

# Algoritmi e Strutture Dati

## Strutture di dati

Alberto Montresor and Davide Rossi

Università di Bologna

24 ottobre 2024

This work is licensed under a Creative Commons  
Attribution-ShareAlike 4.0 International License.



# Sommario

## 1 Strutture dati astratte

- Definizioni
- Sequenza
- Insiemi
- Dizionari
- Alberi e grafi

## 2 Implementazione strutture dati elementari

- Lista
- Pila
- Coda

# Introduzione

## 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

## “Specifiche” e “implementazione” di un tipo di dato astratto

- **Specifiche:** “manuale d’uso”, nasconde i dettagli implementativi all’utilizzatore
- **Implementazione:** realizzazione vera e propria

## Esempi

Specifiche	Implementazione
Numeri reali	IEEE-754
Pile	Pile basate su vettori Pile basate su puntatori
Code	Code basate su vettori circolari Code basate su puntatori

# Strutture di dati

## Strutture di dati

Le strutture di dati sono collezioni di dati, caratterizzate più dall'organizzazione della collezione piuttosto che dal tipo dei dati contenuti.

## Come caratterizzare le strutture dati

- un insieme di operatori che permettono di manipolare la struttura
- 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)

## Tipologie di strutture dati

**Lineari** → I dati sono organizzati in una sequenza ordinata dove ogni elemento ha un predecessore e un successore (es. Array)

**Non-Lineari** → I dati non seguono una sequenza lineare, ma hanno relazioni gerarchiche e ricetticolate (es. Albero)

**Statiche** → La dimensione è fissa al momento della creazione e non può cambiare (es. Array) (vantaggio: accesso rapido a elementi (indici))  
(svantaggio: spreco di memoria se sottoutilizzato)

**Dinamiche** → La dimensione può variare durante l'esecuzione (es. liste concatenate)  
(vantaggi: Flessibilità, ottimizzazione della memoria)  
(svantaggi: Overhead di memoria per puntatori)

**Omoogenee** → Tutti gli elementi sono dello stesso tipo (es. Array (solo interi))  
(vantaggi: semplificazione di gestione d'accesso)

**Disomogenee** → Gli elementi possono essere di tipi diversi (es. Classi (Java/Py) → campi contip: eterogenei)  
(vantaggi: Modellazione di oggetti complessi) (es. Tuple (Py) → possono contenere tipi misti)

• Dinamica non vuol dire ArrayList (dimensione ∞). Si considera un m limite dell'array. Ma non viene considerato quando viene creata la lista.  
↳ Non è un parametro

# Strutture di dati

Tipo	Java	C++	Python
Sequenze	List, Queue, Deque LinkedList, ArrayList, Stack, ArrayDeque	list, forward_list vector stack queue, deque	list tuple
Insiemi	Set TreeSet, HashSet, LinkedHashSet	set unordered_set	set, frozenset
Dizionari	Map HashTree, HashMap, LinkedHashMap	map unordered_map	dict
Alberi	-	-	-
Grafi	-	-	-

# Sequenza

## Sequenza

Una struttura dati **dinamica, lineare** che rappresenta una sequenza di valori **ordinati posizionalmente**, che ammette occorrenze ripetute di uno stesso valore.

- L'ordine all'interno della sequenza è importante

## 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

---

## SEQUENCE

---

% Restituisce **true** se la sequenza è vuota

**boolean isEmpty()**

% Restituisce **true** se  $p$  è uguale a  $pos_0$  oppure a  $pos_{n+1}$

**boolean finished(Pos p)**

% Restituisce la posizione del primo elemento

**Pos head()**

% Restituisce la posizione dell'ultimo elemento

**Pos tail()**

% Restituisce la posizione dell'elemento che segue  $p$

**Pos next(Pos p)**

% Restituisce la posizione dell'elemento che precede  $p$

**Pos prev(Pos p)**

---

# Sequenza – Specifica

---

## SEQUENCE (continua)

---

% Inserisce l'elemento  $v$  di tipo ITEM nella posizione  $p$ .

% Restituisce la posizione del nuovo elemento che diviene il predecessore di  $p$

**Pos insert(Pos  $p$ , ITEM  $v$ )**

% Rimuove l'elemento contenuto nella posizione  $p$ .

% Restituisce la posizione del successore di  $p$ ,

% che diviene successore del predecessore di  $p$

**Pos remove(Pos  $p$ )**

% Legge l'elemento di tipo ITEM contenuto nella posizione  $p$

**ITEM read(Pos  $p$ )**

% Scrive l'elemento  $v$  di tipo ITEM nella posizione  $p$

**write(Pos  $p$ , ITEM  $v$ )**

---

# Sequenza

## Java

```
List<String> lista = new LinkedList<String>();  
lista.add("two");  
lista.addFirst("one");  
lista.addLast("three");
```

Result: [ 'one', 'two', 'three' ]

## C++

```
std::list<int> lista;  
lista.push_front(2);  
lista.push_front(1);  
lista.push_back(3);
```

Result: [1,2,3]

## Python

```
lista = ["one", "three"]  
lista.insert(1, "two")
```

Result: [ 'one', 'two', 'three' ]

# Insiemi

## Insieme

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

## Operazioni ammesse

- Operazioni base:
  - inserimento
  - cancellazione
  - verifica appartenenza
- Operazioni di ordinamento
  - Massimo
  - Minimo

- Operazioni insiemistiche:
  - unione
  - intersezione
  - differenza
- Iteratori:
  - **foreach**  $x \in S$  do

restituisce un  
nuovo insieme  
dalla differenza  
di due insiemi

# Insiemi – Specifica

---

## SET

---

% Restituisce la cardinalità dell'insieme

**int size()**

% Restituisce **true** se  $x$  è contenuto nell'insieme

**boolean contains(ITEM  $x$ )**

% Inserisce  $x$  nell'insieme, se non già presente

**insert(ITEM  $x$ )**

% Rimuove  $x$  dall'insieme, se presente

**remove(ITEM  $x$ )**

% Restituisce un nuovo insieme che è l'unione di  $A$  e  $B$

**SET union(SET  $A$ , SET  $B$ )**

% Restituisce un nuovo insieme che è l'intersezione di  $A$  e  $B$

**SET intersection(SET  $A$ , SET  $B$ )**

% Restituisce un nuovo insieme che è la differenza di  $A$  e  $B$

**SET difference(SET  $A$ , SET  $B$ )**

---

# Insiemi

Java

```
Set<String> items = new TreeSet<>();  
docenti.add("rock");  
docenti.add("paper");  
docenti.add("scissors");
```

TreeSet

Result: { "scissors", "paper", "rock" }

C++

```
std::set<std::string> frutta;  
frutta.insert("mele");  
frutta.insert("pere");  
frutta.insert("banane");  
frutta.insert("mele");  
frutta.remove("mele")
```

Result: { "banane", "pere" }

Python

```
items = { "rock", "paper",  
         "scissors", "rock" }  
print(items)  
print("Spock" in items)  
print("lizard" not in items)  
  
Result:  
{ "rock", "paper", "scissors" }  
False  
True
```

# Dizionari

## Dizionario

*key  $\rightarrow$  value*

Struttura dati che rappresenta il concetto matematico di **relazione univoca**  $R : D \rightarrow C$ , o associazione chiave-valore.

- Insieme  $D$  è il **dominio** (elementi detti **chiavi**)
- Insieme  $C$  è il **codominio** (elementi detti **valori**)

## Operazioni ammesse

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

# Dizionari – Specifica

---

## DICTIONARY

---

% Restituisce il valore associato alla chiave  $k$  se presente, **nil**  
altrimenti

**ITEM** **lookup(ITEM**  $k$ )

% Associa il valore  $v$  alla chiave  $k$

**insert(ITEM**  $k$ , **ITEM**  $v$ )

% Rimuove l'associazione della chiave  $k$

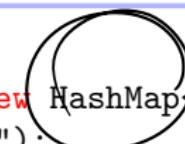
**remove(ITEM**  $k$ )

---

# Array associativi, mappe e dizionari

## Java

```
Map<String, String> capoluoghi = new HashMap<>();  
capoluoghi.put("Toscana", "Firenze");  
capoluoghi.put("Lombardia", "Milano");  
capoluoghi.put("Sardegna", "Cagliari");
```



## C++

```
std::map<std::string, int> wordcounts;  
std::string s;  
  
while (std::cin >> s && s != "end")  
    ++wordcounts[s];
```

## Python

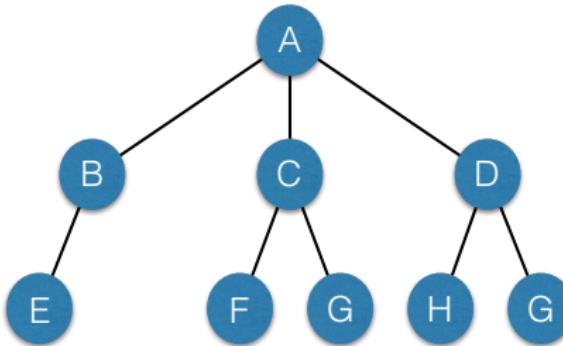
```
v = {}  
v[10] = 5  
v["alberto"] = 42  
v[10]+v["alberto"]
```

