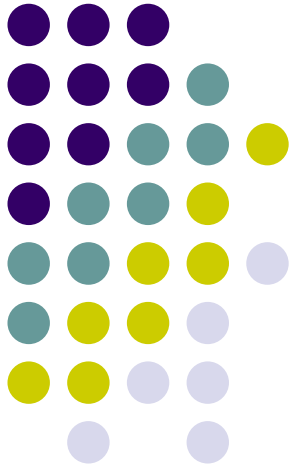
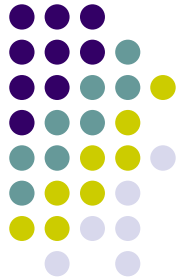
Array List (vettore)

Corso: Strutture Dati

Docente: Annalisa De Bonis



Le sequenze



- In questo corso esamineremo le sequenze

  Array list (vettore), Node list e Sequence

  - Rappresentano insiemi di elementi disposti secondo un ordine lineare
  - Supportano metodi per accedere, inserire e cancellare elementi in posizioni arbitrarie
- Queue, Stack e Deque possono essere visti come sequenze particolari in cui possiamo inserire, cancellare e avere accesso solo ad elementi particolari (primo/ultimo)

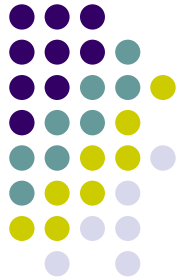
Array List, Position List e Sequence

- Il TDA Array List è un'estensione della struttura dati *concreta* array
  - Si accede agli elementi attraverso il loro indice
    - L'indice di un elemento specifica la posizione dell'elemento all'interno dell'ordinamento lineare (primo, secondo, ecc.) e non la posizione fisica dell'elemento
- Il TDA Node List e` la versione orientata agli oggetti della struttura dati *concreta* lista linkata
  - La versione astratta del nodo e` il TDA Position
- Il TDA Sequence unifica i TDA Array List e Node List

Il TDA Array List

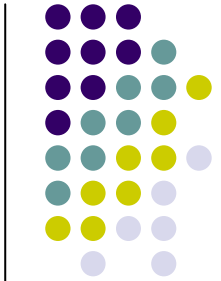
- E` una sequenza lineare che permette di accedere, inserire e cancellare un elemento attraverso il suo indice
  - L'indice di un elemento (index) e` il numero di elementi che lo precedono
  - L'indice del primo elemento e` 0
  - In generale l'i-esimo elemento del vettore ha indice i-1
- Se si specifica un indice sbagliato si verifica un errore
  - Esempio: indice negativo o maggiore di n-1 nel caso di un vettore di n elementi

Operazioni su Array List



- get(i) restituisce, senza rimuoverlo, l'elemento di indice i di V
- set(i, e) restituisce e rimpiazza in V l'elemento di indice i con l'elemento e
- add(i, e) inserisce in V un elemento e con indice i
- remove (i) restituisce e rimuove da V l'elemento di indice i
- size()
- IsEmpty()

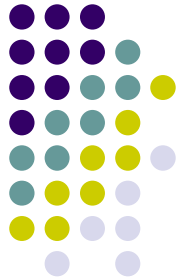
Add e remove



- Dopo aver eseguito add(i, e) tutti gli elementi che prima avevano indice maggiore o uguale di i dopo l'inserimento hanno il loro indice incrementato di 1

- Dopo aver eseguito remove(i) tutti gli elementi che prima avevano indice maggiore di i dopo la rimozione hanno il loro indice decrementato di 1

Interfaccia IndexList associata al TDA Array List



- Richiede l'eccezione IndexOutOfBoundsException che viene lanciata quando si specifica come argomento di un metodo un indice sbagliato

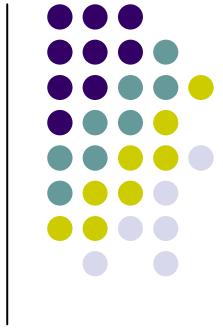
Interfaccia IndexList associata al TDA Array List

n=numero di elementi nel vettore

```
public interface IndexList <E> {

//restituisce e rimuove l'elemento di indice i
// e lancia un'eccezione se i<0 oppure i>n-1
public E remove(int i) throws
IndexOutOfBoundsException;

//inserisce l'elemento di indice i
// e lancia un' eccezione se i<0 oppure i>n
public void add(int i, E e) throws
IndexOutOfBoundsException;
```

