

Code

Le code: specifiche e realizzazioni attraverso rappresentazioni sequenziali e collegate.

Algoritmi e Strutture Dati + Lab

A.A. 18/19

Informatica

Università degli Studi di Bari "Aldo Moro"

Nicola Di Mauro

Le code

- Una coda è un tipo astratto che consente di rappresentare una sequenza di elementi in cui è possibile aggiungere elementi ad un estremo (“il fondo”) e togliere elementi dall'altro estremo (“la testa”)
- Tale disciplina di accesso è detta fifo (first in first out)
- È particolarmente adatta a rappresentare sequenze nelle quali l'elemento viene elaborato secondo l'ordine di arrivo (lista d'attesa, insieme di dispositivi in attesa di assegnazione di risorse, etc.)

Specifica sintattica

- Tipi: coda, boolean, tipoelem
- Operatori:
 - creacoda: () → coda
 - codavuota: (coda) → boolean
 - leggicoda: (coda) → tipoelem
 - fuoricoda: (coda) → coda
 - incoda: (tipoelem, coda) → coda

Specifica semantica

- Tipi:
 - coda: insieme delle sequenze $q = \langle a_1, a_2, \dots, a_n \rangle$, $n \geq 0$, di elementi di tipo `tipoelem` caratterizzata dall'accesso `fifo`;
 - boolean: insieme dei valori di verità;
- Operatori:
 - `creacoda = q'`
 - post: $q' = \langle \rangle$ sequenza vuota
 - `codavuota(q) = b`
 - post: $b = \text{vero}$ se $q = \langle \rangle$, $b = \text{falso}$ altrimenti
 - `leggicoda(q) = a`
 - pre: $q = \langle a_1, a_2, \dots, a_n \rangle$ e $n \geq 1$
 - post: $a = a_1$

Specifica semantica /2

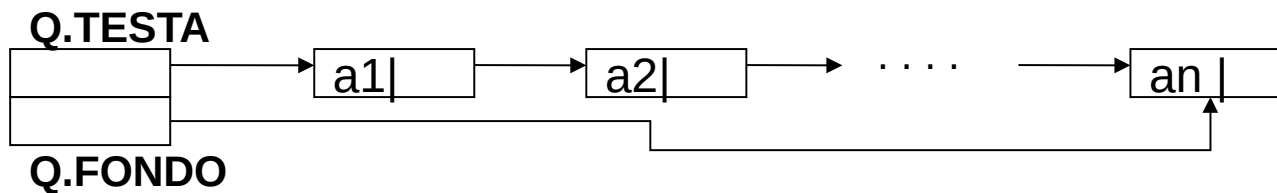
- fuoricoda(q) = q'
 - pre: $q = \langle a_1, a_2, \dots, a_n \rangle$ e $n \geq 1$
 - post: $q' = \langle a_2, a_3, \dots, a_n \rangle$ se $n > 1$, $q' = \langle \rangle$ se $n = 1$
- incoda(a, q) = q'
 - pre: $q = \langle a_1, a_2, \dots, a_n \rangle$ e $n \geq 0$
 - post: $q' = \langle a_1, a_2, \dots, a_n, a \rangle$

Rappresentazione

- In generale le possibili rappresentazioni delle code sono analoghe a quelle delle pile con l'attenzione che è conveniente consentire l'accesso sia all'elemento inserito per primo sia all'elemento inserito per ultimo.