Result: 47

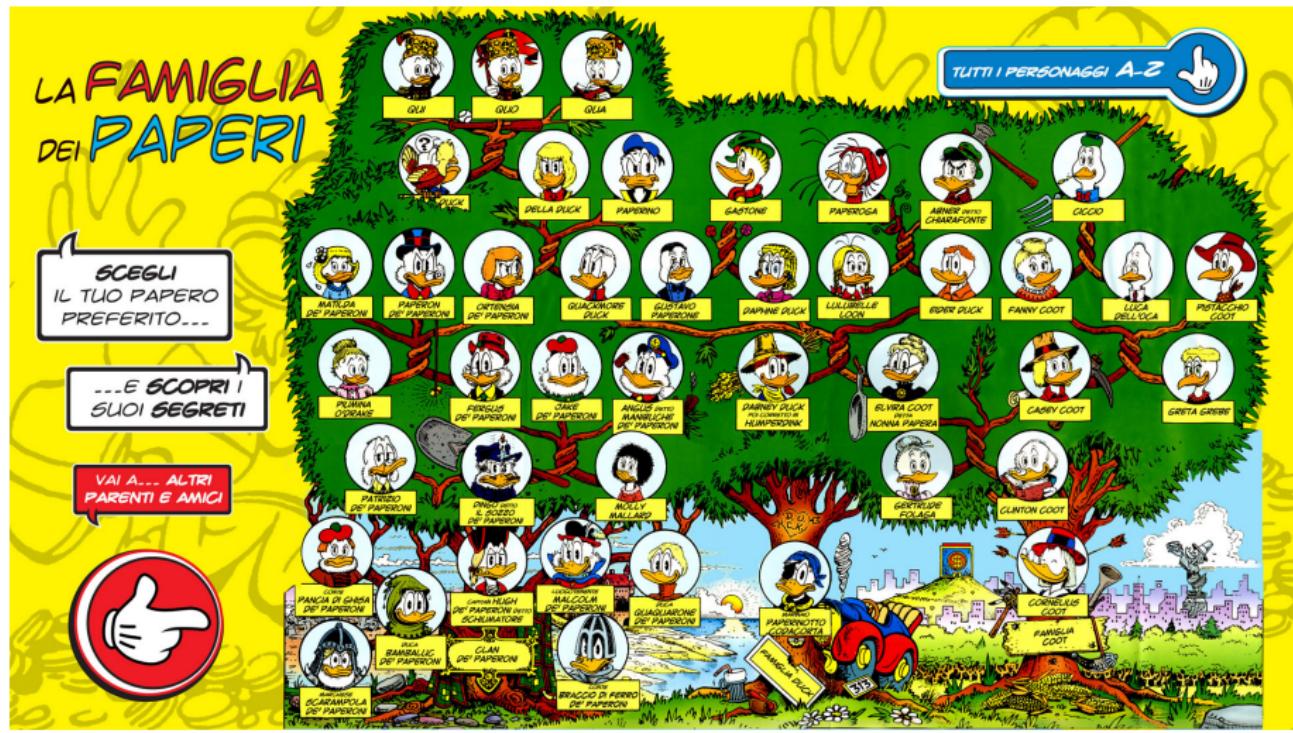
# Alberi e grafi

## Alberi ordinati

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



## Alberi e grafi



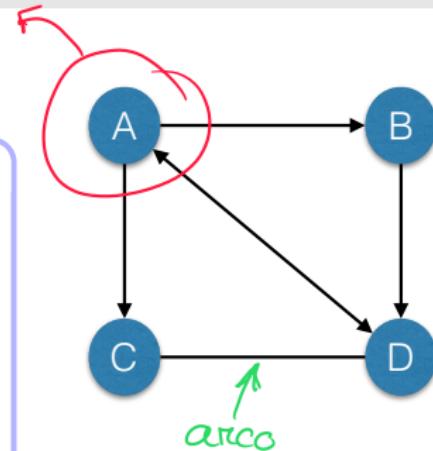
# Alberi e grafi

## Grafi

La struttura dati grafo è composta da:

- un insieme di elementi detti nodi o vertici
- un insieme di coppie (ordinate oppure no) di nodi detti archi

node/vertice



## Operazioni

- Tutte le operazioni su alberi e grafi ruotano attorno alla possibilità di effettuare visite su di essi
- Specifica completa più avanti

# Commenti

- 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  $\leftrightarrow$  lista
  - Albero astratto  $\leftrightarrow$  albero basato su puntatori
- 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: lookup  $O(\log n)$ , ricerca minimo  $O(1)$

# Sommario

## 1 Strutture dati astratte

- Definizioni
- Sequenza
- Insiemi
- Dizionari
- Alberi e grafi

## 2 Implementazione strutture dati elementari

- Lista
- Pila
- Coda

# Lista



## Lista (Linked List)



Una sequenza di nodi, contenenti dati arbitrari e 1-2 puntatori all'elemento successivo e/o precedente.

### Note

### Sequenza

- Contiguità nella lista  $\not\Rightarrow$  contiguità nella memoria
- Tutte le operazioni hanno costo  $O(1)$

### Possibili implementazioni

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

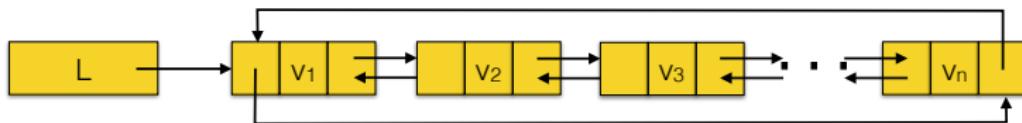
# Liste



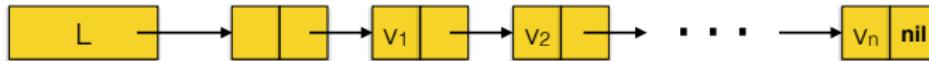
Monodirezionale



Bidirezionale



Bidirezionale circolare



Monodirezionale con sentinella

# Lista bidirezionale con sentinella

LIST

LIST pred

% Predecessore

LIST succ

% Successore

ITEM value

% Elemento

LIST List()

boolean finished(Pos p)

└ return (p = this)

ITEM read(Pos p)

└ return p.value

→ Legge l'elemento con indice  
p

write(Pos p, ITEM v)

└ p.value = v

→ Scrive il valore v  
dell'elemento con indice p

POS insert(Pos p, ITEM v)

ELEMENTO (Posizione, Valore)

└ LIST t = List()

↳ Puntatore

t.value = v

→ viene impostato il predecessore  
di t

t.pred = p.pred

→ viene impostato t come  
successore del predecessore

p.pred.succ = t

t.succ = p

p.pred = t

return t;

boolean isEmpty()

ne pred e succ puntano  
alla stessa sentinella  
→ isEmpty → true

POS head()

→ restituisce la  
posizione del primo elemento  
↳ LA LISTA  
E' VUOTA

POS tail()

→ restituisce la posizione del  
ultimo elemento

POS next(Pos p)

ritorna l'indice del successivo  
elemento con indice p

↳ POS vecchio(Pos p)

└ return p.succ

p.pred.succ = p.succ

→ ritrovare l'elemento a  
indice p

p.succ.pred = p.pred

LIST t = p.succ

reimposta i puntatori  
e Sovrascrive  
la lista vecchia

delete p

return t;

→ ritorna la lista  
modificataP non ha  
puntatori

POS prev(Pos p)

ritorna l'indice del precedente  
elemento con indice p

LIST e Pos sono tipi equivalenti

Utilizzo sentinella → La sentinella cosi' speciali: (if(head == null))  
gli inserimenti in testa/coda diventano operazioni unigocci

# Lista bidirezionale senza sentinella – Java

```
class Pos {  
  
    Pos succ;          /** Next element of the list */  
    Pos pred;          /** Previous element of the list */  
    Object v;           /** Value */  
  
    Pos(Object v) {  
        succ = pred = null;  
        this.v = v;  
    }  
}
```

|      C o s t r u t t o r e

# Lista bidirezionale senza sentinella – Java

```
public class List {  
  
    private Pos head;      /** First element of the list */  
    private Pos tail;      /** Last element of the list */  
  
    public List() {  
        head = tail = null;  
    }  
  
    public Pos head()          { return head; }  
    public Pos tail()          { return tail; }  
    public boolean finished(Pos pos) { return pos == null; }  
    public boolean isEmpty()    { return head == null; }  
    public Object read(Pos p)   { return p.v; }  
    public void write(Pos p, Object v) { p.v = v; }  
}
```

# Lista bidirezionale senza sentinella – Java

```
public Pos next(Pos pos) {  
    return (pos != null ? pos.succ : null);  
}  
  
public Pos prev(Pos pos) {  
    return (pos != null ? pos.pred : null);  
}  
  
public void remove(Pos pos) {  
    if (pos.pred == null)  
        head = pos.succ;  
    else  
        pos.pred.succ = pos.succ;  
    if (pos.succ == null)  
        tail = pos.pred;  
    else  
        pos.succ.pred = pos.pred;  
}
```

# Lista bidirezionale senza sentinella – Java

```
public Pos insert(Pos pos, Object v) {  
    Pos t = new Pos(v);  
    if (head == null) {  
        head = tail = t; // Insert in a emtpy list  
    } else if (pos == null) {  
        t.pred = tail; // Insert at the end (se manca la posizione)  
        tail.succ = t;  
        tail = t;  
    } else {  
        t.pred = pos.pred; // Insert in front of an existing position  
        if (t.pred != null)  
            t.pred.succ = t;  
        else  
            head = t;  
        t.succ = pos;  
        pos.pred = t;  
    }  
    return t;  
}
```

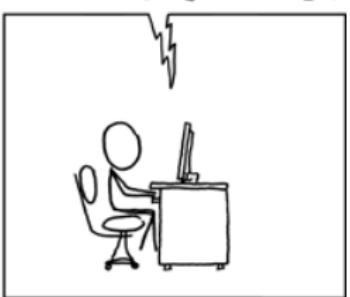
# Liste

```
prev->next = toDelete->next;  
delete toDelete;
```

```
// if only forgetting were  
// this easy for me.
```



```
assert "It's going to be okay.:";
```