Continua nella prossima slide

Interfaccia IndexList

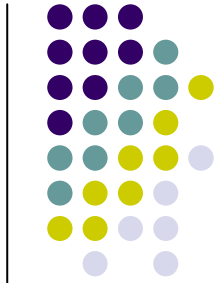


n=numero di elementi nel vettore

```
//rimpiazza con e l'elemento di indice i restituendolo in output
// e lancia un'eccezione se i <0 oppure i>n-1
public E set(int i, E e) throws IndexOutOfBoundsException;
//restituisce l'elemento di indice
// e lancia un' eccezione se i<0 oppure i>n-1
public E get(int i) throws IndexOutOfBoundsException;
```

Continua nella prossima slide

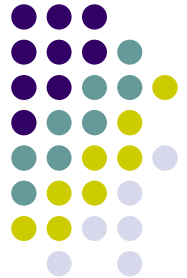
Interfaccia IndexList



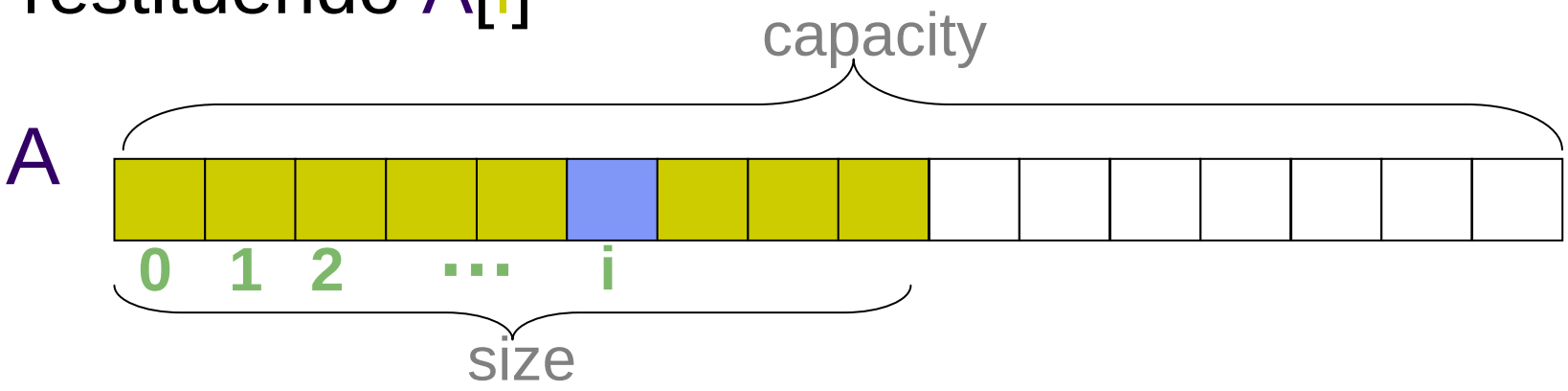
```
public boolean isEmpty();

public int size();

}
```

NB: la classe java.util.ArrayList fornisce tutti i medodi della nostra interfaccia IndexList più altri metodi aggiuntivi

