TECNICO SUPERIORE WEB DEVELOPER FULL STACK

#7 - Strutture dati

Alcune definizioni che dovreste già sapere

Dato

In un linguaggio di programmazione, un dato è un valore che una variabile può assumere

Tipo di dato astratto

Un modello matematico, dato da una collezione di valori e un insieme di operazioni ammesse su questi valori

Tipi di dato primitivi

Forniti direttamente dal linguaggio
Esempi: int (+,-,*,/, %), boolean (!, &&, ||)

Tipi di dati

Definiamo con **specifica** l'insieme di funzionalità di un tipo di dato astratto, che nasconde i dettagli implementativi all'utilizzatore.

Definiamo **implementazione** la realizzazione vera e propria.

Ad esempio, la specifica di una "Coda" è l'insieme dei suoi metodi, mentre l'implementazione potrebbe essere "Coda basata su vettore circolare" o "Coda basata su puntatori".

Strutture di dati

Le strutture di dati sono collezioni di dati, delle quali ci interessa più l'organizzazione dei dati stessi nella collezione piuttosto che il tipo dei dati in essa contenuti. Esse sono caratterizzate da un insieme di operatori che permettono di manipolare la struttura e da un modo sistematico di organizzare l'insieme dei dati.

Alcune tipologie di strutture di dati:

- Lineari / Non lineari (presenza di una sequenza)
- Statiche / Dinamiche (variazione di dimensione, contenuto)
- Omogenee / Disomogenee (dati contenuti)

Quali strutture di dati vedremo?

Tipo	Java	C#	Python
Sequenza	List, Queue, Stack	List, Queue, Stack	list, tuple
Insieme	Set, HashSet	HashSet	set, frozenset
Dizionario	Map, HashMap	Dictionary, SortedDictionary	dict
Albero	*Da definire di volta in volta*		
Grafo	*Da definire di volta in volta*		

Sequenza

Sequenza - Definizione

Una sequenza è una struttura dati **dinamica**, **lineare** che rappresenta una serie **ordinata (nel senso di “prima” e “dopo”)** di valori, dove un valore può comparire più di una volta. L’ordine all’interno della sequenza è importante.

Sequenza - Operazioni ammesse

- Data la posizione, è possibile aggiungere / togliere elementi
- $s = s_1, s_2, \dots, s_n$ l'elemento s_i è in posizione pos_i
- Esistono le posizioni (fittizie) pos_0, pos_{n+1}
- È possibile accedere direttamente alla testa/coda
- È possibile accedere sequenzialmente a tutti gli altri elementi

Sequenza - Specifica

boolean isEmpty()	Restituisce true se la sequenza è vuota
boolean finished(Pos p)	Restituisce true se p è uguale a pos_0 oppure a pos_{n+1}
Pos head()	Restituisce la posizione del primo elemento
Pos tail()	Restituisce la posizione dell'ultimo elemento
Pos next(Pos p)	Restituisce la posizione dell'elemento che segue p
Pos prev(Pos p)	Restituisce la posizione dell'elemento che precede p

Sequenza - Specifica

Pos insert (Pos p, Item v)	Inserisce l'elemento v di tipo Item nella posizione p e restituisce la posizione del nuovo elemento, che diviene il predecessore di p.
Pos remove (Pos p)	Rimuove l'elemento contenuto nella posizione p e restituisce la posizione del successore di p, che diviene successore del predecessore di p
Item read (Pos p)	Legge l'elemento di tipo Item contenuto nella posizione p
write (Pos p, Item v)	Scrive l'elemento v di tipo Item nella posizione p

Insieme

Insieme - Definizione

Un insieme è una struttura dati **dinamica**, **non lineare** che memorizza una collezione **non ordinata** di elementi senza valori ripetuti.

L'ordinamento fra elementi è dato dall'eventuale relazione d'ordine definita sul tipo degli elementi stessi (ad esempio alfabetico, numerico…), non dalla struttura.