# Pila

## 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)

---

### STACK

---

% Restituisce true se la pila è vuota

**boolean isEmpty()**

% Inserisce v in cima alla pila

**push(ITEM v)**

% Estraie l'elemento in cima alla pila e lo restituisce al chiamante

**ITEM pop()**

% Legge l'elemento in cima alla pila

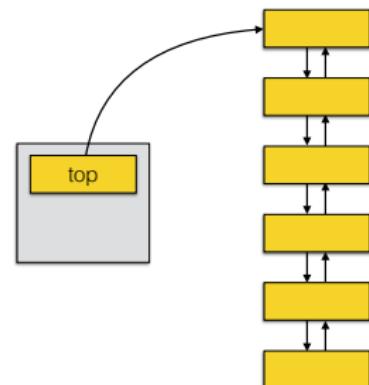
**ITEM top()**

---

# Pila

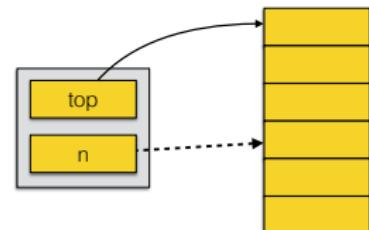
## Possibili utilizzi

- Nei linguaggi con procedure:
  - gestione dei record di attivazione
- Nei linguaggi stack-oriented:
  - le operazioni prendono gli operandi dallo stack e inseriscono il risultato nello stack
  - Es: Postscript, Java bytecode



## Possibili implementazioni

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



# Pila basata su vettore – Pseudocodice

*dinamica*

---

<b>STACK</b>		
ITEM[] <i>A</i>	<b>% Elementi</b> <b>% Cursore</b> <b>% Dim. massima</b>	<b>boolean isEmpty()</b>
int <i>n</i>		<b>return</b> <i>n</i> = 0
int <i>m</i>		
STACK Stack(int <i>dim</i> )		<b>ITEM pop()</b>
STACK <i>t</i> = new STACK		<b>precondition:</b> <i>n</i> > 0
<i>t.A</i> = new int[1... <i>dim</i> ]		ITEM <i>t</i> = <i>A</i> [ <i>n</i> ] <i>n</i> = <i>n</i> - 1 <b>return</b> <i>t</i>
<i>t.m</i> = <i>dim</i>		
<i>t.n</i> = 0		
<b>return</b> <i>t</i>		
ITEM top()		<b>push(ITEM <i>v</i>)</b>
<b>precondition:</b> <i>n</i> > 0		<b>precondition:</b> <i>n</i> < <i>m</i>
<b>return</b> <i>A</i> [ <i>n</i> ]		<i>n</i> = <i>n</i> + 1 <i>A</i> [ <i>n</i> ] = <i>v</i>

---

# Pila basata su vettore – Java

```
public class VectorStack implements Stack {  
  
    /** Vector containing the elements */  
    private Object[] A;  
  
    /** Number of elements in the stack */  
    private int n;  
  
    public VectorStack(int dim) {  
        n = 0;  
        A = new Object[dim];  
    }  
  
    public boolean isEmpty() {  
        return n==0;  
    }  
}
```

Costruttore

## Pila basata su vettore – Java

```
public Object top() {  
    if (n == 0)  
        throw new IllegalStateException("Stack is empty");  
    return A[n-1];  
} return A[n-1];
```

```
public Object pop() {  
    if (n == 0)  
        throw new IllegalStateException("Stack is empty");  
    return A[--n];  
} return A[--n];
```

```
public void push(Object o) {  
    if (n == A.length)  
        throw new IllegalStateException("Stack is full");  
    A[n++] = o;  
} A[n++] = o;  
}
```