Un'implementazione di IndexList basata sugli array

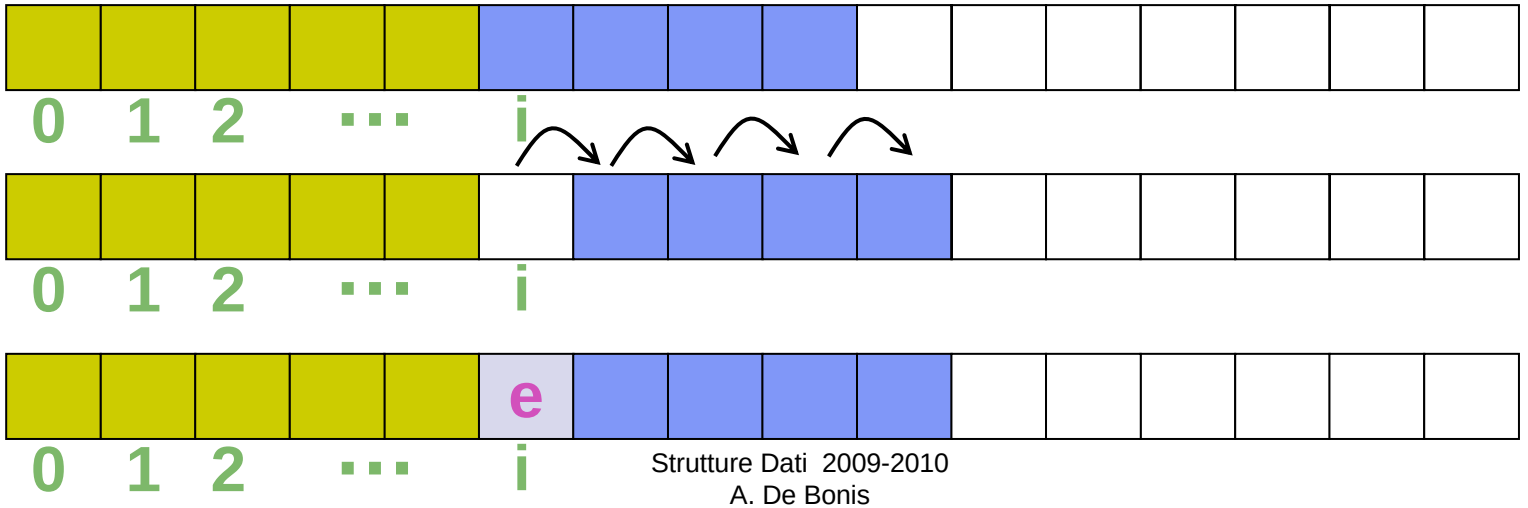


- Usa un array A di dimensione capacity
- Una variabile size mantiene traccia del numero di elementi presenti nel vettore
- get(i) e` implementata in tempo $O(1)$ restituendo A[i]



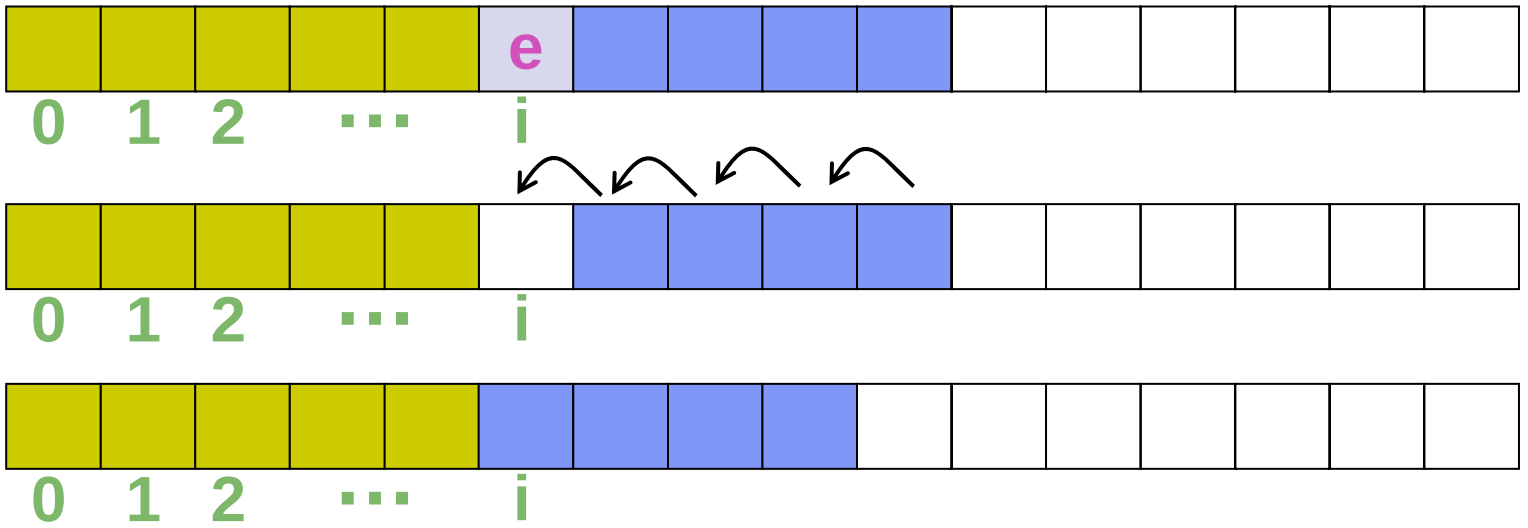
Un'implementazione di IndexList basata sugli array: inserimento