Insieme - Operazioni ammesse

Operazioni base:

- Inserimento
- Cancellazione
- Verifica contenimento

Operazioni insiemistiche:

- Unione
- Intersezione
- Differenza

Operazioni di ordinamento:

- Massimo
- Minimo

Iteratori:

$$\textbf{foreach } x \in S \textbf{ do}$$

Insieme - Specifica

int size()	Restituisce la cardinalità dell'insieme
boolean contains (Item x)	Restituisce true se x è contenuto nell'insieme
insert (Item x)	Inserisce x nell'insieme, se non già presente
remove (Item x)	Rimuove x dall'insieme, se presente
Set union (Set A, Set B)	Restituisce un nuovo insieme che è l'unione di A e B
Set intersection (Set A, Set B)	Restituisce un nuovo insieme che è l'intersezione di A e B
Set difference (Set A, Set B)	Restituisce un nuovo insieme che è la differenza di A e B

Dizionario

Dizionario - Definizione

Un dizionario è una struttura dati che rappresenta il concetto matematico di relazione univoca R: D ⟶ C, detta anche **associazione chiave-valore**.

- L'insieme D è detto *dominio* (gli elementi sono detti chiavi)
- L'insieme C è detto *codominio* (gli elementi sono detti *valori*)

Dizionario - Operazioni ammesse

- Ottenere il valore associato ad una particolare chiave (se presente), o il valore **nil** se assente
- Inserire una nuova associazione chiave-valore, cancellando eventuali associazioni precedenti per la stessa chiave
- Rimuovere un’associazione chiave-valore esistente

Dizionario - Specifica

Item lookup (Item k)	Restituisce il valore associato alla chiave k se presente, **nil** altrimenti
insert (Item k, Item v)	Associa il valore v alla chiave k
remove (Item k)	Rimuove l'associazione della chiave k

Definizioni di base

Una “collezione” è una struttura dati che permette di raggruppare elementi in un’unica raccolta.

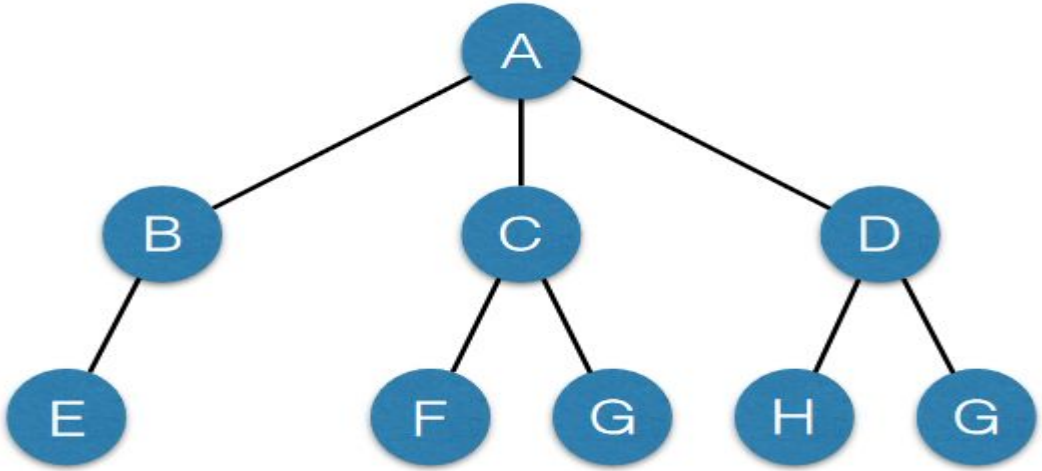
Gli array (mono o multi-dimensionali) sono un particolare tipo di collezione, dove i dati sono tutti dello stesso tipo.