*Se tocca il limite della Pila*

# Coda



## 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**)

---

### QUEUE

---

% Restituisce **true** se la coda è vuota

**boolean isEmpty()**

% Inserisce *v* in fondo alla coda

**enqueue(ITEM *v*)**

% Estraie l'elemento in testa alla coda e lo restituisce al chiamante

**ITEM dequeue()**

% Legge l'elemento in testa alla coda

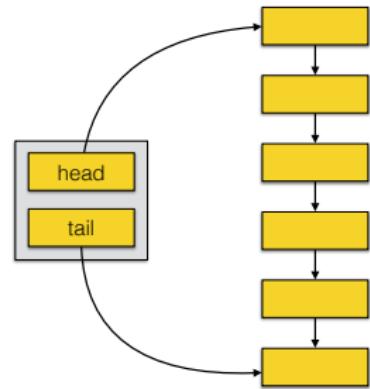
**ITEM top()**

---

# Coda

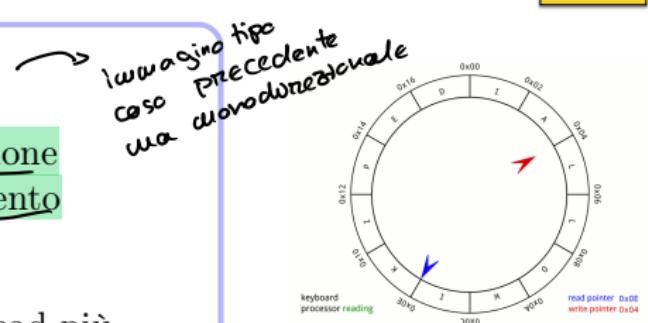
## Possibili utilizzi

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



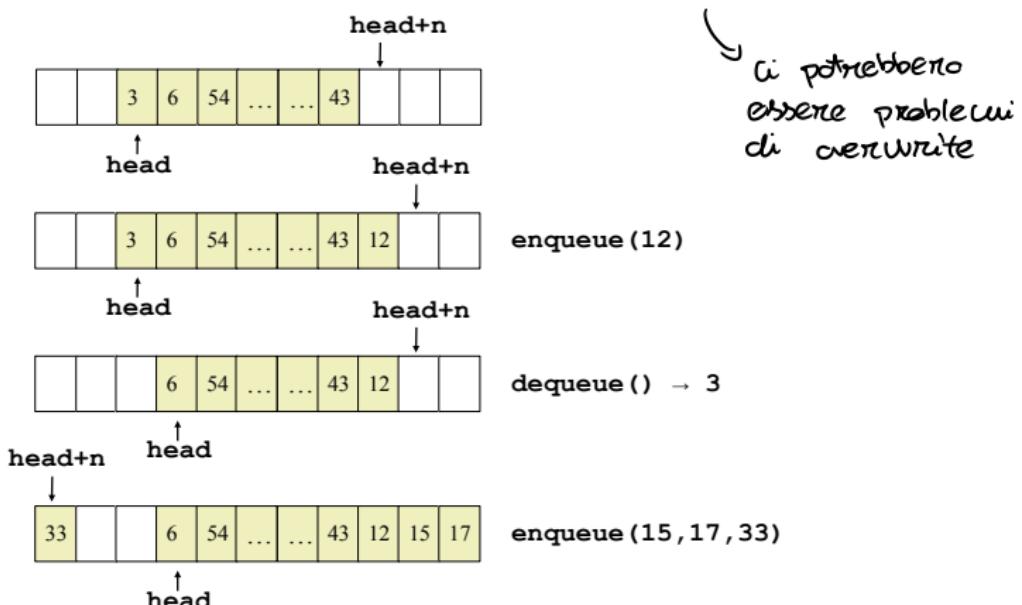
## Possibili implementazioni

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

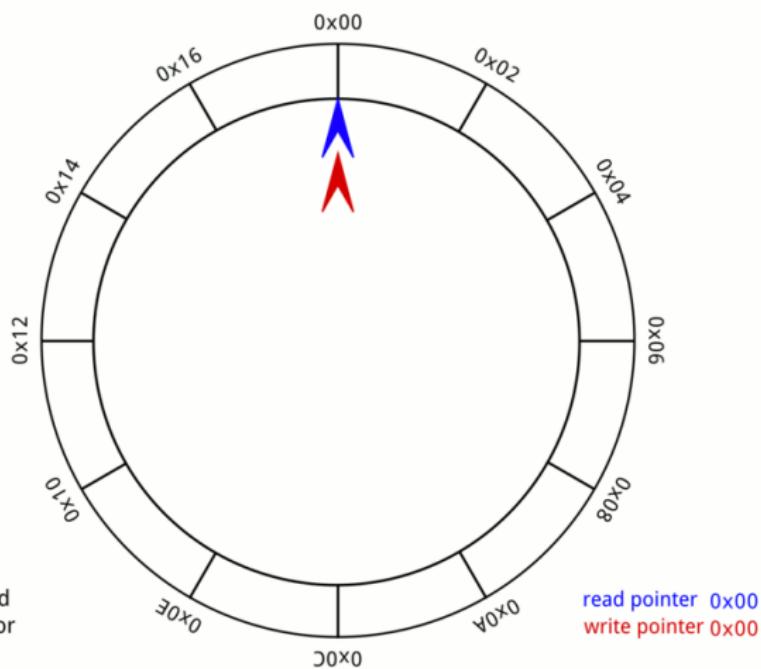


# Coda basata su vettore circolare

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



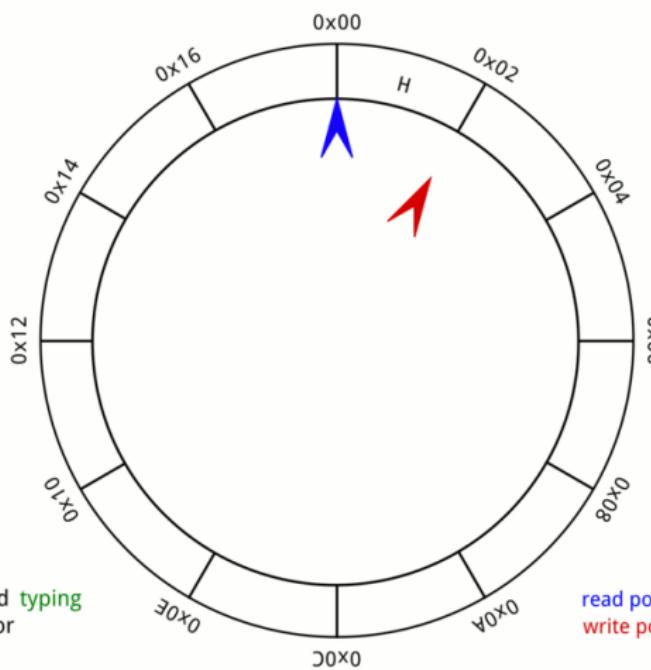
# Coda basata su vettore circolare



By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

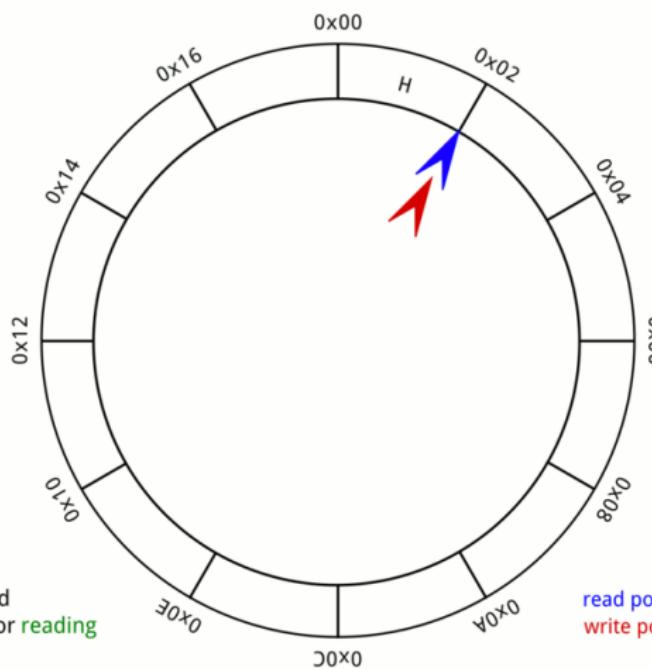
Commons

# Coda basata su vettore circolare



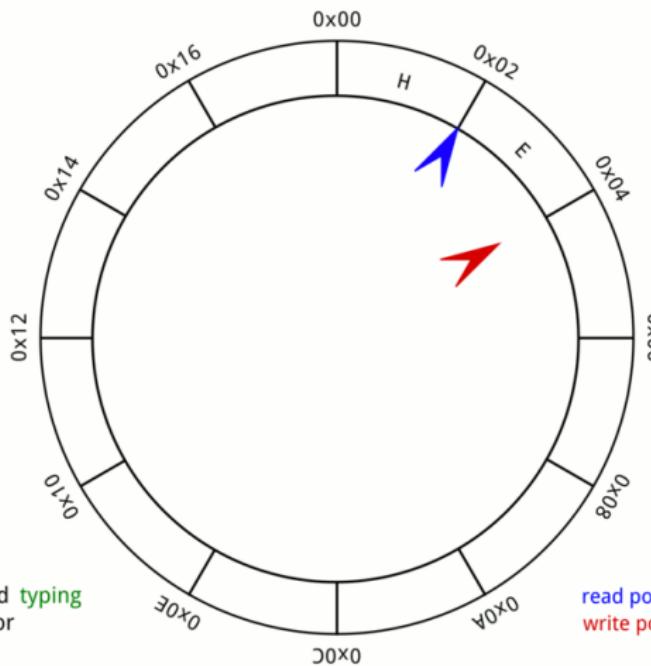
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia Commons

# Coda basata su vettore circolare



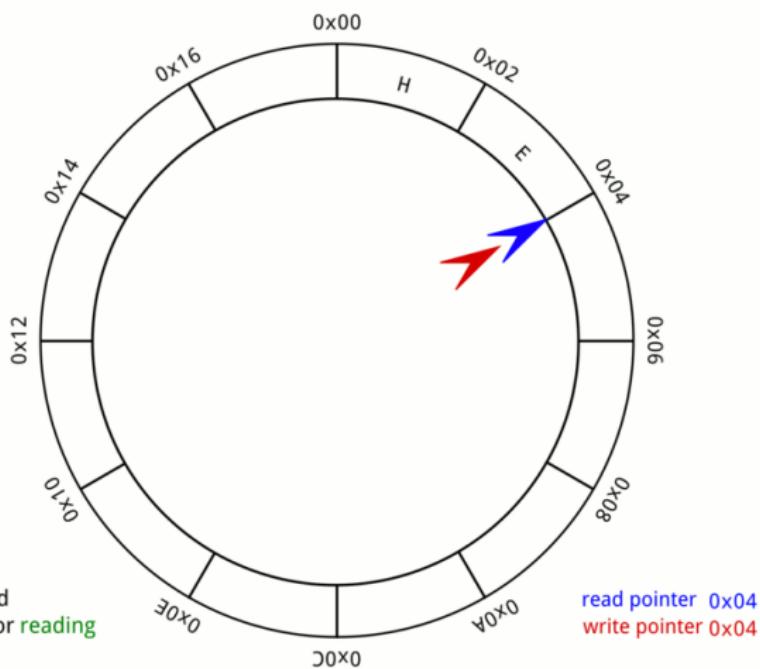
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia Commons

# Coda basata su vettore circolare



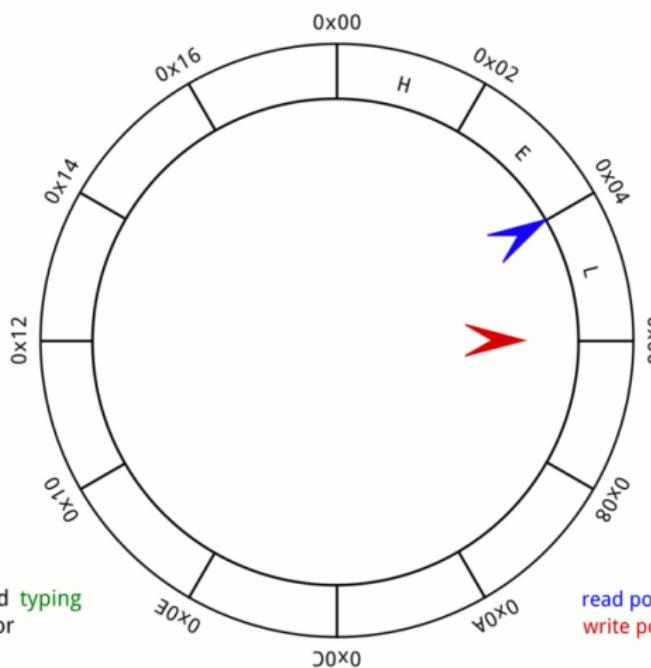
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia Commons

# Coda basata su vettore circolare



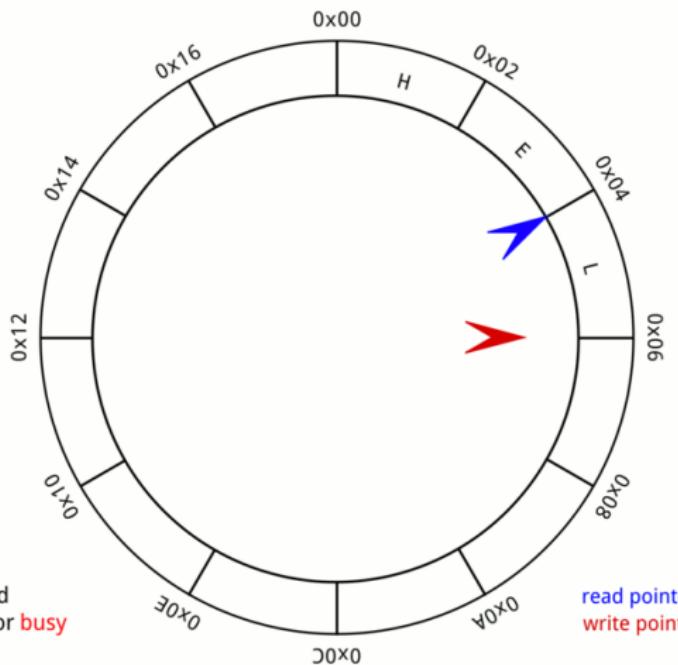
By Muhamad Ajjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia Commons

# Coda basata su vettore circolare



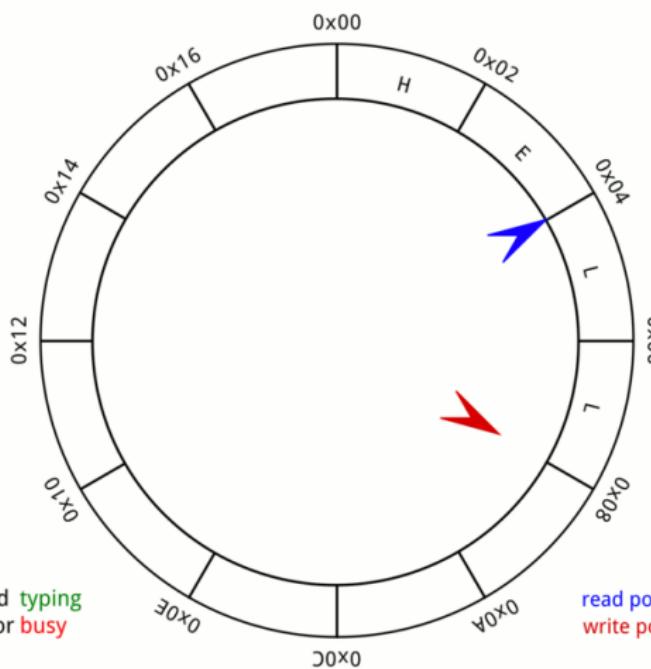
By Muhamad Ajjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia Commons

