Aggregazione di dati elementari

Strategia più semplice: disporli "l'uno di seguito all'altro" a formare una sequenza lineare, identificando ciascun dato con la posizione occupata

Due modi di realizzarla



Riflettono due diversi modi di allocazione dei dati in memoria

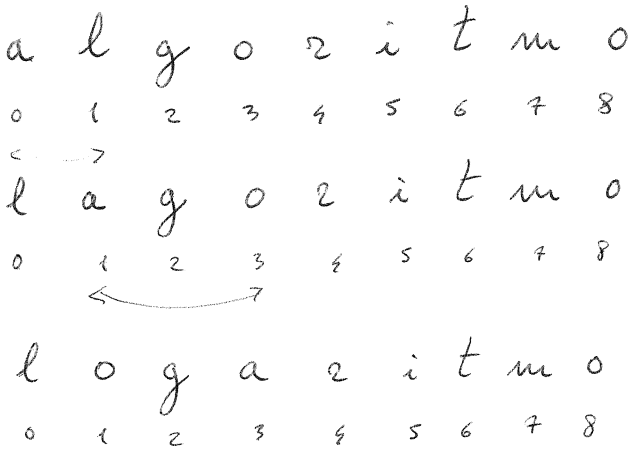
Più formalmente

Sequenza lineare = insieme finito di elementi disposti consecutivamente, in cui ognuno ha associato un indice di posizione, a partire dall'indice 0. Indichiamo, quindi, con

$$a_0\ a_1\ a_2\ \dots\ a_{n-1}$$

una sequenza lineare di n elementi, dove in posizione j si trova l'elemento $(j+1)$-esimo.

In una sequenza l'ordine tra gli elementi è importante.



L'operazione più elementare su una sequenza lineare consiste nell'accesso ai suoi elementi, specificandone la posizione.

Nella sequenza precedente, quando richiesto l'elemento in posizione 7 si ottiene $a_7 = m$

L'accesso diretto significa che, dato un indice i, accediamo direttamente all'elemento a_i, senza dover attraversare la sequenza

L'accesso ha un costo uniforme, indipendente dall'indice di posizione.

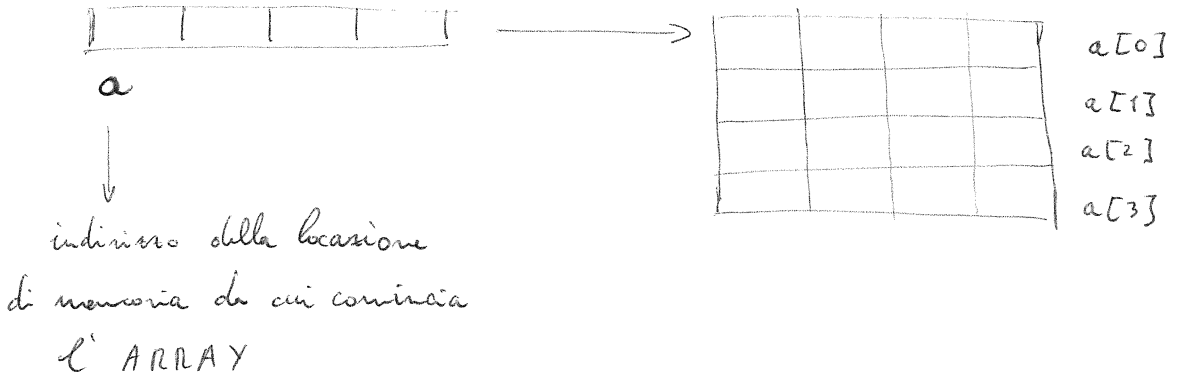
Indichiamo con il termine ARRAY le sequenze lineari ad accesso diretto e con $a[i]$ l'$(i+1)$-esimo elemento di una array a.

