Algoritmi e Strutture Dati

Strutture dati elementari

Alberto Montresor
Università di Trento

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/2.5/ or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

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`** (!, &&, ||)

© Alberto Montresor

Tipi di dati

- **“Specifica” e “implementazione” di un tipo di dato astratto**
  - *Specifica*:
    - “manuale d'uso”, nasconde i dettagli implementativi all'utilizzatore
  - *Implementazione*:
    - realizzazione vera e propria
- **Esempi**
  - Numeri reali vs IEEE754
  - Stack vs Stack basati su Array, Stack basati su puntatori
  - `java.util.Map` vs `java.util.TreeMap`

Strutture dati

- **I dati sono spesso riuniti in insiemi detti strutture dati**
  - sono particolari tipi di dato, caratterizzati più dall'*organizzazione dei dati* più che dal tipo dei dati stessi
  - il tipo dei dati contenuti può essere addirittura parametrico
- **Una struttura dati è composta quindi da:**
  - un modo sistematico di organizzare i dati
  - un insieme di operatori che permettono di manipolare la struttura
- **Alcune tipologie di strutture dati:**
  - *lineari* / *non lineari* (presenza di una sequenza)
  - *statiche* / *dinamiche* (variazione di dimensione, contenuto)
  - *omogenee* / *disomogenee* (dati contenuti)

© Alberto Montresor

Insiemi dinamici

- **Struttura dati "generale":** ***insieme dinamico***
  - Può crescere, contrarsi, cambiare contenuto
  - Operazioni base: inserimento, cancellazione, ricerca
  - Il tipo di insieme (= struttura) dipende dalle operazioni
- **Elementi**
  - Elemento: oggetto "puntato" da un riferimento/puntatore
  - Composto da:
    - campo chiave di identificazione
    - dati satellite
    - campi che fanno riferimento ad altri elementi dell'insieme

© Alberto Montresor

Insiemi dinamici

- **Operazioni di interrogazione**
  - Item search(Key k)
  - Item successor(Item x)
  - Item predecessor(Item x)
  - Item minimum()
  - Item maximum()
- **Operazioni di modifica**
  - void insert(Item x)
  - void delete(Item x)

© Alberto Montresor

Linked List

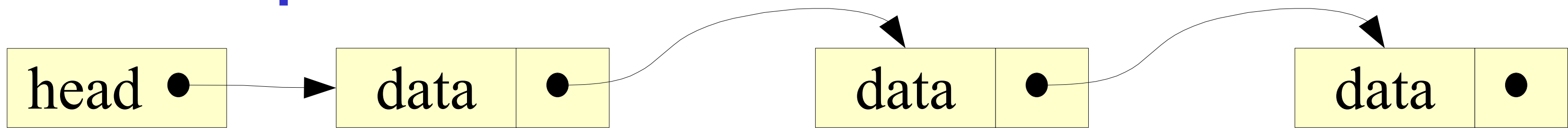
- **Liste puntate (Linked List)**
  - Una sequenza di nodi, contenenti dati arbitrari e 1-2 reference (puntatori, link) all'elemento successivo e/o precedente.
  - Struttura dati autoreferenziale
    - contiene link a dati aventi la stessa struttura
- **Tipo di accesso**

	Liste	Array
Random	O(n)	O(1)
Inserimento	O(1)	O(n)
Cancellazione	O(1)	O(n)