# Coda basata su vettore circolare



By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia Commons

# Coda basata su vettore circolare



By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

Commons

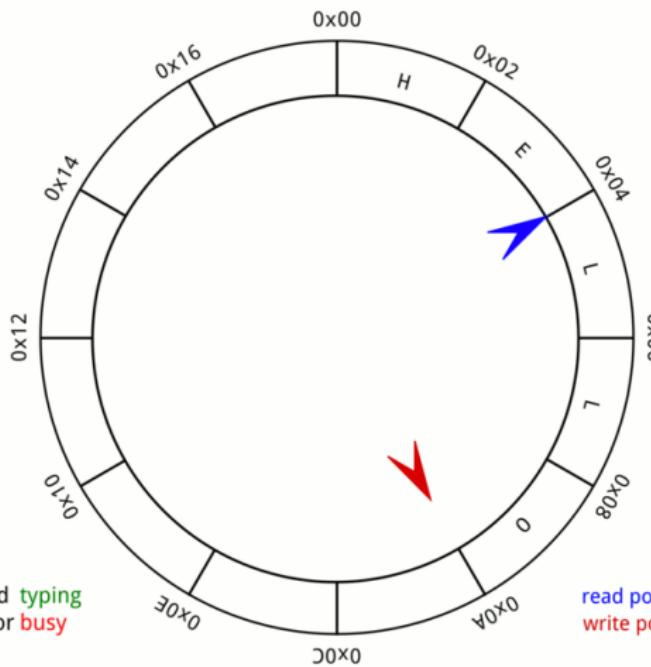
Alberto Montresor and Davide Rossi

ASD - Strutture dati

24 ottobre 2024

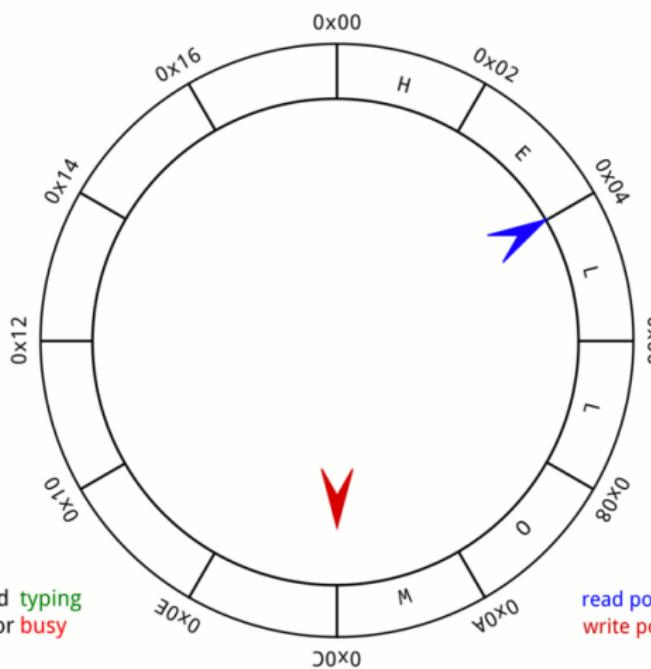
32 / 35

# Coda basata su vettore circolare



By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia Commons

# Coda basata su vettore circolare



By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

Commons

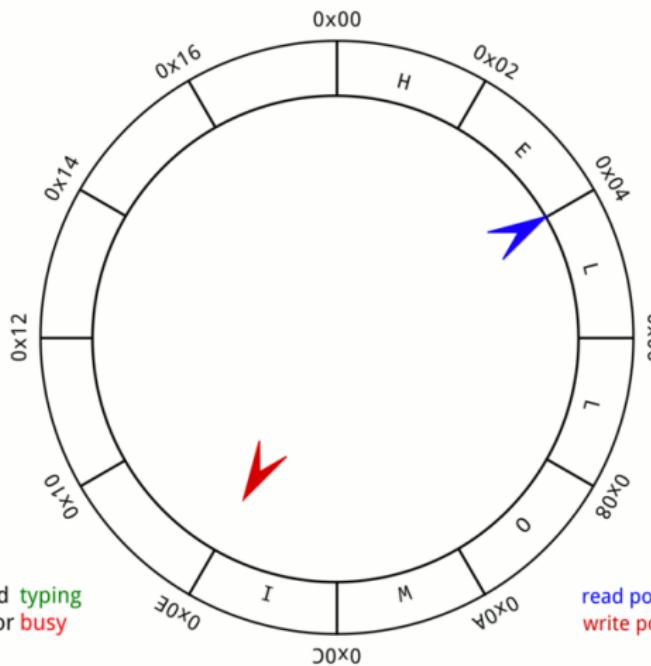
Alberto Montresor and Davide Rossi

ASD - Strutture dati

24 ottobre 2024

32 / 35

# Coda basata su vettore circolare



By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

Commons

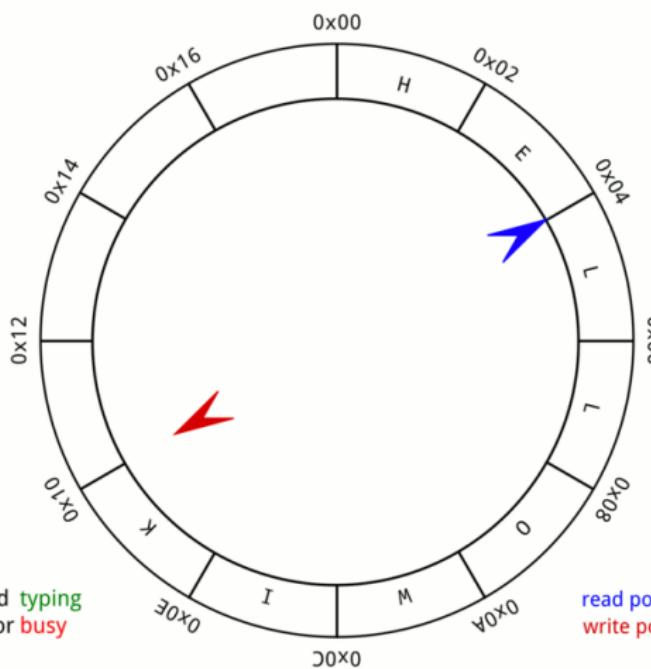
Alberto Montresor and Davide Rossi

ASD - Strutture dati

24 ottobre 2024

32 / 35

# Coda basata su vettore circolare



By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

Commons

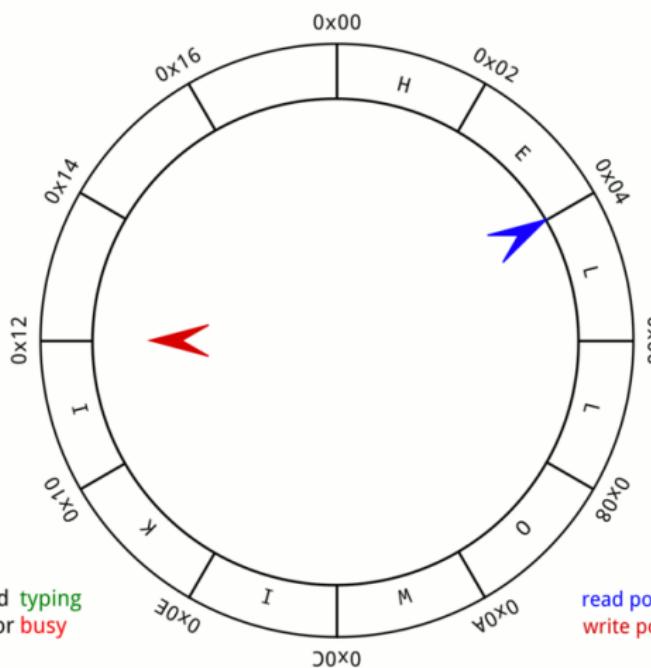
Alberto Montresor and Davide Rossi

ASD - Strutture dati

24 ottobre 2024

32 / 35

# Coda basata su vettore circolare



By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

Commons

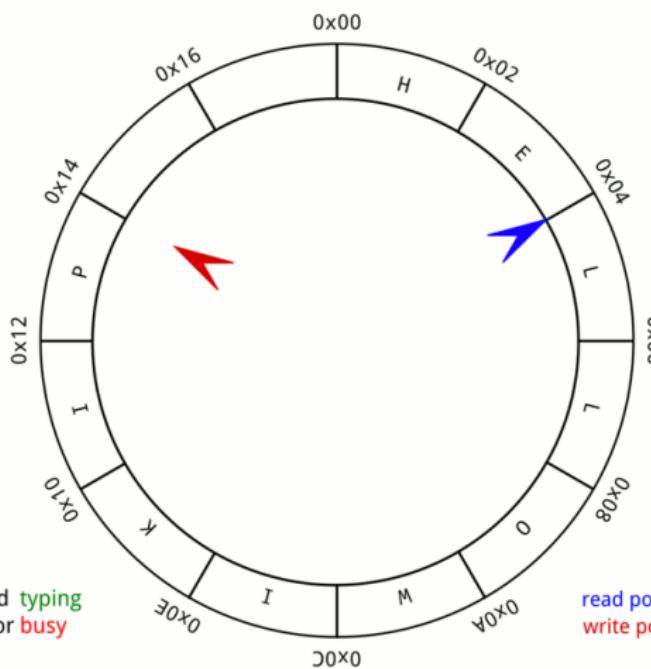
Alberto Montresor and Davide Rossi

ASD - Strutture dati

24 ottobre 2024

32 / 35

# Coda basata su vettore circolare



By Muhamad Ajjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

Commons

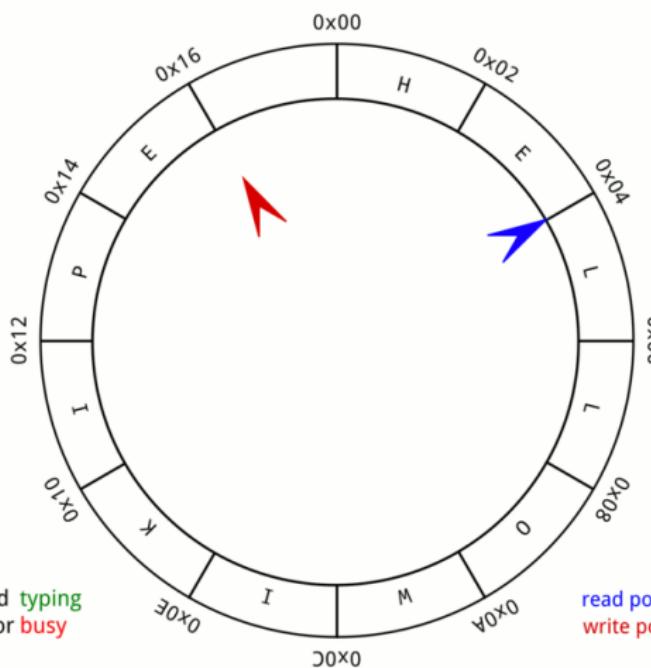
Alberto Montresor and Davide Rossi

ASD - Strutture dati

24 ottobre 2024

32 / 35

# Coda basata su vettore circolare



By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