Altri tipi di collezione “notevoli” sono

- Liste
- Dizionari
- Tabelle Hash

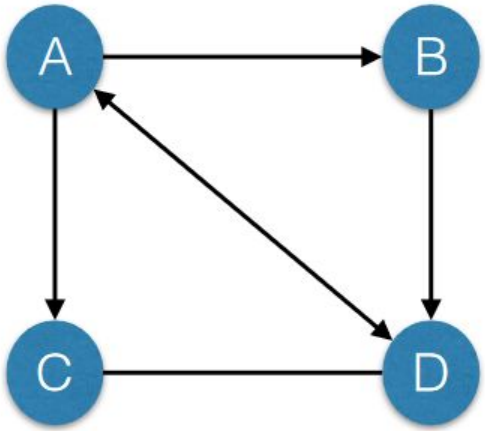
Cenni veloci agli alberi

Un **albero ordinato** è dato da un insieme finito di elementi detti nodi. Uno di questi nodi è designato come **radice**, mentre i rimanenti nodi, se esistono, sono partizionati in insiemi ordinati e disgiunti, anch'essi alberi ordinati.



Cenni veloci ai grafi

La struttura dati grafo è composta da un insieme di elementi detti **nodi** o **vertici** e da un insieme di coppie (ordinate oppure no) di nodi detti **archi**.



Operazioni su alberi e grafi

Tutte le operazioni su alberi e grafi ruotano attorno alla possibilità di effettuare le cosiddette **visite** su di essi.

Vedremo la specifica completa di queste strutture più avanti.

Considerazioni

I concetti di sequenza, insieme, dizionario sono collegati

- Insieme delle chiavi / insieme dei valori
- Scorrere la sequenza di tutte le chiavi

Alcune realizzazioni sono "naturali"

- Sequenza↔lista
- Albero astratto↔albero basato su puntatori

Considerazioni

Esistono tuttavia realizzazioni alternative

- Insieme come vettore booleano
- Albero come vettore dei padri

La scelta della struttura di dati ha riflessi sull'efficienza e sulle operazioni ammesse

- *Dizionario come hash table*:
  lookup $O(1)$, ricerca minimo $O(n)$
- *Dizionario come albero binario di ricerca*:
  lookup e ricerca minimo $O(\log n)$

Esempi di implementazione di strutture dati elementari

Lista (Linked List)

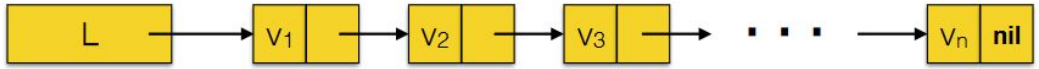
Implementazione del tipo “Sequenza”; è una sequenza di nodi, contenenti dati arbitrari e 1-2 puntatori all’elemento successivo e/o precedente.

- Contiguità nella lista ≠ contiguità nella memoria
- Tutte le operazioni tranne la ricerca (quindi trovare il precedente, successivo, il primo, l’ultimo) hanno costo $O(1)$

Possibili implementazioni

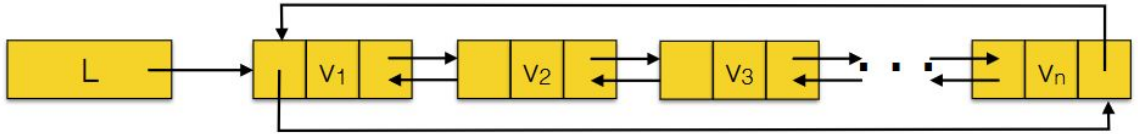
- Bidirezionale / Monodirezionale
- Con sentinella / Senza sentinella
- Circolare / Non circolare