- Nell'operazione add(i,e) abbiamo bisogno di fare spazio al nuovo elemento spostando in avanti gli n-i elementi A[i],A[i+1],…,A[n-1]
- Nel caso pessimo (i=0) cio` richiede tempo *O*(n)
- Lancia l'eccezione FullIndexListException se size = A.length



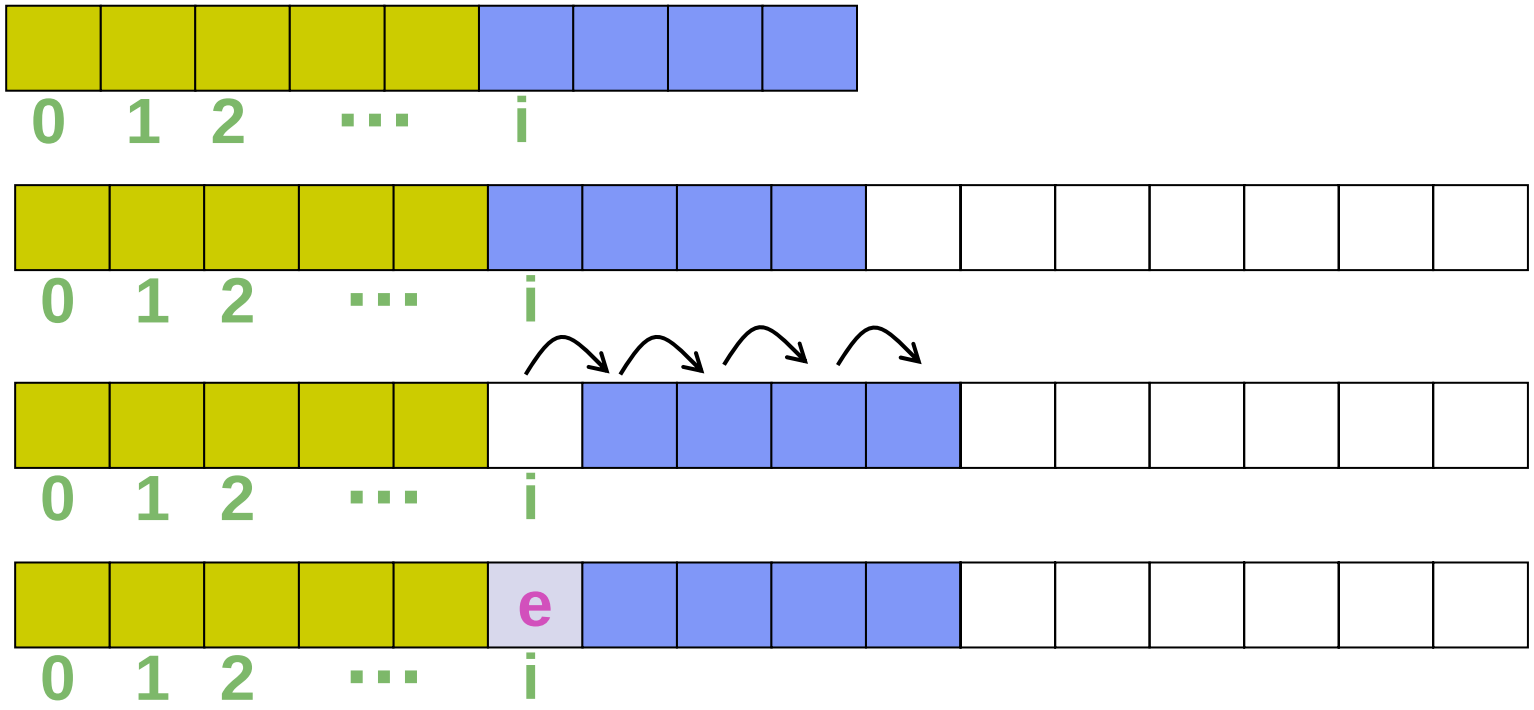
Un'implementazione di IndexList basata sugli array: cancellazione

- Nell'operazione remove(i) dobbiamo riempire il buco lasciato dall'elemento rimosso spostando indientro gli n-i-1 elementi A[i+1],A[i+2],…,A[n-1]
- Nel caso pessimo (i=0) ciò richiede tempo $O(n)$