Commons

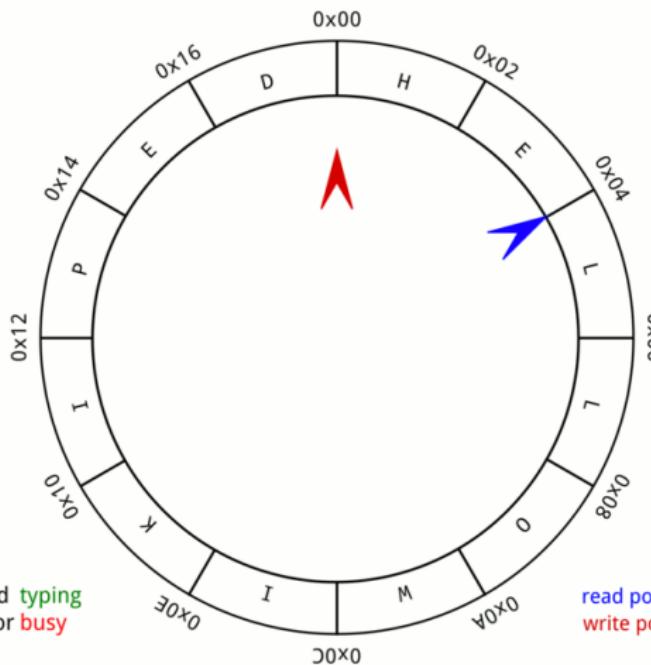
Alberto Montresor and Davide Rossi

ASD - Strutture dati

24 ottobre 2024

32 / 35

# Coda basata su vettore circolare



By Muhamad Ajjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

Commons

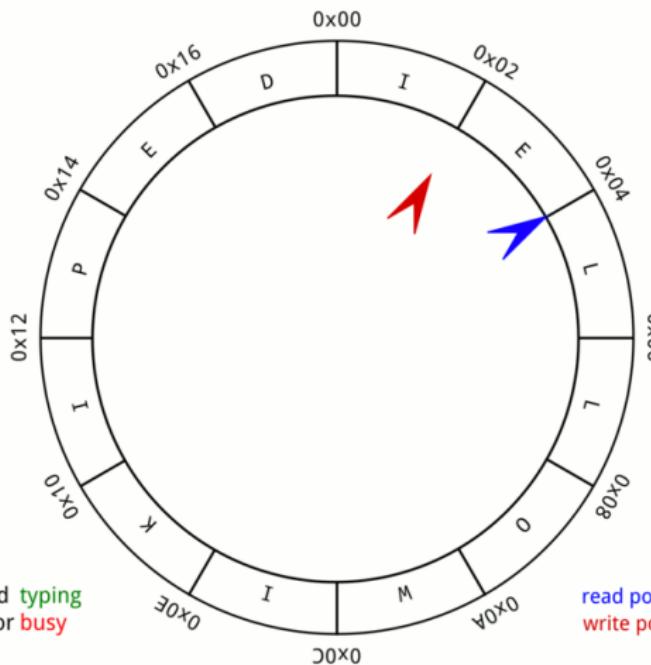
Alberto Montresor and Davide Rossi

ASD - Strutture dati

24 ottobre 2024

32 / 35

# Coda basata su vettore circolare



By Muhamad Ajjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

Commons

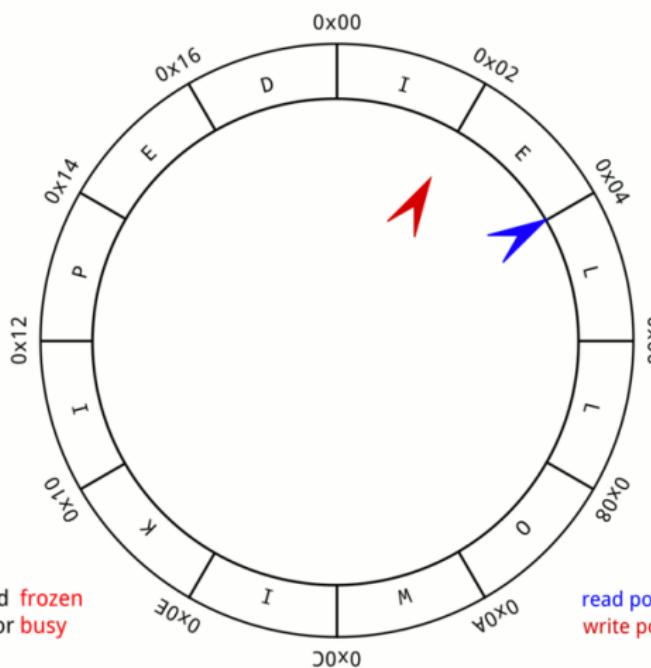
Alberto Montresor and Davide Rossi

ASD - Strutture dati

24 ottobre 2024

32 / 35

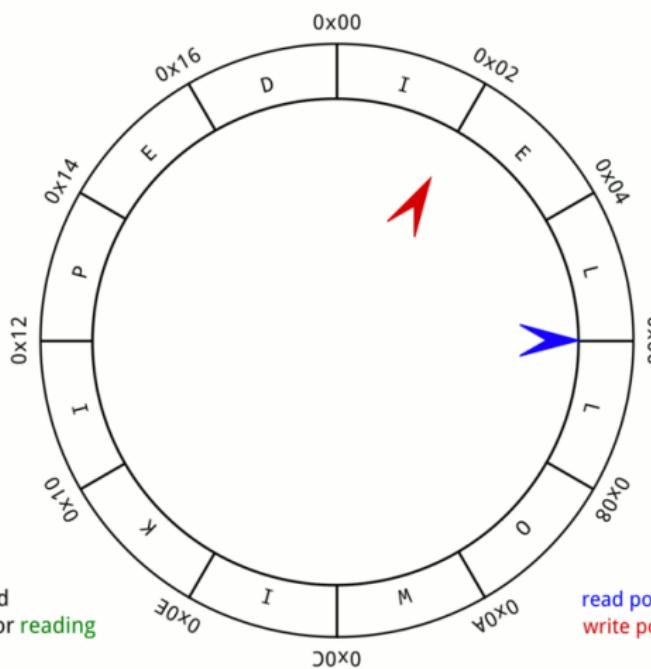
# Coda basata su vettore circolare



By Muhamad Ajjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

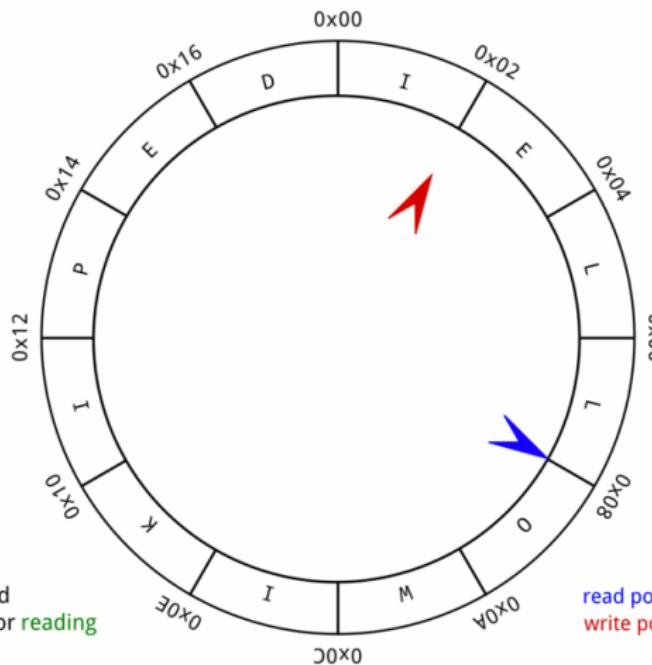
Commons

# Coda basata su vettore circolare



By Muhamad Ajjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia Commons

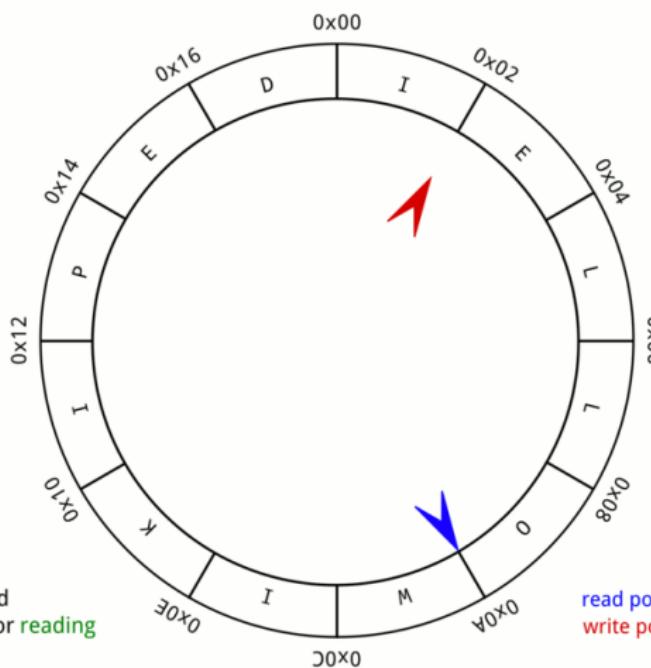
# Coda basata su vettore circolare



By Muhamad Ajjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

Commons

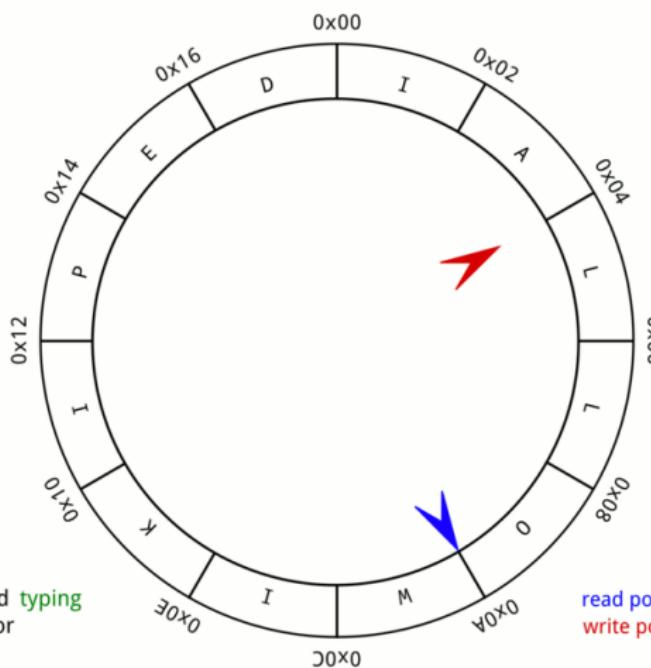
# Coda basata su vettore circolare



By Muhamad Ajjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

Commons

# Coda basata su vettore circolare



By Muhamad Ajjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

Commons

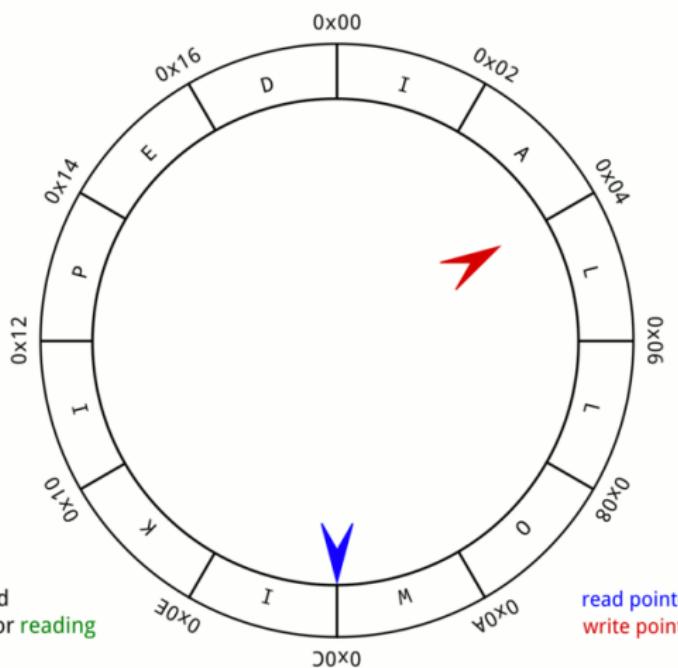
Alberto Montresor and Davide Rossi

ASD - Strutture dati

24 ottobre 2024

32 / 35

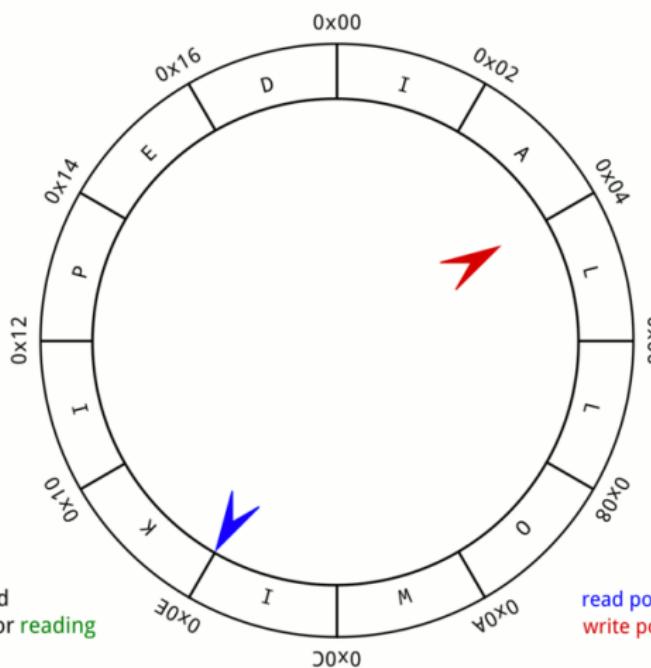
# Coda basata su vettore circolare



By Muhamad Ajjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

Commons

# Coda basata su vettore circolare



By Muhamad Ajjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

