TIPI DI DATO ASTRATTI E CONCRETI

ALGORITMI E STRUTTURE DATI

I Tipi di Dato

I Tipi di Dato

Sono “contenitori” di dati

Fissano le politiche di accesso

Es.

- First In First Out
- Last In First Out
- Ad accesso casuale (Random Access)

Tipi di Dato Astratti e Concreti

Dobbiamo distinguere tra:

- Tipi di Dato Astratti (ADT): modelli che specificano:
  - il dominio dei dati da memorizzare
  - le primitive di accesso
  - in alcuni casi, definiscono come vengono memorizzati i dati
- Tipi di Dato Concreti (CDT): implementano gli ADT e:
  - codificano ogni primitiva di accesso
  - possono fornire qualche funzionalità "fuori-specifica"

Tipi di Dato Astratti e Concreti

Dobbiamo distinguere tra:

- Tipi di Dato Astratti (ADT): modelli che specificano:
  - il dominio dei dati da memorizzare
  - le primitive di accesso
  - in alcuni casi, definiscono come vengono memorizzati i dati
- Tipi di Dato Concreti (CDT): implementano gli ADT e:
  - codificano ogni primitiva di accesso
  - possono fornire qualche funzionalità “fuori-specifica”

Due CDT possono implementare la stessa ADT in modi diversi

Array

Array

Consentono l'accesso ai dati memorizzati tramite indici

0	1	2	3	4	5	6	7	8	9
-4	0	1	2	5	6	7	11	12	13

Array

Consentono l'accesso ai dati memorizzati tramite indici

0	1	2	3	4	5	6	7	8	9
-4	0	1	2	5	6	7	11	12	13

Le primitive fornite sono:

- `create(n)` crea un array di dimensione n
- `get(i)` restituisce il valore in posizione i
- `set(i, v)` scrive il valore v in positione i
- `size()` restituisce la dimensione dell'array

Array

Consentono l'accesso ai dati memorizzati tramite indici

0	1	2	3	4	5	6	7	8	9
-4	0	1	2	5	6	7	11	12	13

Le primitive fornite sono:

- `create(n)` crea un array di dimensione n
- `get(i)` restituisce il valore in posizione i
- `set(i, v)` scrive il valore v in positione i
- `size()` restituisce la dimensione dell'array

In Python sono implementati dalla classe `list` (!?!?!).

CDT in Python: Array

```
>>> A = [2, -3, 0, 5]       # A is [2,-3,0,5]

>>> A[0]                    # the first index is 0
2

>>> A[-1]                   # the last index is -1
5                           # the 2nd last is -2, etc.

>>> A[1] = 7                # now A is [2,7,0,5]
>>> A
[2, 7, 0, 5]

>>> len(A)                  # A stores 4 values
4
```

Liste

Liste

Sono sequenze di valori che forniscono le seguenti funzionalità

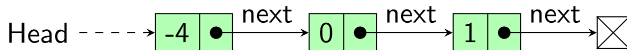
- `create()` crea una lista vuota
- `head()` restituisce il primo valore della lista
- `tail()` restituisce una "vista" alla lista priva del primo valore
- `prepend(v)` aggiunge *v* in testa alla lista
- `is_empty()` verifica se la lista è vuota

Liste Concatenate

ADT derivate dalle liste

Propongono l'organizzazione dei dati:

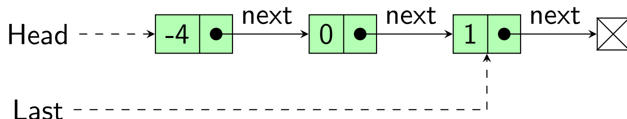
- ogni valore è contenuto in un nodo
- ogni nodo ha un riferimento al nodo successivo della lista



La lista può essere visitata dal primo elemento all'ultimo.

Liste Concatenate: Una Piccola Aggiunta

Può essere vantaggioso, mantenere un riferimento all'ultimo nodo



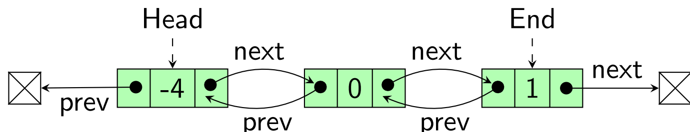
Senza dover scandire la lista, possiamo implementare

- `last()` restutuisce il valore in fondo alla lista
- `append(v)` aggiunge v in fondo alla lista
- `extend(L)` accoda la lista L