Esempi di implementazione della Lista

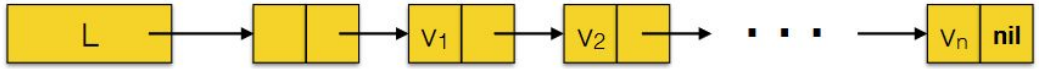


Monodirezionale

Bidirezionale



Bidirezionale circolare



Monodirezionale con sentinella

Implementazione

LIST

```
LIST pred
LIST succ
ITEM value

LIST List()
    LIST t = new LIST
    t.pred = t
    t.succ = t
    return t

boolean isEmpty()
    return pred = succ = this

POS head()
    return succ

POS tail()
    return pred

POS next(POS p)
    return p.succ

POS prev(POS p)
    return p.pred

boolean finished(POS p)
    return (p = this)

ITEM read(POS p)
    return p.value

write(POS p, ITEM v)
    p.value = v

POS insert(POS p, ITEM v)
    LIST t = List()
    t.value = v
    t.pred = p.pred
    p.pred.succ = t
    t.succ = p
    p.pred = t
    return t;

POS p)
    p.pred.succ = p.succ
    p.succ.pred = p.pred
    LIST t = p.succ
    delete p
    return t;
```

Pila (Stack)

Una struttura dati dinamica, lineare in cui l'elemento rimosso dall'operazione di cancellazione è predeterminato:
"quello che per meno tempo è rimasto nell'insieme"
(LIFO - Last-in, First-out)

Pila - Specifica

boolean isEmpty ()	Restituisce true se la pila è vuota
push (Item k)	Inserisce l'elemento k in cima alla pila
Item pop ()	Rimuove l'elemento in cima alla pila e lo restituisce
Item top()	Legge l'elemento in cima alla pila senza rimuoverlo

Pila - Commenti

Possibili utilizzi

- Nei linguaggi con procedure: gestione dei record di attivazione

Possibili implementazioni

- Tramite *liste bidirezionali*: puntatore all'elemento top
- Tramite *vettore dimensione*: limitata, overhead più basso

Pila con vettore, lo pseudocodice

```
STACK
ITEM[] A                          % Elementi
int n                             % Cursore
int m                             % Dim. massima

STACK Stack(int dim)
    STACK t = new STACK
    t.A = new int[1 ... dim]
    t.m = dim
    t.n = 0
    return t

ITEM top()
    precondition: n > 0
    return A[n]

boolean isEmpty()
    return n = 0

ITEM pop()
    precondition: n > 0
    ITEM t = A[n]
    n = n − 1
    return t

push(ITEM v)
    precondition: n < m
    n = n + 1
    A[n] = v
```

Implementazione

Coda (Queue)

Una struttura dati dinamica, lineare in cui l'elemento rimosso dall'operazione di cancellazione è predeterminato:

"quello che per più tempo è rimasto nell'insieme"

(FIFO - First-in, First-out)

Coda - Specifica

boolean isEmpty ()	Restituisce true se la coda è vuota
enqueue (Item k)	Inserisce l'elemento k in fondo alla coda
Item dequeue ()	Rimuove l'elemento in testa alla coda e lo restituisce
Item top()	Legge l'elemento in testa alla coda senza rimuoverlo