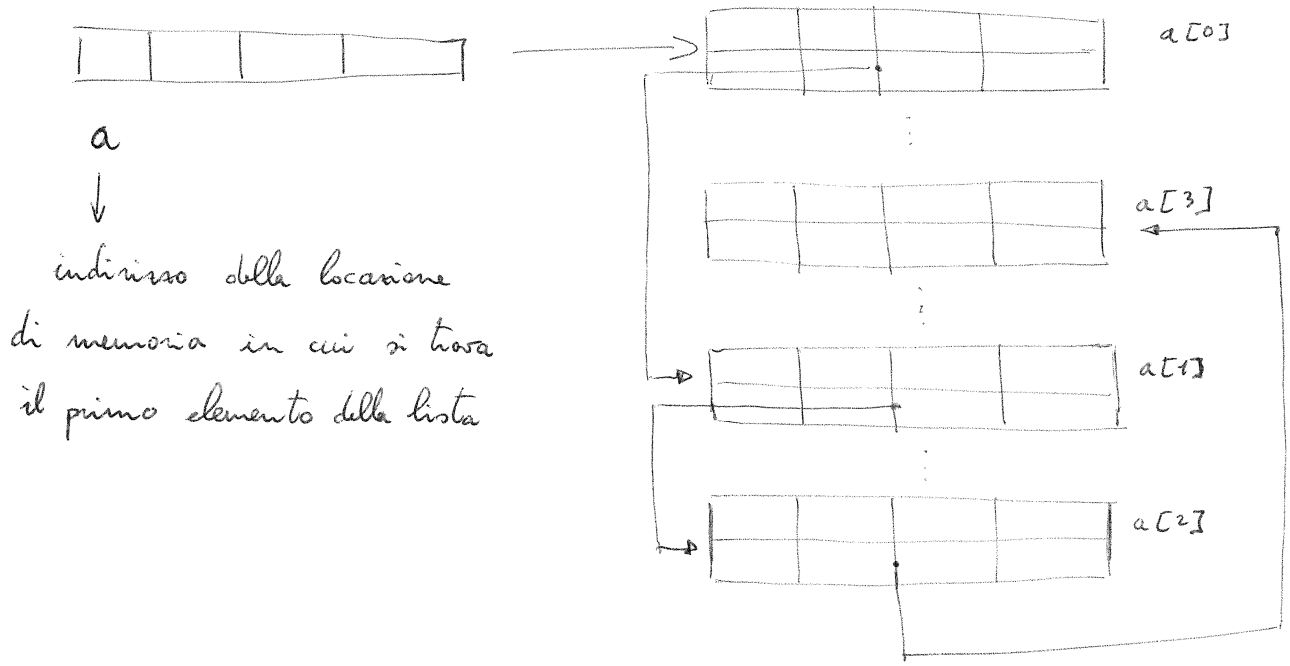
L'altra modalità consiste nel raggiungere l'elemento desiderato a partire dal primo. Viene detta accesso sequenziale.

L'accesso ha un costo che è $O(i)$, per accedere all'elemento in posizione i.

Indichiamo con il termine LISTE le sequenze lineari ad accesso sequenziale.

Allocazione di memoria (Memoria con indirizzi e locazioni di 1 BYTE)





Ogni elemento della lista contiene un altro campo con l'indirizzo dell'elemento successivo della lista.

Le locazioni di memoria in un ARRAY sono contigue, quelle di una LISTA sparse in memoria.

ARRAY di dimensione variabile

Realizzano, come le liste, sequenze lineari dinamiche.
Offerti da diversi linguaggi di programmazione (C++, C#, JAVA)

Idea: riducono o aumentano la taglia dell'ARRAY a seconda degli inserimenti e delle cancellazioni di elementi dalla sequenza.

Ricopiano gli elementi in un altro ARRAY più piccolo o più grande.

Osservazione: effettuare l'operazione di copia in un nuovo ARRAY, più piccolo o più grande, AD OGNI inserimento o cancellazione avrebbe un costo che è $O(n)$, dove n indica il numero di elementi presenti nell'ARRAY al momento dell'operazione. Non va bene...

Con qualche accorgimento possiamo "pagare" $O(n)$ per $\Omega(n)$ operazioni di inserimento e cancellazione.

Sia d la taglia dell'ARRAY a (celle di memoria inizialmente allocate) e sia $n \leq d$, il numero di elementi effettivamente contenuti nella sequenza memorizzata in a.