Commons

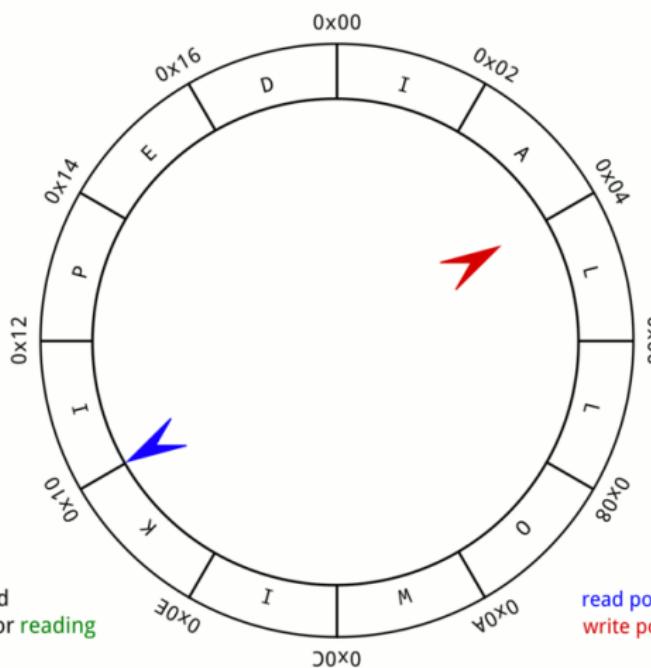
Alberto Montresor and Davide Rossi

ASD - Strutture dati

24 ottobre 2024

32 / 35

# Coda basata su vettore circolare



By Muhamad Ajjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

Commons

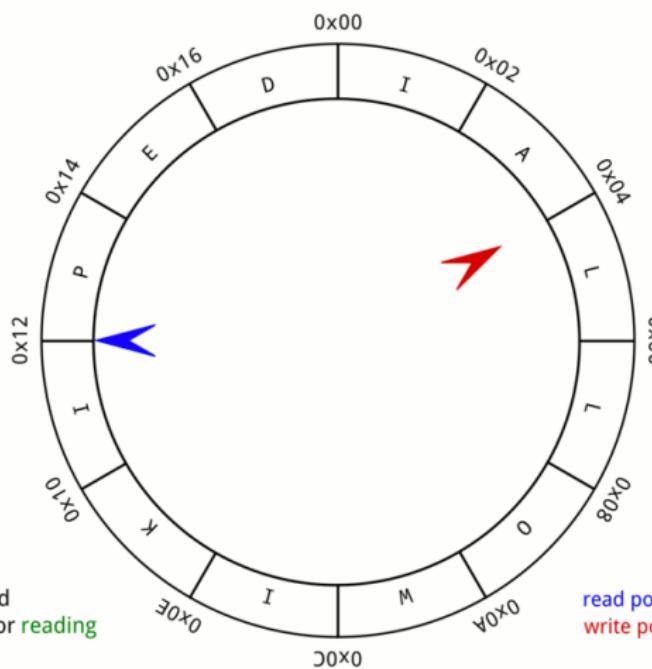
Alberto Montresor and Davide Rossi

ASD - Strutture dati

24 ottobre 2024

32 / 35

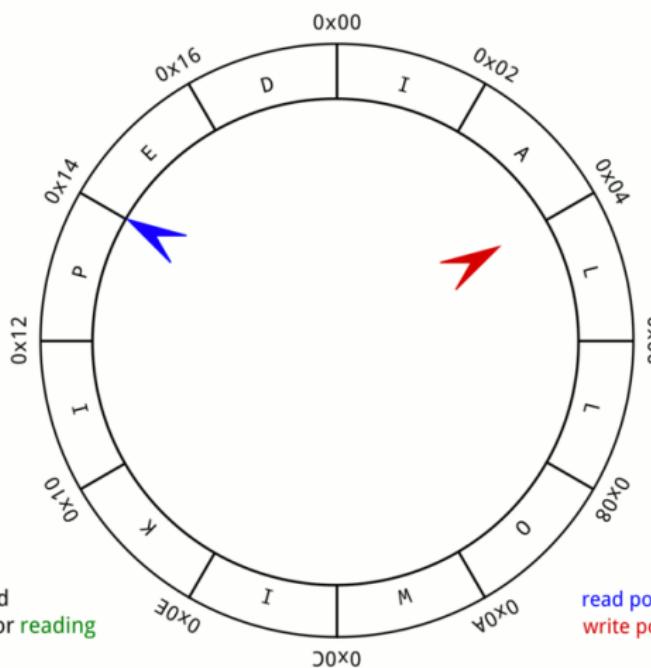
# Coda basata su vettore circolare



By Muhamad Ajjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

Commons

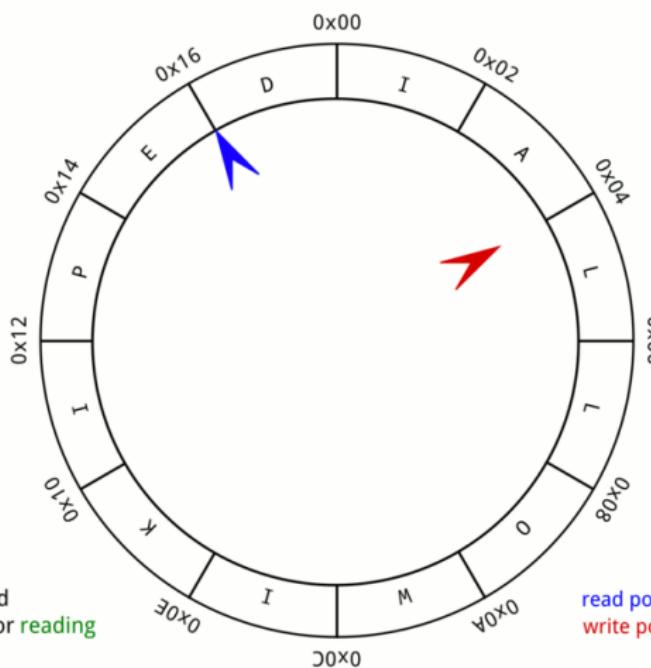
# Coda basata su vettore circolare



By Muhamad Ajjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

Commons

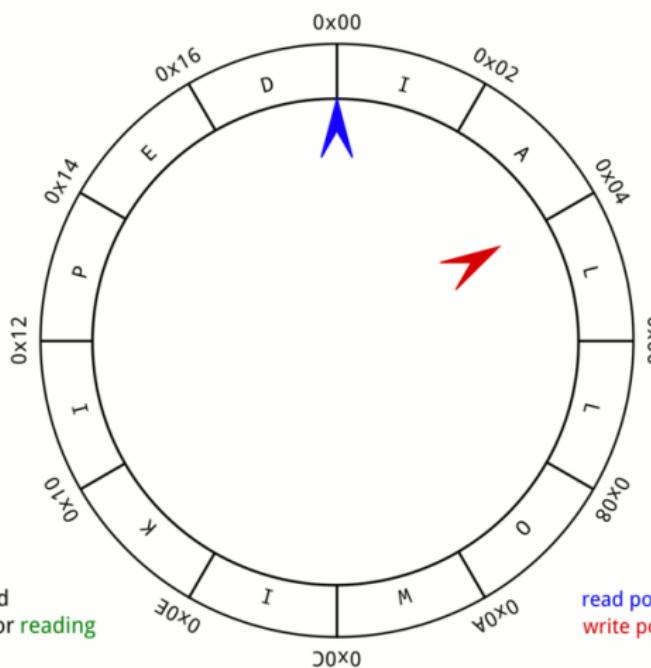
# Coda basata su vettore circolare



By Muhamad Ajjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

Commons

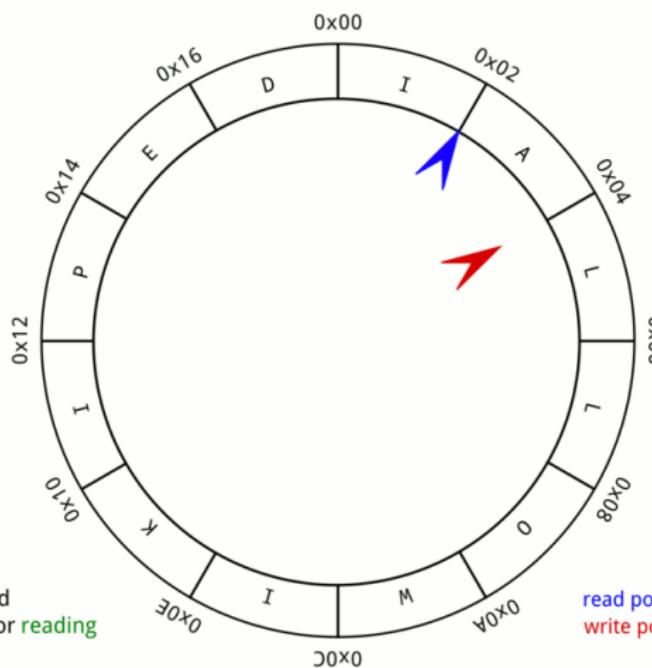
# Coda basata su vettore circolare



By Muhamad Ajjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

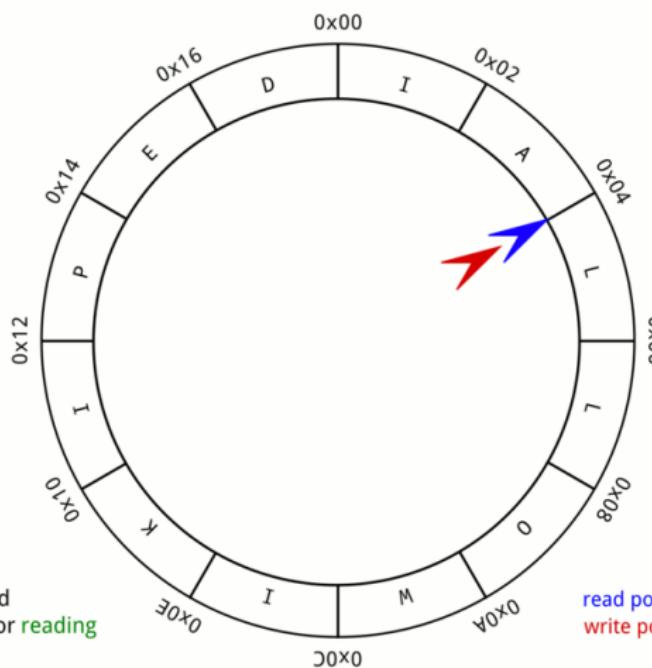
Commons

# Coda basata su vettore circolare



By Muhamad Ajjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia Commons

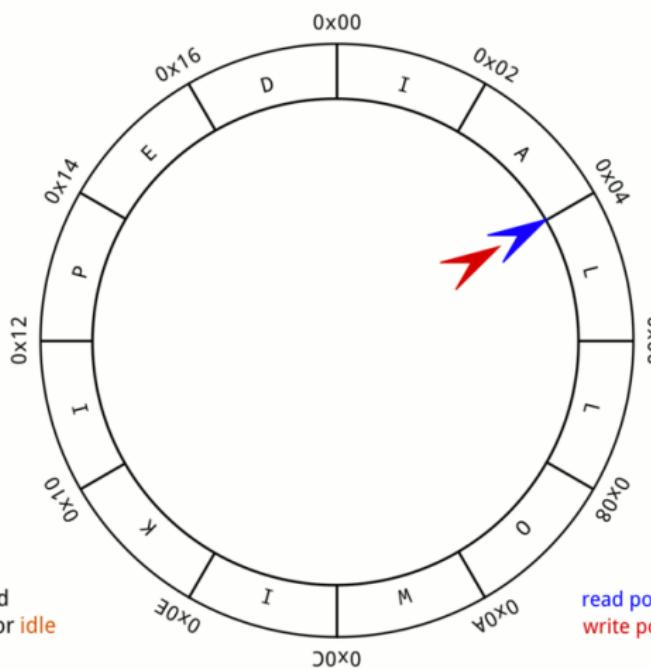
# Coda basata su vettore circolare



By Muhamad Ajjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

Commons

# Coda basata su vettore circolare



By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

Commons

# Coda basata su vettore circolare – Pseudocodice

testa → indice che punta al primo elemento inserito

## QUEUE

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

QUEUE Queue(int dim) *Costruttore*

```

QUEUE t = new QUEUE
t.A = new int[0...dim - 1]
t.m = dim      Inizializza una