Coda - Commenti

Possibili utilizzi

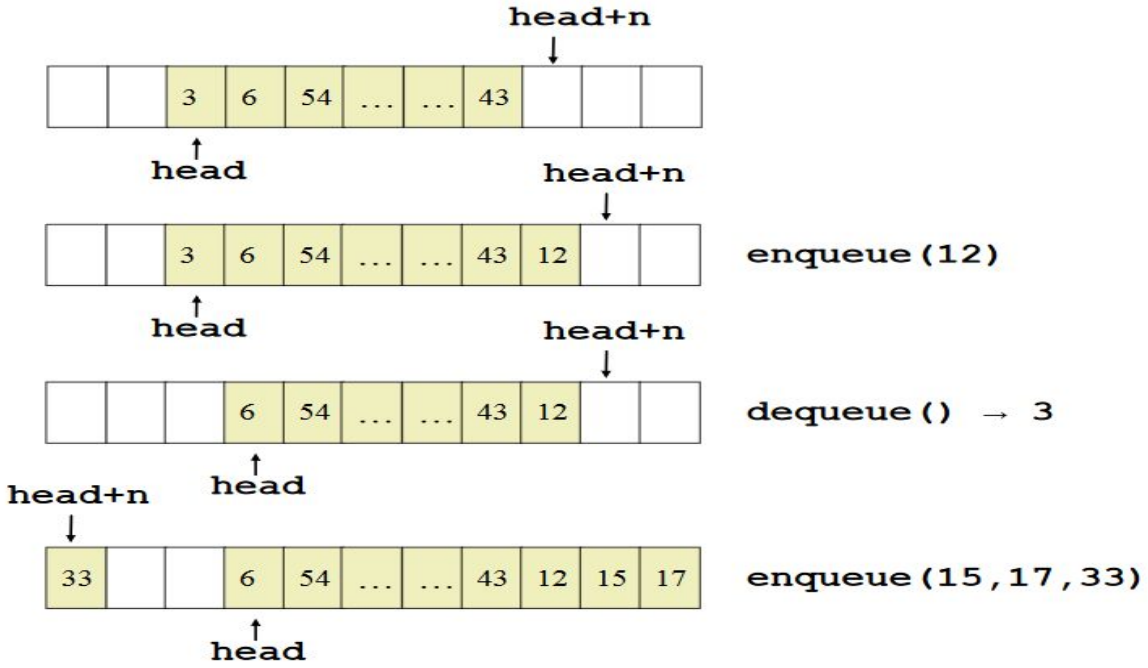
- Nei sistemi operativi, i processi in attesa di utilizzare una risorsa vengono gestiti tramite una coda
- La politica FIFO è equa (fair)

Possibili implementazioni

- Tramite liste monodirezionali: puntatore head, per estrazione puntatore tail, per inserimento
- Tramite array circolari dimensione limitata, overhead più basso

Coda basata su vettore circolare

- La circolarità può essere implementata con l'operazione modulo
- Bisogna prestare attenzione ai problemi di overflow (buffer pieno)



Coda con vettore circolare, lo pseudocodice

QUEUE

```
ITEM[] A                                  % Elementi
int n                                     % Dim. attuale
int testa                                 % Testa
int m                                     % Dim. massima

QUEUE Queue(int dim)
    QUEUE t = new QUEUE
    t.A = new int[0...dim − 1]
    t.m = dim
    t.testa = 0
    t.n = 0
    return t

ITEM top()
    precondition: n > 0
    return A[testa]

boolean isEmpty()
    return n = 0

ITEM dequeue()
    precondition: n > 0
    ITEM t = A[testa]
    testa = (testa + 1) mod m
    n = n − 1
    return t

enqueue(ITEM v)
    precondition: n < m
    A[(testa + n) mod m] = v
    n = n + 1
```

Implementazione

Disclaimer

Quasi tutte queste strutture utilizzano i costrutti “Generics”, ossia la definizione di tipo (la collezione) comprende anche il tipo degli oggetti che essa contiene (il tipo degli elementi della lista).

In generale, quindi, la dichiarazione tipo di una collezione ha questa forma:

```
Collezione<Elementi> coll = new Collezione<Elementi>();
```

Liste

Liste

Come gli array, le liste sono sequenze omogenee, ma:

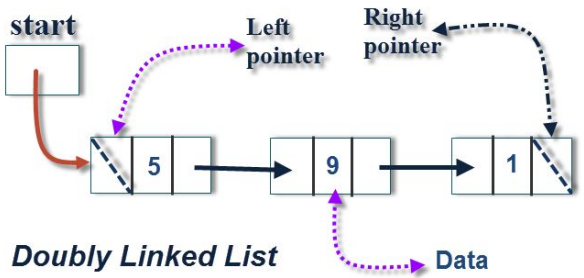
- sono potenzialmente infinite (nel senso che possiamo aggiungere elementi a piacere)
- l'aggiunta di elementi può avvenire in qualunque punto della lista.

Tuttavia, l'implementazione della lista può seguire due possibili strategie.

#1: Array come lista collegata, la classe LinkedList

Nella classe LinkedList gli elementi vengono memorizzati in una struttura con due riferimenti, uno all'elemento precedente e uno all'elemento successivo.