STRATEGIA

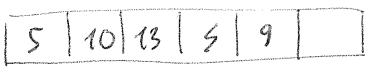
Fin quando c'è spazio, aumentiamo n di una unità.
Se $n = d$, allochiamo un array b di taglia $2 \cdot d$ e copiamo in esso gli elementi. Poniamo poi $a = b$

Viceversa, quando $n = d/4$, allochiamo un array b di taglia $d/2$ e ricopiamo in esso gli n elementi. In codice:

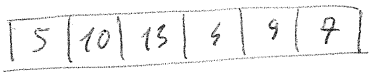
```
VerificaRaddoppio()
IF (n == d) {
    b = NuovoArray(2d);
    FOR (i=0; i<n; i=i+1)
        b[i] = a[i];
    a = b
    d = 2d;
}
```

```
VerificaDimezzamento()
IF ((d>1) && (n == d/4)) {
    b = NuovoArray(d/2)
    FOR (i=0; i<n; i=i+1)
        b[i] = a[i];
    a = b;
    d = d/2;
}
```

Esempio



$d = 6$, $n = 5$



$d = 6$, $n = 6$ (aggiunto dell'ins. di 7)

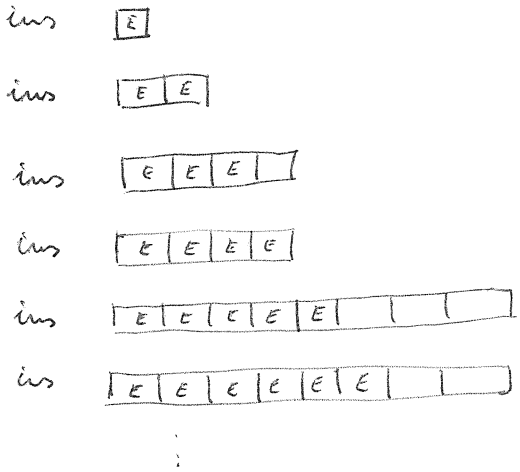


$d = 12$, $n = 7$

Osservazione: NON c'è ridimensionamento ad ogni operazione

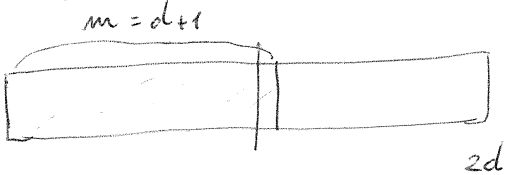
Teorema. L'esecuzione di n operazioni di inserimento e cancellazione in un array dinamico richiede tempo $O(n)$. Inoltre la taglia d è $O(n)$.

Dimostrazione. Per prendere familiarità, prima di tutto, vediamo cosa accade ($d = 1$, $n = 1$) con una sequenza di inserimenti.



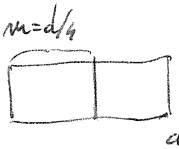
C'è un raddoppio quando $n = 2^k$.

In generale, subito dopo un raddoppio, ci sono $n = d+1$ elementi nell'ARRAY di taglia $2d$.



Occorrono almeno
- $n-1$ inserimenti per un nuovo raddoppio
- $n/2$ cancellazioni per un dimezzamento

Invece, subito dopo un dimezzamento, ci sono $n = d/4$ elementi nell'ARRAY di taglia $d/2$.

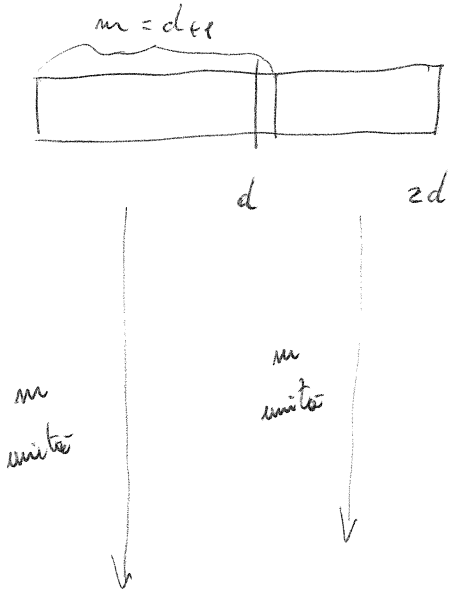


Occorrono almeno
- $n+1$ inserimenti per un raddoppio
- $n/2$ cancellazioni per un nuovo dimezzamento

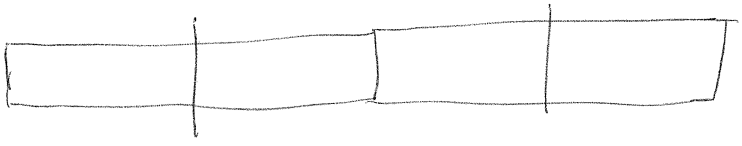
In tutti i casi il costo del ridimensionamento (raddoppio o dimezzamento) può essere "virtualmente distribuito" sulle operazioni che lo hanno generato a partire dall'ultimo ridimensionamento.