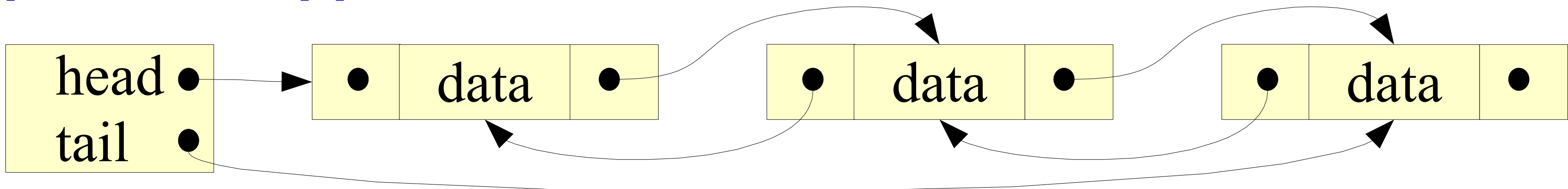
© Alberto Montresor

Linked List

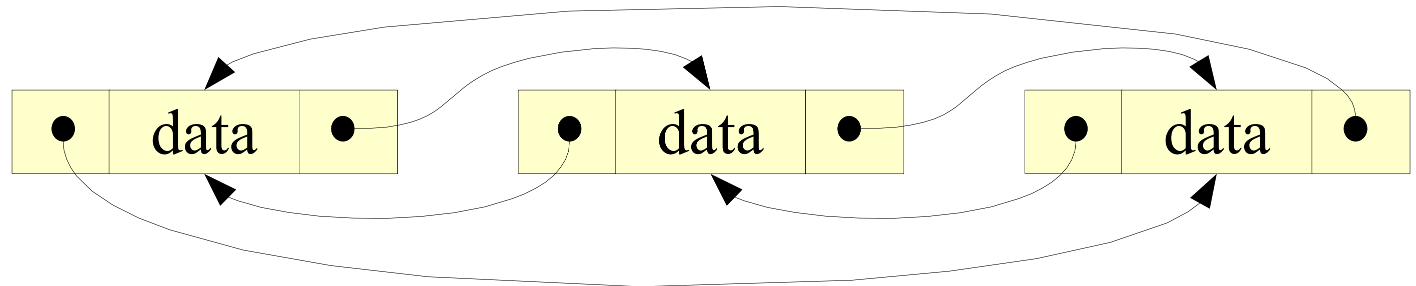
- **Liste puntate semplici**



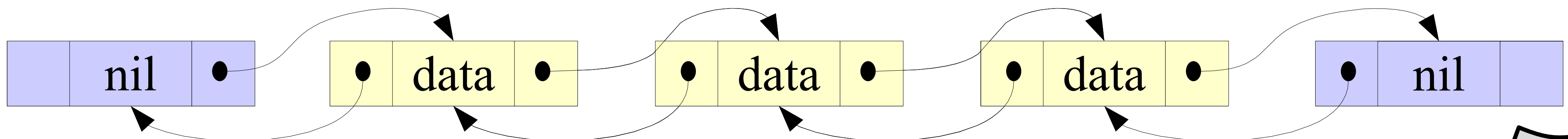
- **Liste puntate doppie**



- **Liste circolari (semplici, doppie)**



- **Liste con sentinella**



© Alberto Montresor

Linked List (Java)

```
class Node {

  ListItem next;
  ListItem prev;
  Key key;
  Object data;

  ListItem(Key key, Object data) {
    next = prev = null;
    this.key = key;
    this.data = data;
  }
}
```

© Alberto Montresor

Linked List (Java)

```
public class List {
  private ListItem head, tail;
  public List() {
    head = tail = null
  }
  public getHead() { return head; }
  public getTail() { return tail; }
  public successor(ListItem where)
   { return where.next; }
  public predecessor(ListItem where)
   { return where.prev; }
```

© Alberto Montresor

Linked List (Java)

```
public void insertHead(ListItem i) {
  if (head == null) { head = tail = i; }
  else {
    i.next = head;
    head.prev = i;
    head = i;
  }
}
public void insertTail(ListItem i) {
  if (head == null) { head = tail = i; }
  else {
    i.prev = tail;
    tail.prev = i;
    tail = i;
  }
}
```

© Alberto Montresor

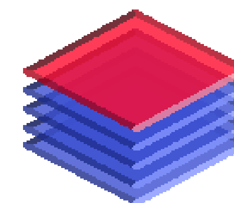
Linked List (Java)

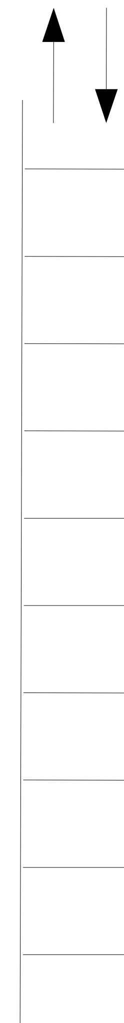
```
public List search(Key key) {
  ListItem i = head;
  while (i != null && i.key != key)
    i = i.next;
  return i;
}
public List delete(ListItem i) {
  if (i.prev == null) head = i.next;
  else i.prev.next = next;
  if (i.next == null) tail = i.prev;
  else i.next.prev = prev;
}
```

© Alberto Montresor

Stack

- **Una pila (stack)**
  - è un insieme dinamico in cui l'elemento rimosso dall'operazione di cancellazione è predeterminato: "*quello che per meno tempo è rimasto nell'insieme*"
  - politica *"last in, first out" (LIFO)*



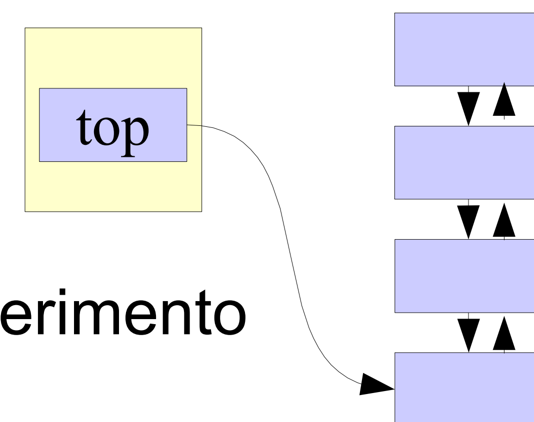


- **Operazioni previste (tutte O(1))**
  - *void push(Item)* # inserisce un elemento
  - *Item pop()* # rimuove l'ultimo elemento
  - *Item top()* # non rimuove l'item; legge solamente
  - *boolean isEmpty()*