Liste Doppiamente Concatenate

Sono ADT derivate dalle liste

I nodi hanno anche un riferimento al predecessore



Possono essere visitate anche dalla fine alla testa

Senza scandire la lista, implementano anche:

- `delete_last()` cancella il valore in ultima posizione

Liste vs Array

Le liste sono ADT dinamiche:

- il numero di valori che memorizzano può variare nel tempo
- non dobbiamo specificare il numero valori contenuti in fase di creazione

Liste vs Array

Le liste sono ADT dinamiche:

- il numero di valori che memorizzano può variare nel tempo
- non dobbiamo specificare il numero valori contenuti in fase di creazione

Gli array invece sono ADT statiche:

- hanno una dimensione fissata
- durante la creazione dobbiamo dire quanti oggetti contengono

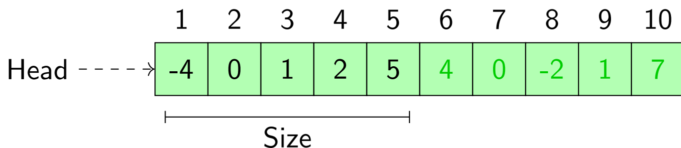
Ma consentono l'indicizzazione senza scansione che è comoda...

Possiamo "Simulare" la Liste con gli Array?

Dobbiamo conoscere la dimensione massima della lista

Poi potremmo ...

- memorizzare i valori in ordine dal primo indice
- tenere traccia della dimensione

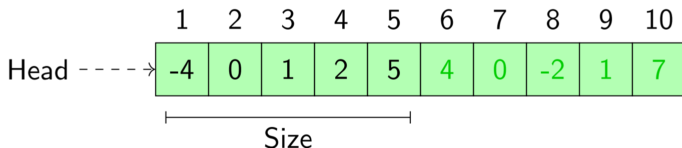


Possiamo “Simulare” la Liste con gli Array?

Dobbiamo conoscere la dimensione massima della lista

Poi potremmo ...

- memorizzare i valori in ordine dal primo indice
- tenere traccia della dimensione



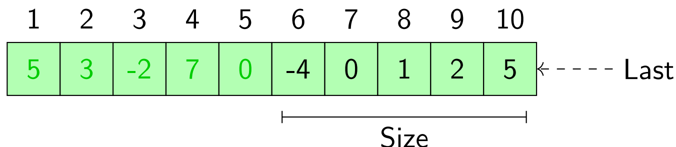
Ottimo per `append(v)` e `delete_last()`, ma... `tail()` e `prepend(v)`?

Possiamo "Simulare" la Liste con gli Array?

Dobbiamo conoscere la dimensione massima della lista

Poi potremmo ...

- memorizzare i valori in ordine dall'ultimo indice
- tenere traccia della dimensione

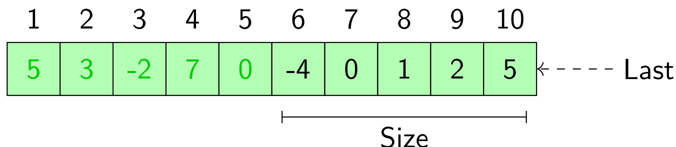


Possiamo "Simulare" la Liste con gli Array?

Dobbiamo conoscere la dimensione massima della lista

Poi potremmo ...

- memorizzare i valori in ordine dall'ultimo indice
- tenere traccia della dimensione



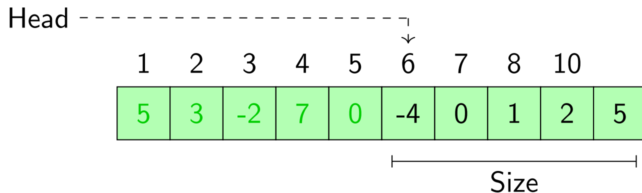
Ok per `tail()` e `prepend(v)`, ma ... `append(v)` e `delete_last()`?

Array Circolari

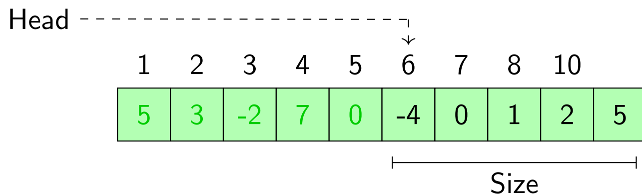
Gli Array Circolari

Rappresentano liste usando array memorizzando:

