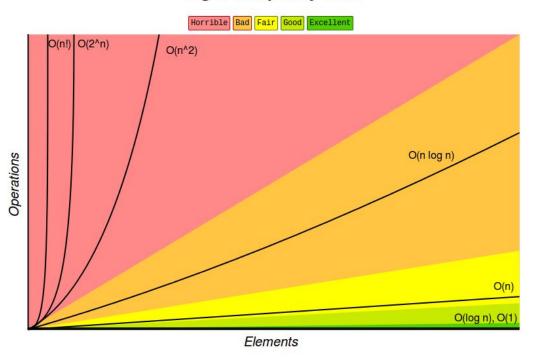
Strutture dati avanzate

Alcune strutture dati di comune utilità

Complessità

Perché è importante valutare la **complessità** di un algoritmo? Perché ci fa capire quanto il nostro software è **scalabile***.

Big-O Complexity Chart



^{*} Scalabilità: In informatica, la caratteristica di un sistema software o hardware facilmente modificabile nel caso di variazioni notevoli della mole o della tipologia dei dati trattati.

II framework .NET

Molti algoritmi e strutture dati sono già implementati nel framework .NET; percui è importante conoscere come lavorano e quali sono le loro caratteristiche in termini di performance e complessità di memoria.

Common Data Structure Operations

Data Structure	Structure Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	0(1)	0(n)	0(n)	0(n)	0(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	0(n)	0(n)	0(1)	0(1)	0(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	0(n)	0(n)	0(1)	0(1)	0(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	0(n)	0(n)	0(1)	0(1)	0(n)
Doubly-Linked List	Θ(n)	Θ(n)	0(1)	Θ(1)	0(n)	0(n)	0(1)	0(1)	0(n)
Skip List	$\theta(\log(n))$	θ(log(n))	$\theta(\log(n))$	$\Theta(\log(n))$	0(n)	0(n)	0(n)	0(n)	0(n log(n))
Hash Table	N/A	θ(1)	Θ(1)	Θ(1)	N/A	0(n)	0(n)	0(n)	0(n)
Binary Search Tree	$\Theta(\log(n))$	Θ(log(n))	θ(log(n))	Θ(log(n))	0(n)	0(n)	0(n)	0(n)	0(n)
Cartesian Tree	N/A	$\theta(\log(n))$	$\theta(\log(n))$	Θ(log(n))	N/A	0(n)	0(n)	0(n)	0(n)
B-Tree	$\Theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\Theta(\log(n))$	0(log(n))	0(log(n))	0(log(n))	0(log(n))	0(n)
Red-Black Tree	$\Theta(\log(n))$	$\theta(\log(n))$	θ(log(n))	θ(log(n))	0(log(n))	0(log(n))	0(log(n))	0(log(n))	0(n)
Splay Tree	N/A	$\theta(\log(n))$	θ(log(n))	Θ(log(n))	N/A	0(log(n))	0(log(n))	0(log(n))	0(n)
AVL Tree	$\Theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\Theta(\log(n))$	0(log(n))	O(log(n))	0(log(n))	0(log(n))	0(n)
KD Tree	$\Theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	θ(log(n))	0(n)	0(n)	0(n)	0(n)	0(n)

Strutture dati in .NET

Nel framework .NET sono già **implementate** molte tipologie di strutture dati.

Possiamo suddividere le strutture dati in semplici e generiche.

Inizieremo a vedere le strutture dati **semplici**, cioè quelle strutture dati **non tipizzate** e che quindi possono ospitare qualsiasi tipo di dato.

Vedremo poi anche le strutture dati **generiche**, cioè quelle strutture dati tipizzate alla loro dichiarazione e quindi che possono memorizzare solo il tipo di dato scelto.

Le strutture dati semplici sono raggruppate insieme nel namespace System.Collections

Le strutture dati generiche sono raggruppate nel namespace System.Collections.Generic

^{*}Namespace: ambito che contiene un set di oggetti correlati così da organizzare il codice ed evitare collisione causate da elementi con lo stesso nome.

ArrayList (l'evoluzione dell'array)

- ricorda la lista della spesa
- gli elementi inseriti nella lista mantengono l'ordine con cui sono stati aggiunti
- la lista si estende automaticamente a seconda di quanti elementi vengono inseriti; a differenza dell'array che andava preventivamente dimensionato alla dichiarazione



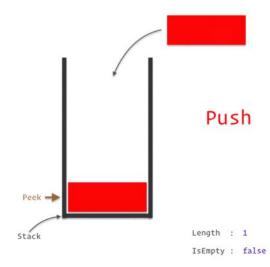
Esempio ArrayList

```
static void Main(string[] args)
                                                                   Defining an array list
                         ArrayList a1 = new ArrayList();
Adding elements
                        a1.Add(1);
                         a1.Add("Example");
to the array list
                         a1.Add(true);
                        Console.WriteLine(a1[0]);
                                                                Displaying the elements
                         Console.WriteLine(a1[1]);
                                                                of the array list
                         Console.WriteLine(a1[2]);
                        Console.ReadKey();
```

Stack (pila)

- ricorda una scatola o una pila di vestiti
- LIFO: Last In First Out
- si auto estende al bisogno