© Alberto Montresor

Stack

- **Possibili utilizzi**
  - Nei linguaggi con procedure: gestione dei record di attivazione
  - Nei linguaggi stack-oriented:
    - Tutte le operazioni elementari lavorano prendendo uno-due operandi dallo stack e inserendo il risultato nello stack
    - Es: Postscript, Java bytecode
- **Possibili implementazioni**
  - Tramite *liste puntate doppie*
    - puntatore all'elemento top, per estrazione/inserimento
  - Tramite *array*
    - dimensione limitata, overhead più basso



Stack

- **Reverse Polish Notation (PNG), o notazione postfissa**
  - Espressioni aritmetiche in cui gli operatori seguono gli operandi
  - Definita dalla grammatica:

    <expr> ::= <numeral> | *<espr> <espr> <operator>*
  - Esempi:
    - (7 + 3) x 5 si traduce in 7 3 + 5 x
    - 7 + 3 x 5 si traduce in 7 3 5 x +
- **Esempio di funzionamento**
  - push 7 — 7
  - push 3 — 3 7
  - push 5 — 5 3 7
  - push x — 15 7
  - push + — 22

© Alberto Montresor

Stack: implementazione tramite array (Java)

```
public class Stack {
  private Object[] buffer = new int[MAX_SIZE];
  private int size;    // Number of elements

  public Stack() { size = 0; }
  public boolean isEmpty() { return size=0; }
  public Object top() {
    if (size == 0) throw new Exception("Empty");
    else return buffer[size-1];
  }
```

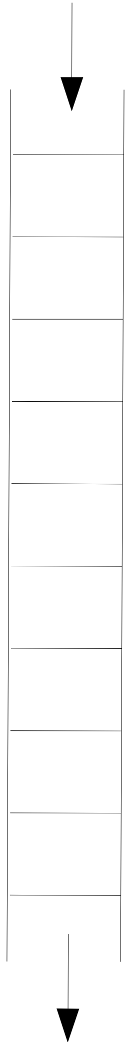
© Alberto Montresor

Stack: implementazione tramite array (Java)

```
  public void push(Object o) {
    if (size < array.length)
      throw Exception("Full");
    buffer[size++] = o;
  }
  public Object pop() {
    if (size == 0) throw new Exception("Empty");
    return buffer[--size];
  }
}
```

Queue

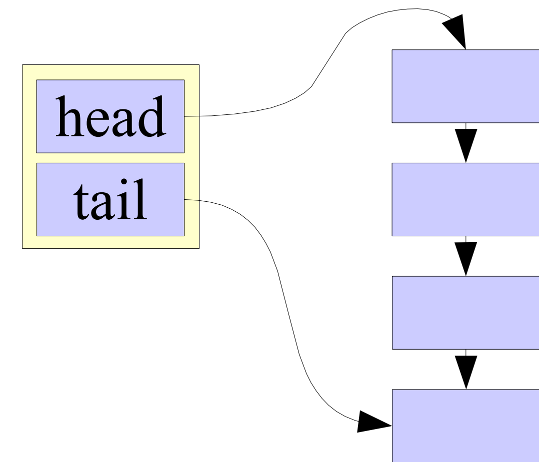
- **Una coda (queue)**
  - è un insieme dinamico in cui l'elemento rimosso dall'operazione di cancellazione è predeterminato: "*quello che per più tempo è rimasto nell'insieme*"
  - politica *"first in, first out" (FIFO)*
- **Operazioni previste (tutte O(1))**
  - *void enqueue(Item)* # sinonimi: put, add, insert
  - *Item dequeue()* # sinonimi: removeFirst, extract
  - *Item head()* # non rimuove l'item; sinonimi *get*
  - *boolean isEmpty()*