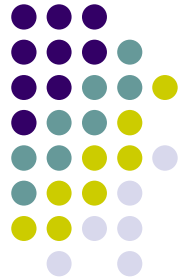


Array estensibili

- Nell'operazione add(i,e) invece di lanciare l'eccezione FullIndexListException, possiamo rimpiazzare l'array con uno piu` grande



Array estensibili: tecnica del raddoppio

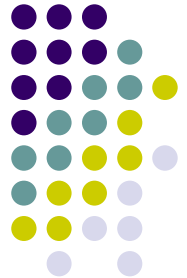


- Ogni volta che l'array si riempie lo sostituiamo con un altro grande il doppio
- Se il numero di elementi n è uguale alla lunghezza dell'array allora l'operazione di add richiede tempo O(n) per trasferire gli elementi nel nuovo array
- Si può dimostrare che n operazioni di add (effettuate a partire da un vettore vuoto) richiedono tempo O(n) per trasferire gli elementi

Array estensibili: tecnica del raddoppio

- Intuizione: subito dopo aver rimpiazzato un array di dimensione N con uno di dimensione 2N devo effettuare N nuovi inserimenti prima di trasferire nuovamente gli elementi in un nuovo array
- Analisi ammortizzata:
- Dimostriamo che il costo di n operazioni di add in un vettore inizialmente vuoto è O(n) (escluso il tempo necessario per gli shift a destra)
- Per semplicità assumiamo che ogni add aggiunga l'elemento alla fine del vettore
  - così non ci sono da fare operazioni di shift

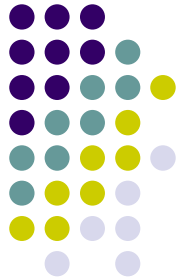
Array estensibili: tecnica del raddoppio



Analisi ammortizzata:

- Immaginiamo di pagare in dollari il lavoro del computer
- Un'operazione che richiede tempo costante richiede il pagamento di 1$
- Ogni operazione di add che NON determina l'allocazione di un nuovo array richiede tempo costante ➔ costo = 1$
- Ogni operazione di add che causa l'allocazione di un nuovo array di dimensione 2k richiede di trasferire k elementi ➔ costo = k+1 $
- Idea: pago in più le operazioni di add poco costose in modo da anticipare denaro per eventuali future operazioni di add più costose

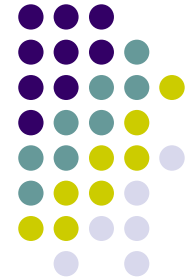
Array estensibili: tecnica del raddoppio



Analisi ammortizzata:

- Per ogni add pago 3$
  - Se l'operazione di add NON causa l'allocazione di un nuovo array allora i 2$ aggiuntivi vengono depositati
  - Se l'operazione di add causa l'allocazione di un nuovo array allora i 3$ pagati non bastano a coprire il costo dell'operazione e occorre prelevare i dollari depositati dalle precedenti operazioni di add
    - Occorre dimostrare che il numero di dollari depositati è sufficiente a pagare l'operazione