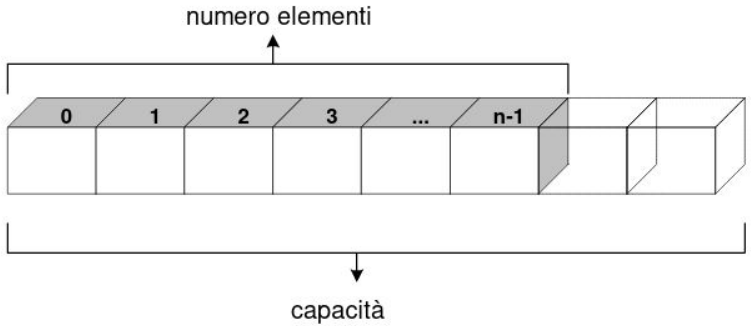
- Anche in questo caso le operazioni di aggiunta e rimozione vengono effettuate automaticamente, ma…
- …è complessivamente una struttura meno efficiente, perché scorre ogni volta tutta la lista



#2: Array come lista, la classe List

La classe List è l'implementazione più semplice: gli elementi vengono memorizzati in un array, ma noi vediamo il tutto come una lista.

- Operazioni che richiedono lo spostamento degli elementi (principalmente aggiunta ed eliminazione *non in coda alla lista*) vengono effettuate automaticamente



Dichiarazione di una lista

Per dichiarare una lista si usa il comando

```
List<tipo> mialista = new List<tipo>();
```

`tipo` può essere un qualunque tipo, sia primitivo…

```
List<int> mialista = new List<int>();
```

…che strutturato. Ad esempio, se abbiamo definito una struct `Punto`:

```
List<Punto> mialista = new List<Punto>();
```

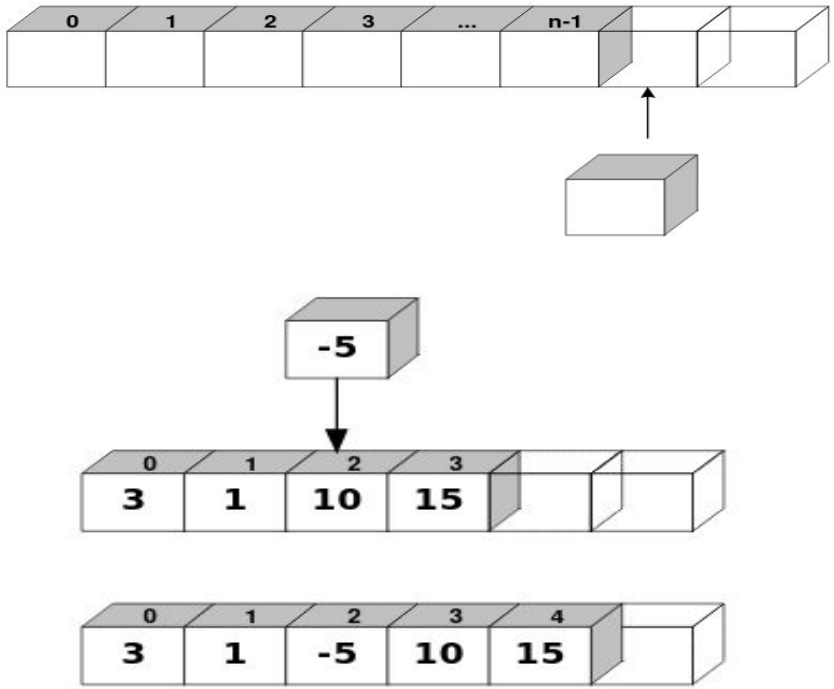
Inserimento di un elemento

Per aggiungere un elemento ad una lista si usano i metodi

- Add(elemento)
- Insert(pos, elemento)

In questo caso:

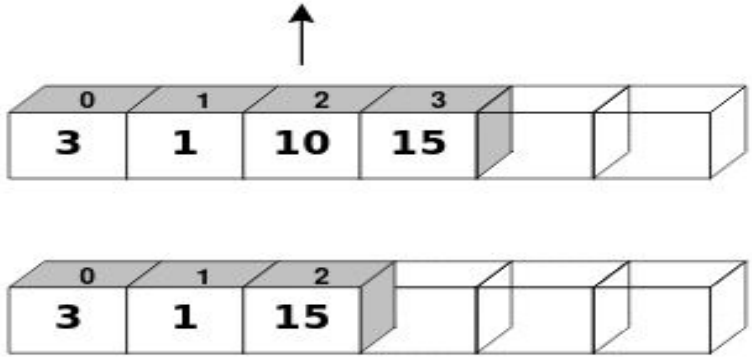
```
mialista.Insert(2, -5);

mialista.Add(15);
```



Rimozione di un elemento

Per rimuovere un elemento da una lista si usano i metodi

- Remove(elemento)
- RemoveAt(pos)



In questo caso:

```
l.Remove(10);

l.RemoveAt(2);
```

Accesso ad un elemento

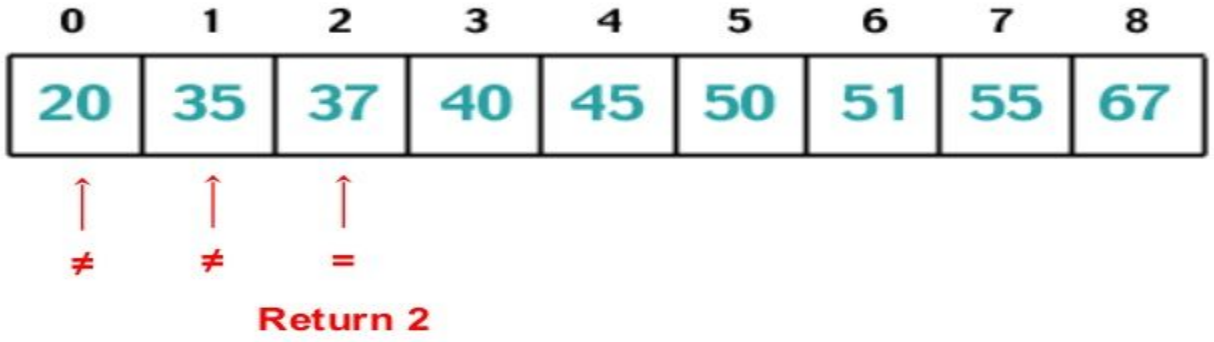
L’accesso ad un elemento si può effettuare come con gli array

```
mialista[posizione] = valore;

variabile = mialista[posizione];
```

Ricerca di un elemento

Poiché nella lista array gli elementi non sono ordinati, la ricerca di un elemento è di tipo lineare, cioè un accesso sequenziale agli elementi partendo dal primo e confronto di ogni elemento con il valore da cercare.

Se, per esempio, cerchiamo il valore 37:



Iterare sulle liste

Le liste sono come gli array, quindi si possono usare sia il for

```
for (int i = 0; i < mialista.Count; i++)
{ ... mialista[i] ... }
```

sia il foreach

```
foreach (var item in mialista)
{ ... item ... }
```

Costo computazionale

- Accesso: $O(1)$
- Inserimento in coda: $O(1)$
- Inserimento in una posizione specifica: $O(n)$
- Rimozione (coda): $O(1)$
- Rimozione in una posizione specifica: $O(n)$
- Ricerca: $O(n)$

Esercizi

- Scrivi una funzione che, dato come parametro una lista di interi, calcoli la percentuale di valori al disotto della media dei numeri contenuti in essa. Scrivi poi un programma che usi questa funzione su una lista di N numeri generati casualmente, con N letto da tastiera.
- Crea una lista di stringhe e popolala attraverso un ciclo che termini all'inserimento della parola 'exit'. A questo punto l'utente può scegliere se:
  - Mostrare la parola più lunga;
  - Mostrare la parola più corta;
  - Mostrare tutte le parole che contengono una certa lettera
  - Mostrare tutte le parole che NON contengono una certa lettera