© Alberto Montresor

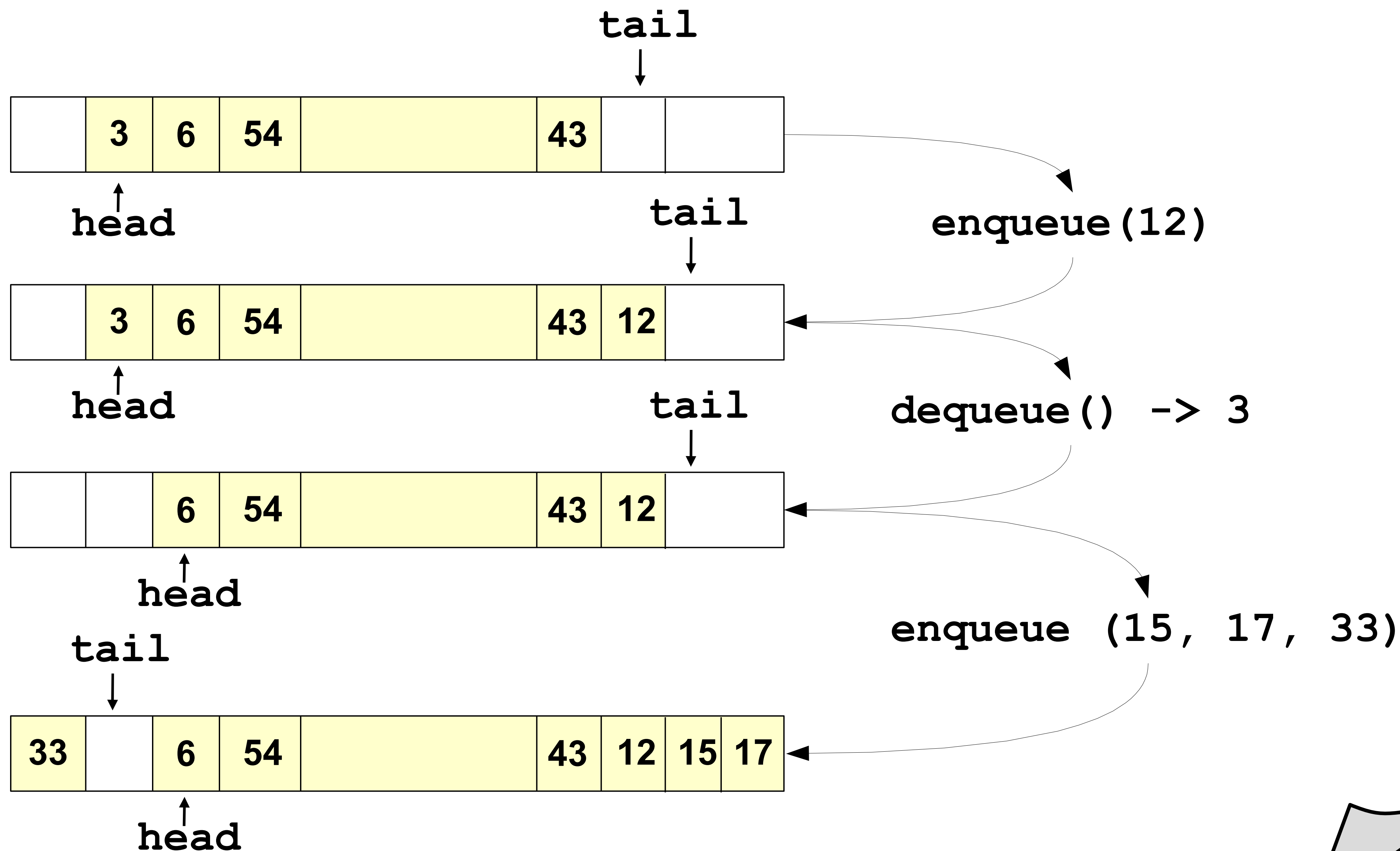
Queue

- **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 puntate semplici*
    - necessari
      - puntatore head (inizio della coda), per estrazione
      - puntatore tail (inizio della coda), per inserimento
  - Tramite *array circolari*
    - dimensione limitata, overhead più basso

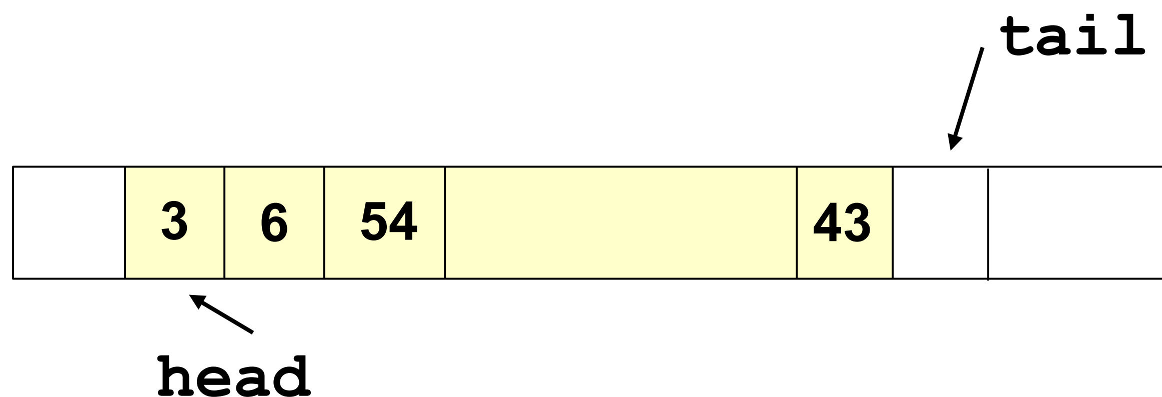
Queue: Implementazione tramite array circolari