Per rendercene conto, supponiamo che il costo virtuale di un inserimento sia 3 unità dove:

- 1 unità serve per inserire l'elemento
- 1 unità serve per spostarlo quando c'è un raddoppio
- 1 unità serve per spostare un elemento della "prima metà" quando c'è un raddoppio



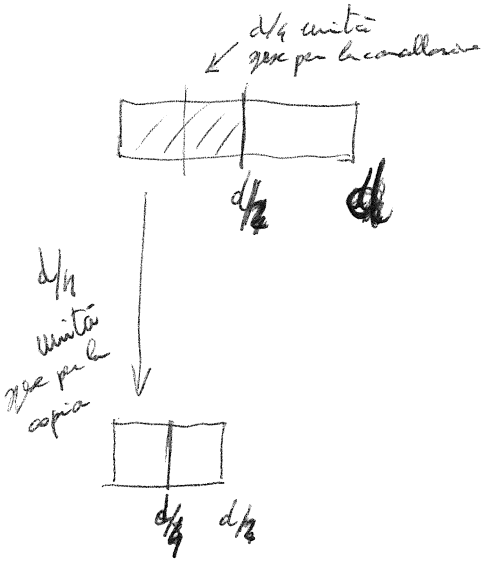
A seguito dell'ultimo raddoppio dalla taglia d a 2d, l'inserimento di m=d elementi ha prodotto un valore pari a 3m di cui m unità sono per l'inserimento effettivo degli m elementi.
Le restanti 2m unità servono per pagare il nuovo raddoppio.



- m per spostare nel nuovo array i primi m elementi del vecchio
- m per spostare nel nuovo array gli ultimi m elementi inseriti nel vecchio array.

Pertanto, un costo virtuale di 3 unità per inserimento, "paga" i raddoppi quando si verificano.

Allo stesso modo si può far vedere che un costo virtuale di 2 unità per cancellazione paga i dimezzamenti quando si verificano.



Il costo virtuale di 2 unità si spende in:

- un'unità per cancellare l'elemento
- un'unità per spostare un elemento della parte rimanente nel nuovo array

Le d/4 cancellazioni che producono il dimezzamento generano un valore di $2 \cdot (d/4)$.

Di questo valore

- d/4 unità si spendono per cancellare i d/4 elementi che generano il dimezzamento
- d/4 unità si spendono per copiare i primi d/4 elementi del vecchio ARRAY nel nuovo

Pertanto, un costo virtuale di 2 unità per cancellazione, "paga" i dimezzamenti quando si verificano.

Discende, quindi, che una sequenza di n inserimenti e cancellazioni ha un costo complessivo pari a $O(n)$.

Osservazione. Se il dimezzamento avvenisse quando $n = d/2$ anziché quando $n = d/4$ il teorema non è più vero.

Infatti basta osservare cosa accade alla seguente sequenza di $n = 2^k$ elementi definita da:

- $n/2$ inserimenti
- $n/2$ inserimento / cancellazione alternati

Poiché $n/2 = \frac{2^k}{2} = 2^{k-1}$, il primo inserimento della seconda metà della sequenza, provoca un raddoppio della tabella dalla taglia $d = n/2$ alla taglia $d = n$.

La successiva cancellazione un dimezzamento dalla taglia d alla taglia $d = n/2$.

E così via ... raddoppi e dimezzamenti alternati!

Ogni operazione della seconda metà della sequenza richiede la copia di $n/2$ elementi in un nuovo ARRAY, più piccolo o più grande.

Pertanto, la seconda metà della sequenza ha un costo $\overset{\text{pari a}}{\vee}\ \Theta(n^2)$.

In conclusione, se il dimezzamento avvenisse quando $n = d/2$, non è più vero che una sequenza di n operazioni ha un costo temporale pari a $O(n)$.