Array estensibili: tecnica del raddoppio

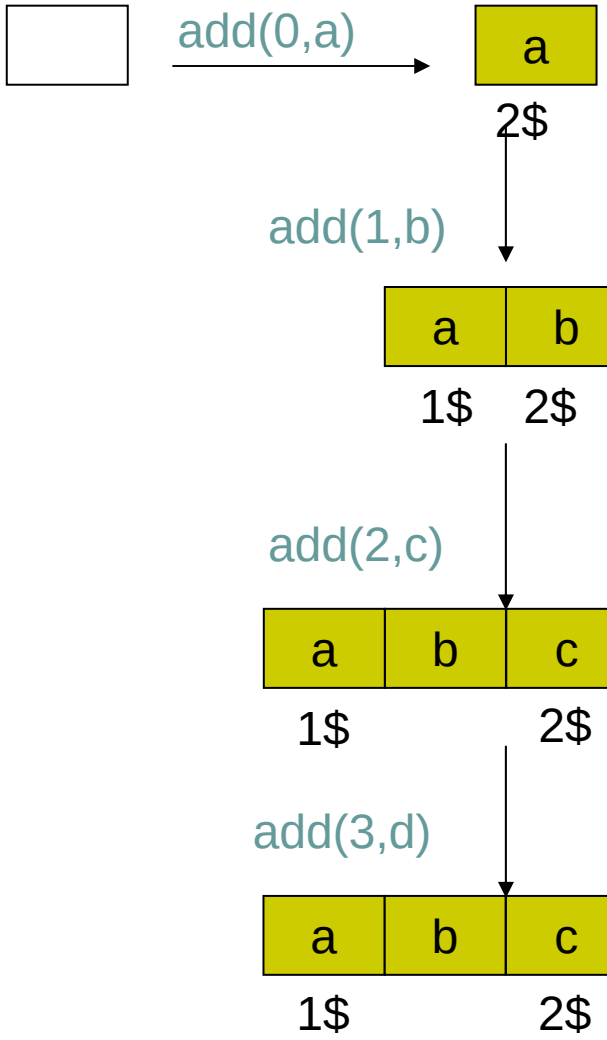


Dim:

- Supponiamo che inizialmente l'array abbia lunghezza 1
- L'(i+1)-esima volta che un'operazione di add richiede di allocare un nuovo array, gli elementi vengono trasferiti da un array di lunghezza 2^i ad un array di lunghezza 2^{i+1}
  - Ciò avviene quando l'array contiene 2^i elementi e quindi per trasferire gli elementi nel nuovo array servono 2^i \$
  - La volta precedente che gli elementi sono stati trasferiti in un nuovo array, il vettore conteneva 2^{i-1} elementi

    ➔ 2^{i-1} inserimenti devono essere stati effettuati da allora ➔ ci sono $2 \bullet 2^{i-1}$\$ = 2^i \$ depositati

Array estensibili: tecnica del raddoppio



Pago 3$: 1$ copre il costo dell'inserimento di a e 2$ vengono depositati

Pago 3$: 1$ copre il costo dell'inserimento di b e 2$ vengono depositati.

Prelevo 1$ depositato per pagare il trasferimento di a

Pago 3$: 1$ copre il costo dell'inserimento di c e 2$ vengono depositati.

Prelevo 2$ depositati per pagare il trasferimento di a e b

Pago 3$: 1$ copre il costo dell'inserimento di d e 2$ vengono depositati

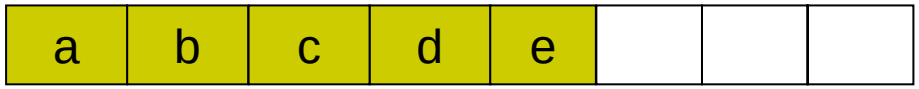
Array estensibili: tecnica del raddoppio

1$ 2$ 2$

Pago 3$: 1$ copre il costo dell'inserimento di c e 2$ vengono depositati.

Prelevo 4$ depositati per pagare il trasferimento di a, b, c, d

add(4,e)

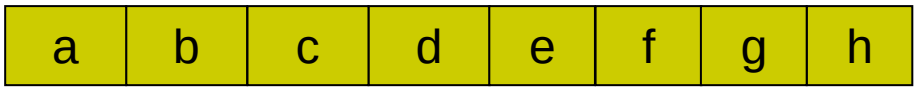


1$ 2$

Pago 3$: 1$ copre il costo dell'inserimento di c e 2$ vengono depositati.

Prelevo 4$ depositati per pagare il trasferimento di a, b, c, d

add(5,f), add(6,g),add(7,h)

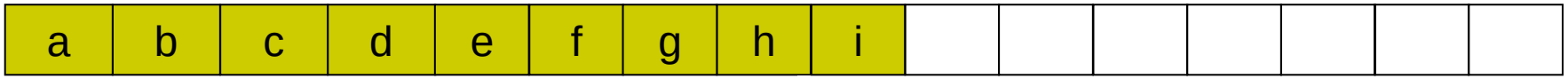


1$ 2$ 2$ 2$ 2$

Pago 3$: 1$ copre il costo dell'inserimento di c e 2$ vengono depositati.