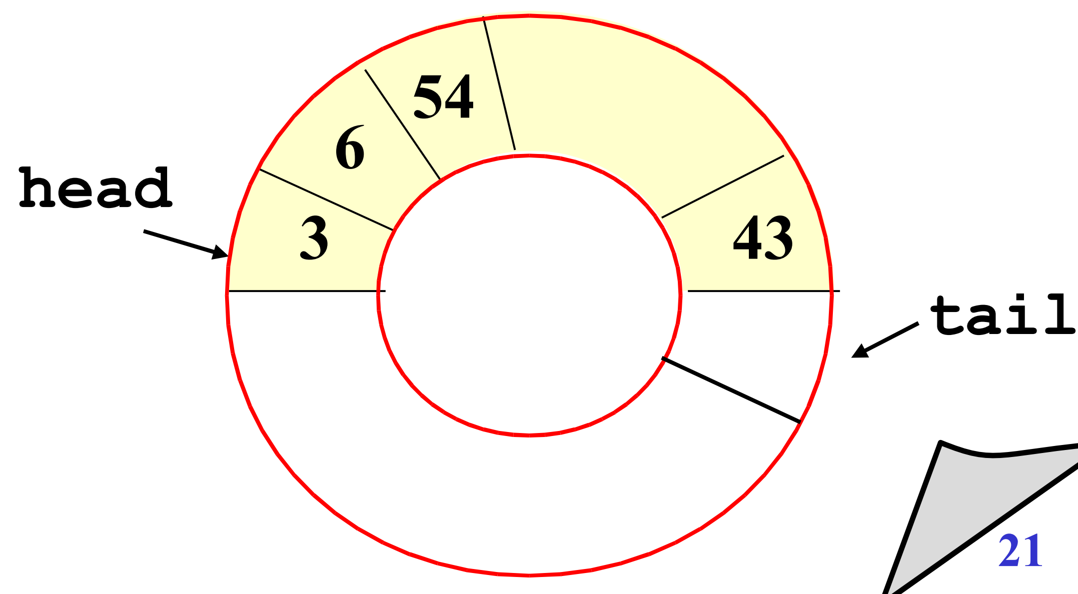


© Alberto Montresor

Queue: Implementazione tramite array circolari

- **La “finestra” dell’array occupata dalla coda si sposta lungo l’array!**
- **Dettagli implementativi**
  - L'array circolare può essere implementato con un'operazione di modulo
  - Bisogna prestare attenzione ai problemi di overflow (buffer pieno)





© Alberto Montresor

Queue: Implementazione tramite array circolari (Java)

```
public class Queue {
  private Object[] buffer = new int[MAX_SIZE];
  private int head;    // "Dequeuing" index
  private int tail;    // "Enqueuing" index
  private int size;    // Number of elements

  public Queue() { head = tail = size = 0; }
  public boolean isEmpty() { return size=0; }
  public Object head() {
    if (size == 0) throw new Exception("Empty");
    else return buffer[head];
  }
```

© Alberto Montresor

Queue: Implementazione tramite array circolari (Java)

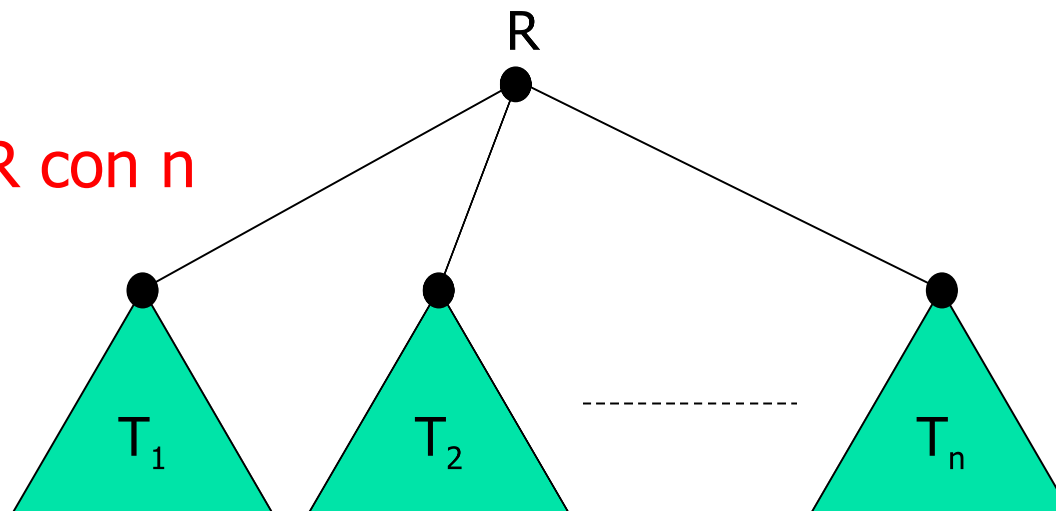
```
public void enqueue(Object o) {
  if (size == array.length)
     throw Exception("Full");
  buffer[tail] = o;
  tail =(tail+1) % array.length;
  size++;
}
public Object dequeue() {
  if (size == 0) return null;
  Object res = buffer[head];
  head =(head+1) % array.length;
  size--;
  return res;
}
}
```

