*buffer.length];
  System.arraycopy(buffer,0, newb,0, buffer.length);
  buffer = newb;
}

// Incremento fisso - Utilizzato in Vector (1.0)
private void incrementStorage() {
  Object[] newb = new Object[buffer.length+INCREMENT];
  System.arraycopy(buffer,0, newb,0, buffer.length);
  buffer = newb;
}
```

Analisi ammortizzata, raddoppiamento del vettore

Costo effettivo di un'operazione add():

$$c_i = \begin{cases} i & \exists k \in \mathbb{Z}_0^+ : i = 2^k + 1 \\ 1 & \text{altrimenti} \end{cases}$$

Assunzioni:

- Dimensione iniziale: 1
- Costo di scrittura di un elemento: 1

n	costo
1	1
2	$1 + 2^0 = 2$
3	$1 + 2^1 = 3$
4	1
5	$1 + 2^2 = 5$
6	1
7	1
8	1
9	$1 + 2^3 = 9$
10	1
11	1
12	1
13	1
14	1
15	1
16	1
17	$1 + 2^4 = 17$

Analisi ammortizzata, raddoppiamento del vettore

Costo effettivo di n operazioni `add()`:

$$
\begin{aligned}
T(n) &= \sum_{i=1}^{n} c_i \\
&= n + \sum_{j=0}^{\lfloor \log n \rfloor} 2^j \\
&= n + 2^{\lfloor \log n \rfloor + 1} - 1 \\
&\leq n + 2^{\log n + 1} - 1 \\
&= n + 2n - 1 = O(n)
\end{aligned}
$$

Costo ammortizzato di un'operazione `add()`:

$$T(n)/n = \frac{O(n)}{n} = O(1)$$

Analisi ammortizzata, incremento del vettore

Costo effettivo di un'operazione add():

$$c_i = \begin{cases} i & (i \bmod d) = 1 \\ 1 & \text{altrimenti} \end{cases}$$

Assunzioni:

- Incremento: d
- Dimensione iniziale: d
- Costo di scrittura di un elemento: 1

Nell'esempio:

- $d = 4$

n	costo
1	1
2	1
3	1
4	1
5	$1 + d = 5$
6	1
7	1
8	1
9	$1 + 2d = 9$
10	1
11	1
12	1
13	$1 + 3d = 13$
14	1
15	1
16	1
17	$1 + 4d = 17$

Analisi ammortizzata, incremento del vettore

Costo effettivo di n operazioni $\mathsf{add}()$:

$$
\begin{aligned}
T(n) &= \sum_{i=1}^{n} c_i \\
&= n + \sum_{j=1}^{\lfloor n/d \rfloor} d \cdot j \\
&= n + d \sum_{j=1}^{\lfloor n/d \rfloor} j \\
&= n + d \frac{(\lfloor n/d \rfloor + 1)\lfloor n/d \rfloor}{2} \\
&\leq n + \frac{(n/d+1)n}{2} = O(n^2)
\end{aligned}
$$

Costo ammortizzato di un'operazione $\mathsf{add}()$:

$$T(n)/n = \frac{O(n^2)}{n} = O(n)$$

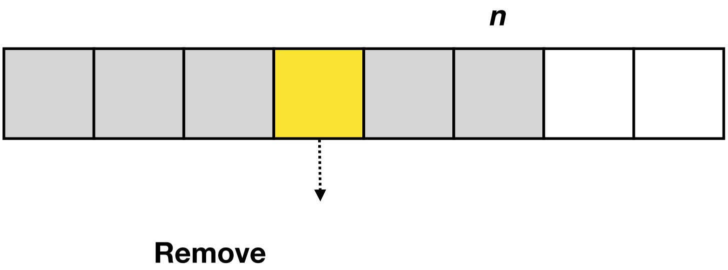
Reality check

Linguaggio	Struttura dati	Fattore espansione
GNU C++	`std::vector`	2.0
Microsoft VC++ 2003	`vector`	1.5
C#	`List`	2.0
Python	`list`	1.125
Java OpenJDK	`ArrayList`	2.0

Vettori dinamici – Cancellazione

Domande

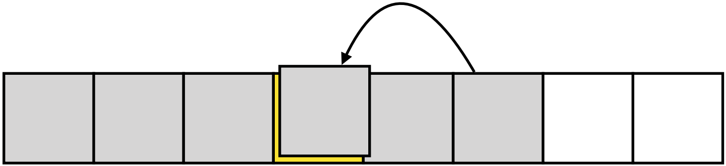
- Quanto costa togliere un elemento da un vettore?
- Quanto costa togliere un elemento da un vettore non ordinato?



Vettori dinamici – Cancellazione

Domande

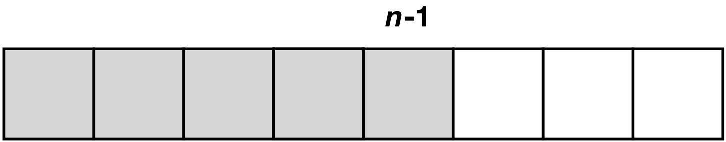
- Quanto costa togliere un elemento da un vettore?
- Quanto costa togliere un elemento da un vettore non ordinato?



Vettori dinamici – Cancellazione

Domande

- Quanto costa togliere un elemento da un vettore?
- Quanto costa togliere un elemento da un vettore non ordinato?



Vettori dinamici – Cancellazione

Contrazione

Per ridurre lo spreco di memoria, è opportuno contrarre il vettore quando il fattore di carico $\alpha = \mathit{dim}/\mathit{capacità}$ diventa troppo piccolo

- dim: numero di elementi attualmente presenti
- Contrazione $\rightarrow$ allocazione, copia, deallocazione

Domanda

Quale soglia per il fattore di carico?