- la dimensione della lista
- l'indice del primo elemento della lista



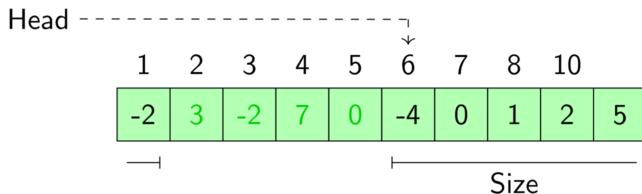
Gli Array Circolari



Per aggiungere -2 in fondo alla lista:

- inseriamo il valore in posizione (Head+Size)%max_size $= 0$
- incrementiamo di 1 la dimensione della lista

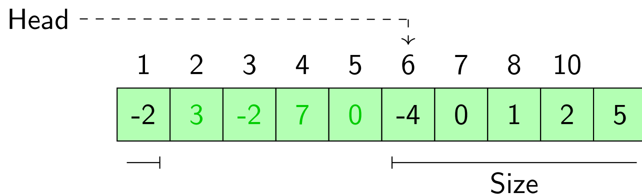
Gli Array Circolari



Per aggiungere -2 in fondo alla lista:

- inseriamo il valore in posizione (Head+Size)%max_size $= 0$
- incrementiamo di 1 la dimensione della lista

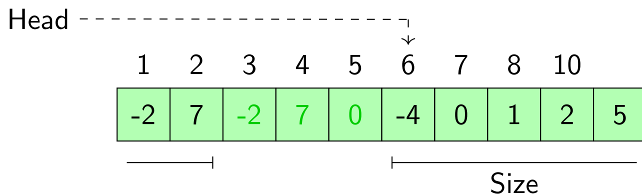
Gli Array Circolari



Per aggiungere 7 in fondo alla lista:

- inseriamo il valore in posizione (Head+Size)%max_size = 1
- incrementiamo di 1 la dimensione della lista

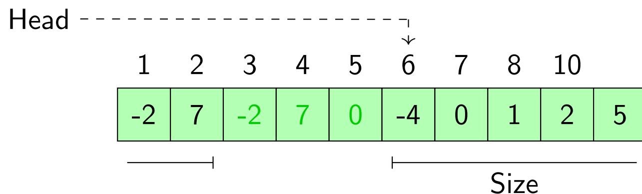
Gli Array Circolari



Per aggiungere 7 in fondo alla lista:

- inseriamo il valore in posizione (Head+Size)%max_size = 1
- incrementiamo di 1 la dimensione della lista

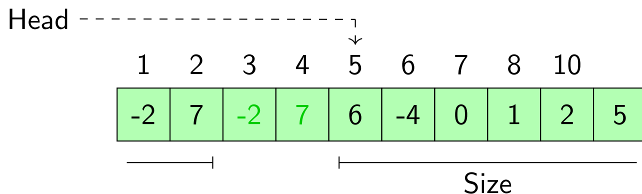
Gli Array Circolari



Per aggiungere 6 in testa alla lista:

- inseriamo il valore in posizione (Head-1)%max_size = 4
- incrementiamo di 1 la dimensione della lista
- riassegnamo l'indice della testa

Gli Array Circolari



Per aggiungere 6 in testa alla lista:

- inseriamo il valore in posizione (Head-1)%max_size = 4
- incrementiamo di 1 la dimensione della lista
- riassegnamo l'indice della testa

Pile

Pile (Stack)

ADT che usano la politica First In Last Out:

- `push(v)` inserisce il valore `v` in cima alla pila
- `pop()` rimuove il valore in cima alla pila e lo restituisce
- `top()` restituisce il valore in cima alla pila senza rimuoverlo dalla stessa
- `is_empty()` restituisce `true` se e solo se la pila è vuota

Pile (Stack)

Come implementare le code?

Pile (Stack)

Come implementare le pile? Con le liste !?!?!?!

- Liste Concatenate
- Array (non servono nemmeno quelli circolari!?!?!)

`push(v)` = `append(v)` e `pop()` = `last()` + `delete_last()`

Code

Code (Queue)

ADT che usano la politica First In First Out:

- `enqueue(v)` inserisce il valore `v` nella coda
- `dequeue()` rimuove dalla coda il valore che ci sta da più tempo
- `head()` restituisce il valore in testa alla coda senza rimuoverlo dalla stessa
- `is_empty()` restituisce `true` se e solo se la coda è vuota

Code (Queue)

Come implementare le code?

Code (Queue)

Come implementare le code? Con le liste !?!?!?!

- Liste Doppiamente Concatenate
- Array Circolari

`enqueue(v)` $=$ `append(v)` e `dequeue()` $\approx$ `head()` $+$ `tail()`