© Alberto Montresor

Alberi radicati

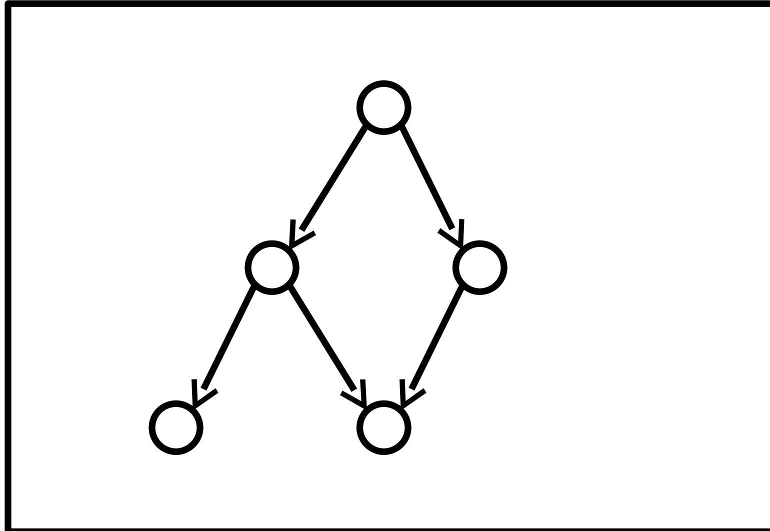
- **Albero: definizione informale**
  - E un insieme dinamico i cui elementi hanno relazioni di tipo gerarchico
- **Albero: definizione formale**
  - Insieme vuoto di nodi oppure costituito da una radice *R* e da 0 o più alberi (detti sottoalberi)
  - La radice di ogni sottoalbero è collegata a *R* da un arco (orientato)

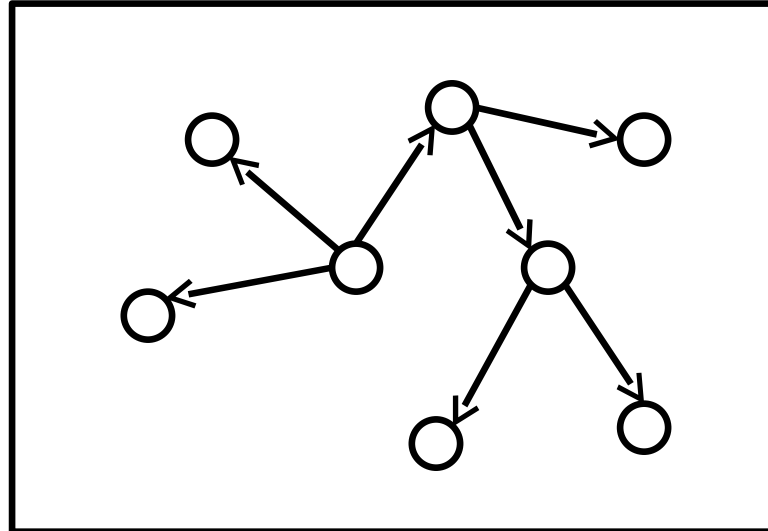
es.: radice R con n sottoalberi

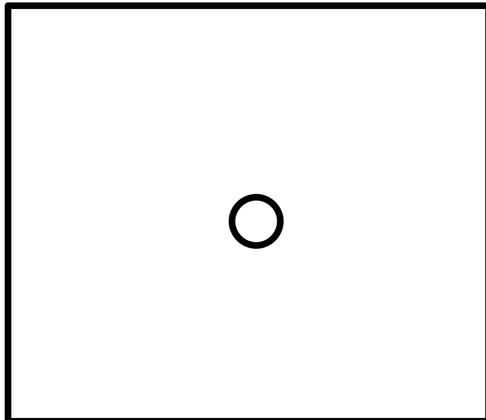


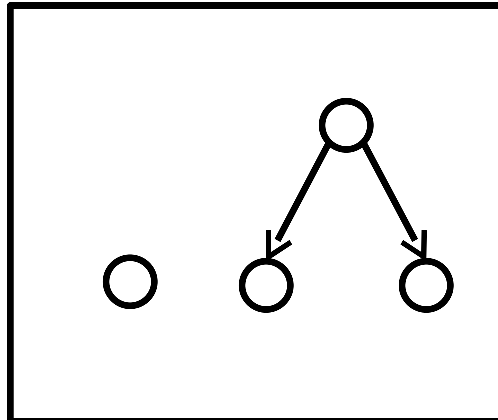
© Alberto Montresor

Alberi?









Algoritmi di visita degli alberi

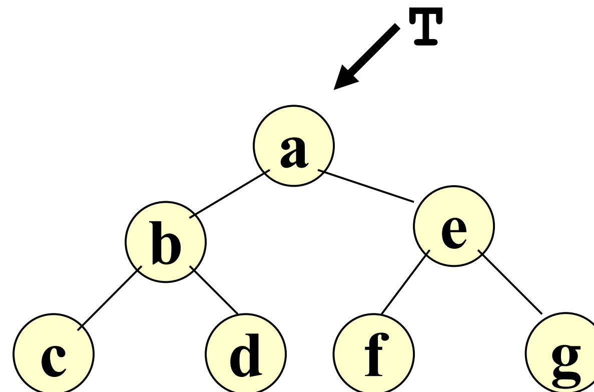
- **Visita (o attraversamento) di un albero:**
  - Algoritmo per “visitare” tutti i nodi di un albero
- **In profondità (depth-first search, a scandaglio): DFS**
  - Vengono visitati i rami, uno dopo l’altro
  - Tre varianti
- **In ampiezza (breadth-first search, a ventaglio): BFS**
  - A livelli, partendo dalla radice

© Alberto Montresor

Visita alberi binari: in profondità pre-ordine

```
visita-preordine(T)
  // Visita T
  if (T.left() ≠ nil)
    visita-preordine(T.left())
  if (T.right() ≠ nil)
    visita-preordine(T.right())
```