Vettori dinamici – Cancellazione

Strategia naif

Una strategia che sembra ovvia è dimezzare la memoria quando il fattore di carico α diventa $\frac{1}{2}$

Dimostrare che questa strategia può portare ad un costo ammortizzato lineare

Vettori dinamici – Cancellazione

Strategia naif

Una strategia che sembra ovvia è dimezzare la memoria quando il fattore di carico α diventa $\frac{1}{2}$

Dimostrare che questa strategia può portare ad un costo ammortizzato lineare

Considerate la seguente sequenza di Inserimenti / Rimozioni in un vettore di capacità 8:

```
Ops: I I I I I R I R I R I R I
Dim: 1 2 3 4 5 4 5 4 5 4 5 4 5
Cap: 8 8 8 8 8 4 8 4 8 4 8 4 8
```

Vettori dinamici – Cancellazione

Qual è il problema?

- Non abbiamo un numero di inserimenti/rimozioni sufficienti per ripagare le espansioni/contrazioni.

Dobbiamo lasciar decrescere il sistema fino a $\alpha = \frac{1}{4}$

- Dopo una contrazione, il fattore di carico diventa $\frac{1}{2}$ (esattamente come dopo un'espansione)

Analisi ammortizzata: usiamo una funzione di potenziale che:

- Vale 0 all'inizio e subito dopo un'espansione o contrazione
- Cresce fino a raggiungere il numero di elementi presenti nella tavola quando α aumenta fino ad 1 o diminuisce fino ad 1/4

Vettori dinamici: contrazione

Funzione di potenziale

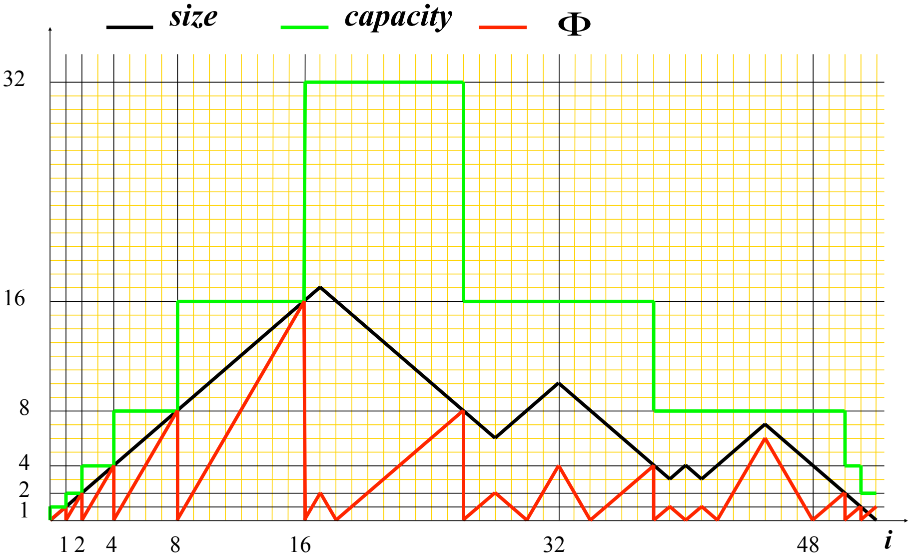
$$\Phi = \begin{cases} 2 \cdot \mathit{dim} - \mathit{capacità} & \alpha \geq \frac{1}{2} \\ \mathit{capacità}/2 - \mathit{dim} & \alpha \leq \frac{1}{2} \end{cases}$$

Alcuni casi esplicativi:

- $\alpha = \frac{1}{2}$ (dopo espansione/contrazione) $\Rightarrow \Phi = 0$
- $\alpha = 1$ (prima di espansione) $\Rightarrow \mathit{dim} = \mathit{capacità} \Rightarrow \Phi = \mathit{dim}$
- $\alpha = \frac{1}{4}$ (prima di contrazione) $\Rightarrow \mathit{capacità} = 4 \cdot \mathit{dim} \Rightarrow \Phi = \mathit{dim}$

In altre parole: subito prima di espansioni e contrazioni il potenziale è sufficiente per "pagare" il costo della copia dei *dim* valori

Vettori dinamici: contrazione



Vettori dinamici: contrazione

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

$$\begin{aligned}
a_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2 \cdot dim_i - capacità_i) - (2 \cdot dim_{i-1} - capacità_{i-1}) \\
&= 1 + 2 \cdot (dim_{i-1} + 1) - capacità_{i-1} - 2 \cdot dim_{i-1} + capacità_{i-1} \\
&= 3
\end{aligned}$$

Se $\alpha = 1$ il costo ammortizzato di un inserimento con espansione è:

$$\begin{aligned}
a_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + dim_{i-1} + (2 \cdot dim_i - capacità_i) - (2 \cdot dim_{i-1} - capacità_{i-1}) \\
&= 1 + dim_{i-1} + 2 \cdot (dim_{i-1} + 1) - 2 \cdot dim_{i-1} - 2 \cdot dim_{i-1} + dim_{i-1} \\
&= 3
\end{aligned}$$

[*Esercizio: Altri casi per valori differenti di* α *e per contrazione*]

Sommario

Conclusioni

Esempi di applicazione dell'analisi ammortizzata

- Espansione / contrazione di tabelle hash
- Insiemi disgiunti con euristica sul rango e compressione dei cammini
- Heap di Fibonacci
- ...