t.testa = 0    coda vuota con
t.n = 0        array di dimensione
return t

```

ITEM top() *Restituisce l'elemento in testa*  
*precondition: n > 0 senza riauolverlo*  
return A[testa]

boolean isEmpty()

return n = 0

• *riavvolve e restituisce l'elemento in testa*

ITEM dequeue()

*precondition: n > 0 → coda non vuota*

ITEM t = A[testa]

testa = (testa + 1) mod m

n = n - 1

return t

• *Aggiorna la testa  
sposta la testa circolarmente*

enqueue(ITEM v) *aggiunge un elemento v in coda*

*precondition: n < m*

A[(testa + n) mod m] = v

n = n + 1

→ *Calcola la posizione di inserimento (testa + n) % m*

*usa l'aritmetica modulare per gestire il "riauvolgimento circolare"*

# Coda basata su vettore circolare – Java

```
public class VectorQueue implements Queue {  
  
    /** Element vector */  
    private Object[] A;  
  
    /** Current number of elements in the queue */  
    private int n;  
  
    /** Top element of the queue */  
    private int head;  
  
    public VectorQueue(int dim) {  
        n = 0;  
        head = 0;  
        A = new Object[dim];  
    }  
  
    public boolean isEmpty() {  
        return n==0;  
    }  
}
```

## Coda basata su vettore circolare – Java

```
public Object top() {  
    if (n == 0)  
        throw new IllegalStateException("Queue is empty");  
    return A[head];  
}  
  
public Object dequeue() {  
    if (n == 0)  
        throw new IllegalStateException("Queue is empty");  
    Object t = A[head];  
    head = (head+1) % A.length;  
    n = n-1;  
    return t;  
}  
  
public void enqueue(Object v) {  
    if (n == A.length)  
        throw new IllegalStateException("Queue is full");  
    A[(head+n) % A.length] = v;  
    n = n+1;  
}
```