Prelevo 8$ depositati per pagare il trasferimento di a, b, c, d,e,f,g,h

add(8,i)



1$ 2$

Array estensibili: tecnica del raddoppio

- In totale ho pagato 3•9 $ = 27$
- In generale per n operazioni di add pago 3•n $

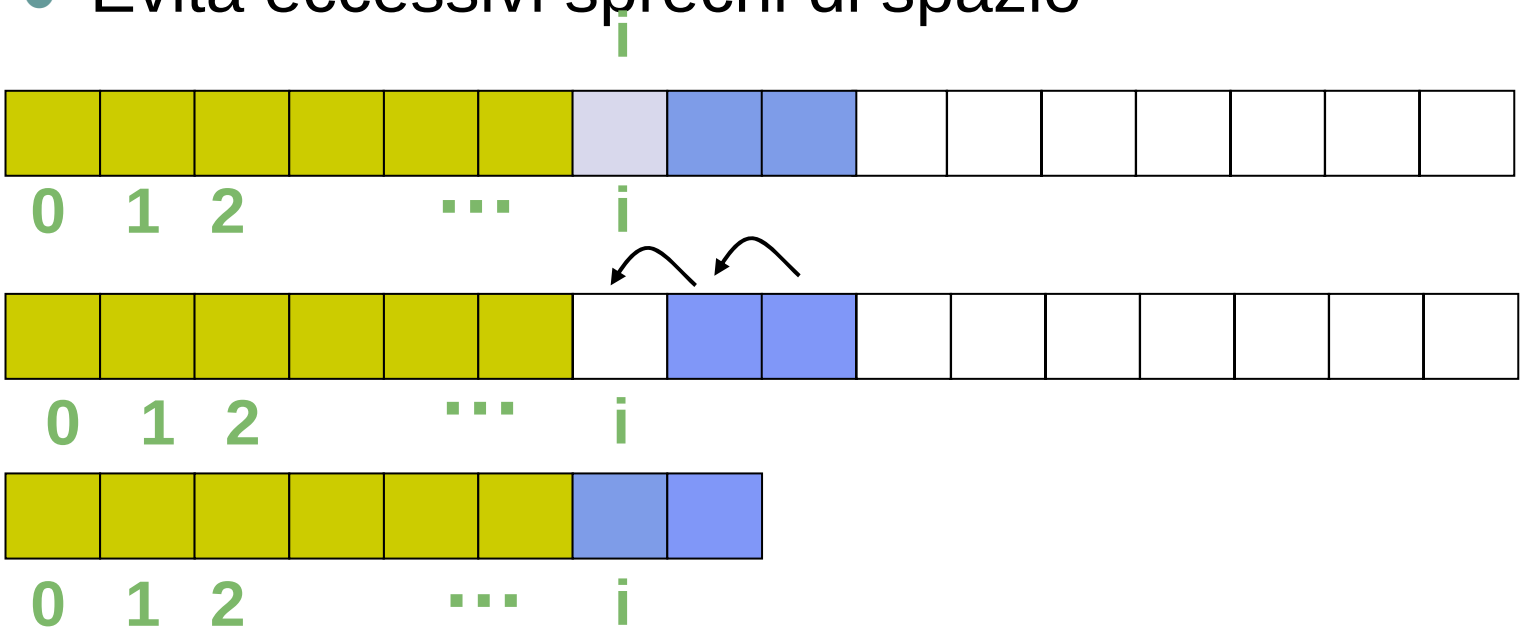
  ➔tempo di esecuzione delle n operazioni di add

  = O(n)

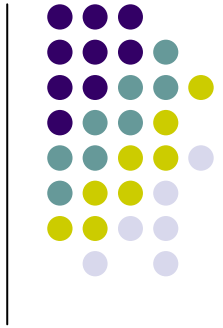
- NB: ricorda che stiamo ignorando il costo di eventuali shift

Array estensibili

- Nell'operazione remove(i) se il numero di elementi è sceso al di sotto di una certa soglia allora l'array può essere rimpiazzato da uno più piccolo
  - Evita eccessivi sprechi di spazio

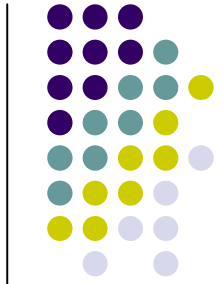


Esercizi



- Implementare l'interfaccia IndexList utilizzando un array (classe ArrayIndexList)
- Scrivere un programma di test della classe ArrayIndexList
- Implementare l'interfaccia Queue mediante i metodi di IndexList (implementato dalla classe ArrayIndexList)

Esercizi



- Implementare l'interfaccia Deque mediante i metodi di IndexList (implementato dalla classe ArrayIndexList)
- Analizzare la complessita` dei metodi di Deque con questa implementazione
- Modificare la classe ArrayIndexList in modo che i metodi addFirst e removeFirst di Deque abbiano tempo di esecuzione costante