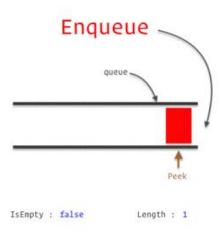
Esempio Stack

```
static void Main(string[] args)
                                    creating a stack variable
   Stack st = new Stack();
                                                                count of stack
                                 Pushing elements
    st.Push(1);
    st.Push(2);
                                 to the stack
                                                                elements
    st.Push(3);
    foreach (Object obj in st){
                                          Displaying stack
                                                                  searching for
        Console.WriteLine(obj);
                                          elements
                                                                  an element
    Console.WriteLine(); Console.WriteLine();
   Console.WriteLine("The number of elements in the stack=" + st.Count);
    Console.WriteLine("Does the stack contain the element 3=" + st.Contains(3));
```

Queue (coda)

- ricorda la coda alla cassa del supermercato
- FIFO: First In First Out
- si auto estende al bisogno





Esempio Queue

```
static void Main(string[] args)
                                       creating a queve variable
    Queue qt = new Queue();
                                  adding elements
    qt.Enqueue(1);
    qt.Enqueue(2);
                                                             count of queve
                                  to the queve
    qt.Enqueue(3);
                                                             elements
    foreach (Object obj in qt){
                                           Displaying queve
        Console.WriteLine(obj);
                                           elements
                                                                    searching for
                                                                   an element
    Console.WriteLine(); Console.WriteLine();
    Console.WriteLine("The number of elements in the queue" + qt.Count);
    Console.WriteLine("Does the queue contain" + qt.Contains(3));
```

Linked List (lista collegata)

- ricorda un treno con i suoi vagoni
- ogni nodo è collegato al successivo
- la **testa** non ha un nodo precedente
- la coda non ha un nodo successivo
- si auto estende al bisogno
- è una struttura dati generica
- .NET namespace:System.Collections.Generic

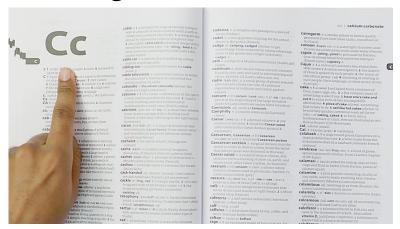


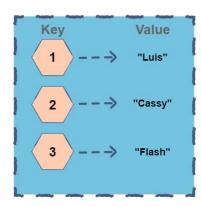
Esempio LinkedList

```
LinkedList<int> lk = new LinkedList<int>();
lk.AddFirst(1);
var node2 = new LinkedListNode<int>(2);
lk.AddLast(node2);
lk.AddLast(3);
lk.AddAfter(node2, 5);
foreach(int n in lk){
        Console.WriteLine(n);
}
```

Hashtable

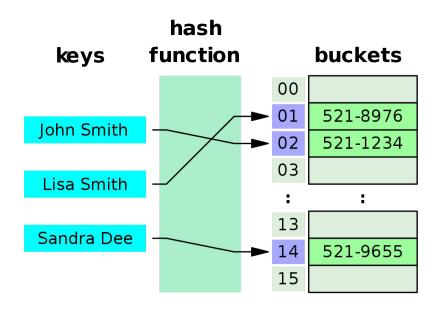
- ricorda un dizionario
- utilizza la tecnica di hash per memorizzare i dati
- questa tecnica rende la ricerca molto veloce: O(1)
- si auto estende al bisogno
- l'ordine degli elementi non è mantenuto



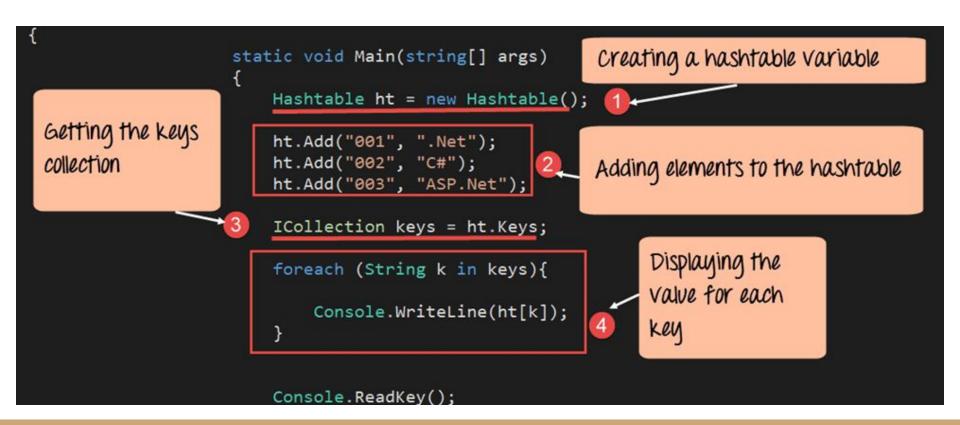


Tecnica di hashing

Tecnica secondo la quale una chiave viene convertita, per mezzo di una funzione matematica di hash, in un valore numerico univoco.



Esempio Hashtable



Algoritmi in .NET

- Ricerca binaria in array: metodo Array.BinarySearch()
- Ricerca sequenziale in Stack, Queue, LinkedList: metodo Contains()
- Ricerca sequenziale in Hashtable: metodo ContainsKey() e
 ContainsValue()
- Ricerca per chiave in Dictionary: TryGetValue() o accesso per chiave.