Estendibile ad alberi di grado n != 2

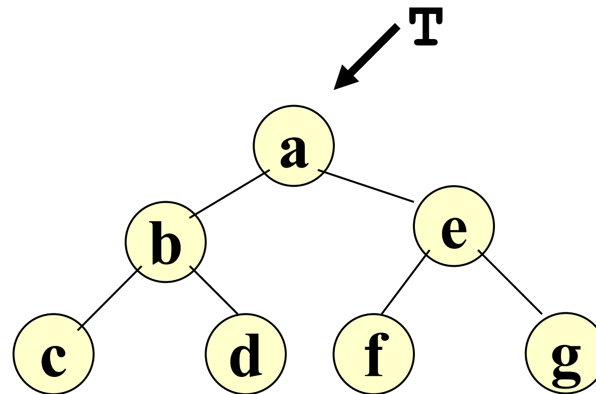


Sequenza: a b c d e f g

© Alberto Montresor

Visita alberi binari: in profondità in-ordine (simmetrica)

```
visita-inordine(T)
  if (T.left() ≠ nil)
    visita-inordine(T.left())
  // Visita T
  if (T.right() ≠ nil)
    visita-inordine(T.right())
```



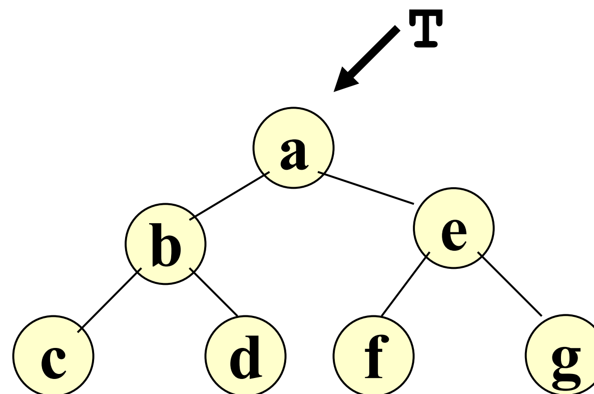
Sequenza: c b d a f e g

© Alberto Montresor

Visita alberi binari: in profondità post-ordine

```
visita-postordine(T)
  if (T.left() ≠ nil)
    visita-postordine(T.left())
  if (T.right() ≠ nil)
    visita-postordine(T.right())
  // Visita T
```

