# COLLEZIONI DI DATI

Tutti i linguaggi offrono di base 3 tipi:

* Liste ordinate
* Insiemi di elementi univoci
* Mappe chiave-valore

Ciascuna di queste hanno possibilità di essere implemenate in vario modo:

* Es: Liste ordinate
  + Vec → contigui
  + LinkedList → sparpagliati con puntatore a next e previous

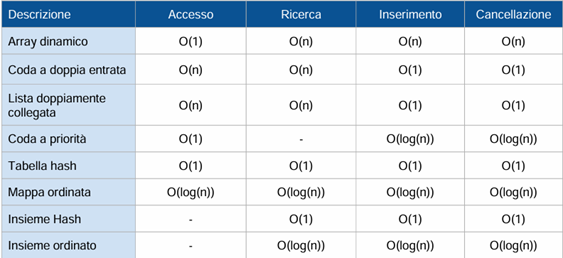
### **LINEARI:**

ArrayDinamico **Vec<T>:** contiene elementi contigui → sempre contigui

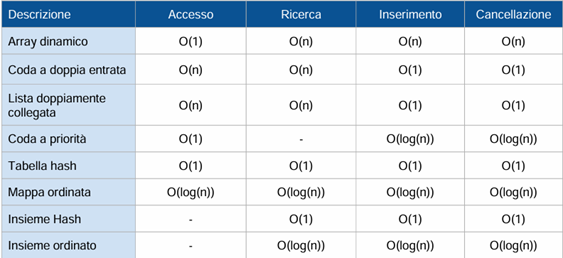
* Nr di elementi cambia nel tempo, posso allargare o contrarre
* Se fatto con costo ammortizzato → se non ho spazio, raddoppio dimensione
* Molto costoso aggiungere in testa perché devo spostare tutti

Coda a doppia entrata **VecDeque<T>** :

* Permettono di inserire in testa o in coda senza costo eccessivo

Lista doppiamente collegata **LinkedList<T>:**

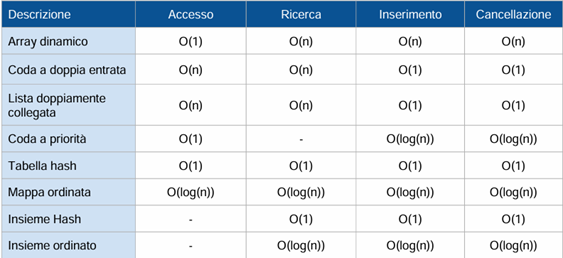
* Ogni elemento ha puntatore a precedente e successivo
* Contiene un puntatore a primo e ultimo
* Facile inserimento ovunque
* Camminare è un problema perché se voglio arrivare ad i-esimo, devo assarli tutti

Coda a priorità **BinaryHeap<T>**:

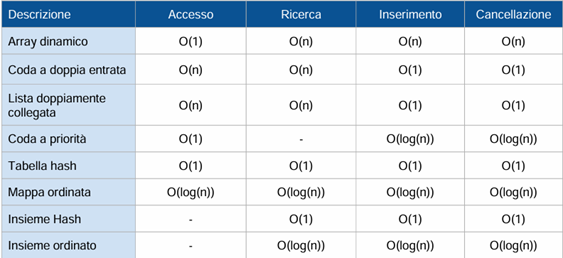
* Struttura a cui posso aggiungere un elemento
* Si avvicina all’uscita in base alla priorità

**MAPPE**: collezioni di chiave univoce a cui è associato un valore

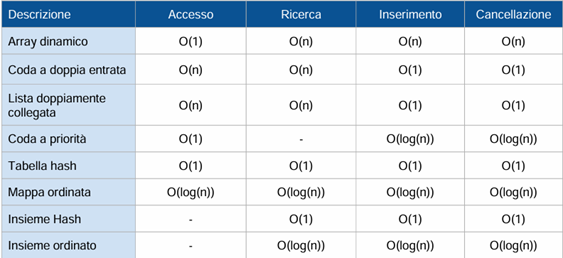
Le chiavi devono essere: immutabili, confrontabili e, a seconda di come viene implementata, devono essere ordinabili o hashabili → trasformabile in un numero tale da garantirmi che valori diversi abbiano chiave diversa.