Estendibile ad alberi di grado n != 2



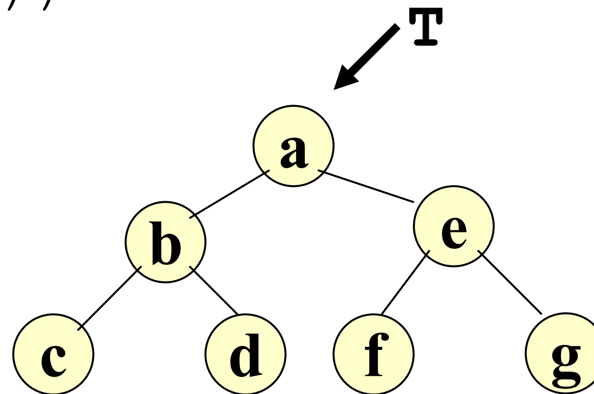
Sequenza: c d b f g e a

© Alberto Montresor

Visita alberi binari: in ampiezza

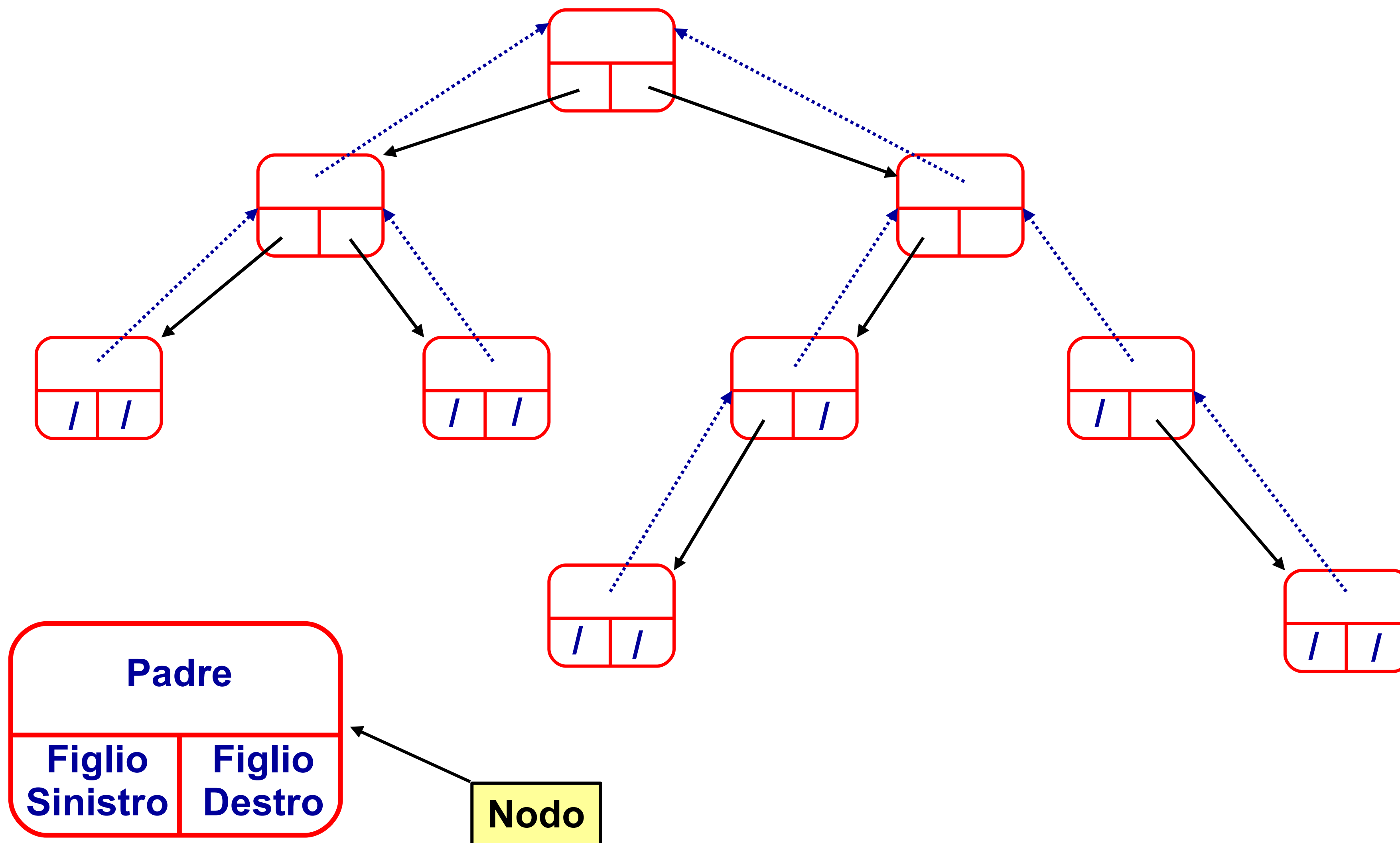
```
visita-ampiezza(T)
  q = new Queue()
  q.insert(T)
  while not q.empty() do
    p := q.dequeue()
    visita p
    q.enqueue(p.left())
    q.enqueue(p.right())
```

Estendibile ad alberi di grado n != 2



Sequenza: a b e c d f g

Implementazione di alberi binari

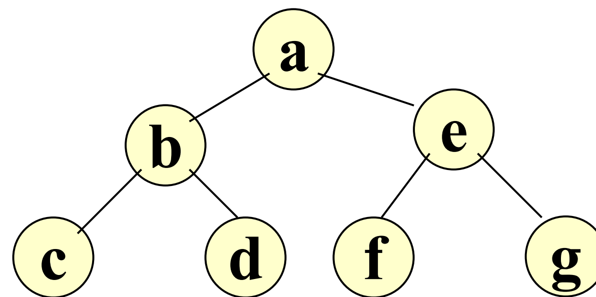


© Alberto Montresor

Implementazione di alberi generali: vettore dei padri

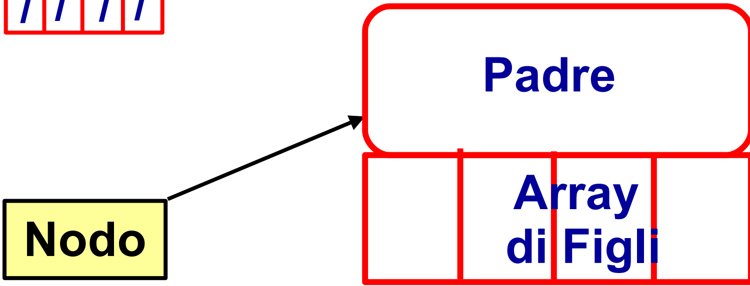
- L'albero è rappresentato da un vettore i cui elementi contengono l'indice del padre
- Esempio:

0	a
1	b
1	e
2	c
2	d
3	f
4	g



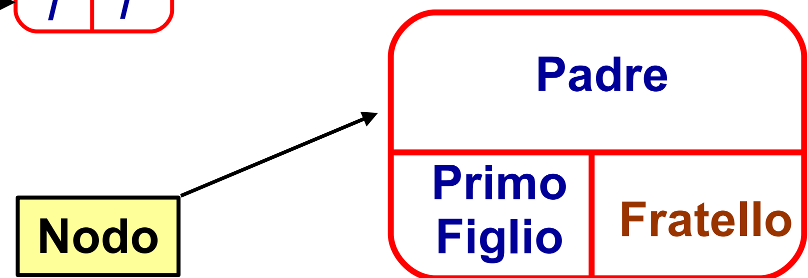
© Alberto Montresor

Implementazione di alberi generali: array di figli



Rischio di sprecare memoria se molti nodi hanno grado minore del grado massimo k.

Implementazione di alberi generali: nodo fratello



***Soluzione**: usare una lista di figli (**fratelli**).*