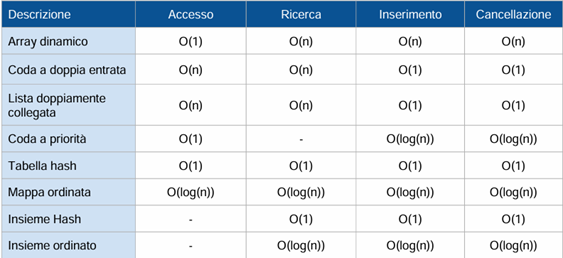
Tabella hash **HashMap<K,V>**:

* Associa a ciascuna chiave un valore

Mappa Ordinata **BtreeMap<K,V>**:

* Ciò che sta a dx ha sempre chiave più grande di x
* Ciò che sta a sx ha sempre chiave più piccolo di x
* Inserimenti molto complicati perché portano a ristrutturazione di albero
  + Albero prova ad essere bilanciato
* Tempo di accesso garantito di log2(nrChiavi)

Insieme Hash **HashSet<T>:**

Insieme ordinato **BtreeSet<T>**:

Tutte le collezioni hanno **dei metodi in comune:**

Immagine che contiene testo, Carattere, schermata, algebra

Descrizione generata automaticamente

Tutte le collezioni **hanno i tratti:**

## TIPO Vec<T>

Rappresenta una sequenza ridimensionabile di elementi di tipo T, allocati sullo heap

Internamente presenta **3 valori privati**:

* **Puntatatore** ad heap → nullo se ancora vuoto
* **Capacity** → capacità totale
* **Size** →nr elementi in uso

**Metodi**:

* **Push** → inserire al fondo
  + Se siamo oltre capacity → parte un processo di riallocazione che alloca il doppio dello spazio attuale
* Esistono metodi per inserire in una determinata posizione → se non esiste, panica.

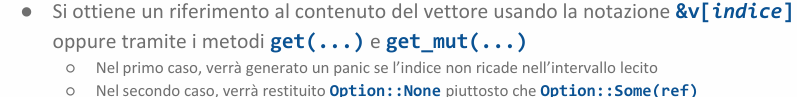
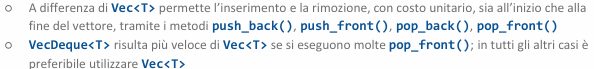


Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamenteImmagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

VecDeque<T>: Modella una coda a doppia entrata: esso alloca sullo heap una serie di elementi di tipo T

* Rust tratta vecdeque come un buffer circolare…
* È un vec che inizia da posizione qualunque, se non può più inserire, rialloca
* Non garantisce che siano contigui, per rendere contiguo in memoria, si usa il metodo **make\_contiguous()**

LinkedList<T>: Lista doppiamente collegata con tempo di accesso costante

* permette di inserire e rimuovere elementi da entrambe le estremità della lista

Va bene se dobbiamo mantenrere un ordine e inserire a metà.i metodi che offre sono scarsi

## **MAPPE**:

Collezione di coppie composte da una **chiave di tipo K e da un valore di tipo V**

Nota: chiave deve essere univoca, tipo K deve implementare i tratti Eq e Hash

### HashMap<K,V>

* + i valori sono salvati sullo heap come singola hash table

### BtreeMap<K,V>

* + i valori sono salvati nello heap come un singolo alber dove ogni entry è un nodo
  + tipo K deve implementare anche Ord

*Capire dove mettere la chiave se ancora non la ho, è una cosa complicata : metodo entry*

**Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamenteMetodo entry**: restituisce un oggetto Entry:

* Nel caso in cui lo ho → ti faccio modificare
* Nel caso in cui non lo ho → ti faccio inserire

Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamente

## **Insiemi**

HashSet<T>: insieme di elementi univoci di tipo T

* Valori salvati su heap come singola hash table (implementato come wrapper in hashmap)

BtreeSet<T>: insieme di elementi univoci di tipo T

* Valori salvati nello heap come un singolo albero (implementato come wrapper in BtreeMap)

BinaryHeap: collezione di elementi di tipo T che ammette solo oggetti ordinabili e li tiene dal più grande al più piccolo.

* T deve implementare Ord