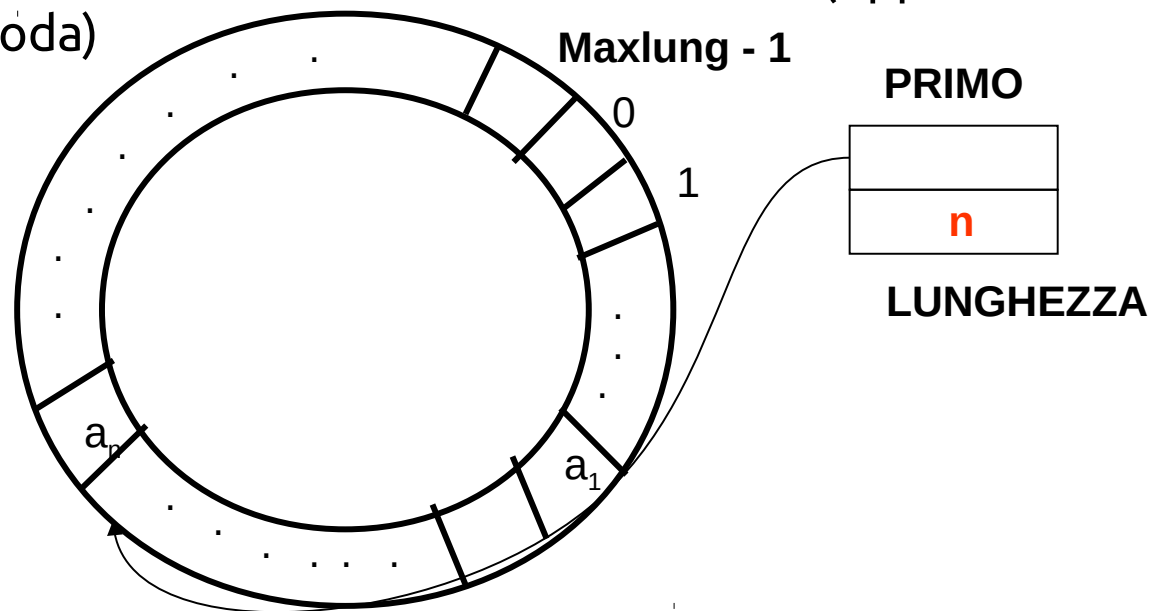
Realizzazione con puntatori

- La coda è realizzata con n celle, la prima delle quali è indirizzata da un puntatore “testa” e l'ultima da un puntatore “fondo”
- La coda vuota è individuata dal valore nullo null del puntatore di testa.



Realizzazione con vettore circolare

- Per le code la rappresentazione sequenziale non è agevole come per le pile
 - È utile gestire l'array in modo circolare.
- Il vettore circolare è inteso come un array di maxlung elementi, con indice da 0 a $\text{maxlung} - 1$, in cui consideriamo l'elemento di indice 0 come successore di quello di indice $\text{maxlung} - 1$.
- Si utilizzano due variabili primo e ultimo
 - il valore di primo indica la posizione dell'array in cui è memorizzato l'elemento inserito per primo, ultimo si riferisce all'ultimo elemento inserito (oppure definisce la lunghezza della coda)



Esercizi su pile e code

- Si vuole realizzare un programma che prende una coda di interi e restituisce un'altra coda ottenuta dalla prima considerando solo valori positivi.
- Trascuriamo le dichiarazioni relative alla implementazione della coda

```
estrai (coda q, coda q1 per riferimento)
    creacoda(q1)
    while not codavuota(q)
        e = leggicoda(q)
        if e > 0 then
            incoda(e, q1)
        fuoricoda(q)
```

Esercizi su pile e code /2

- Se volessimo conservare la coda originale dovremmo usare una coda ausiliaria.

```
estrai1 (coda q per riferimento, coda q1 per riferimento)
    creacoda(q1)
    creacoda(qaux)
    while not codavuota(q) do
        e = leggicoda(q)
        if e > 0 then
            incoda (e,q1)
        fuoricoda(q)
        incoda(e, qaux)
    creacoda(q)
    // ripristino della coda originaria
    while not codavuota(qaux) do
        e = leggicoda(qaux)
        incoda (e,q)
        fuoricoda (qaux)
```

Aritmetica postfissa

- Le espressioni aritmetiche sono scritte in notazione infissa, cioè i simboli delle operazioni appaiono tra gli oggetti su cui operano.
- Nella notazione postfissa gli operatori si pongono dopo gli oggetti su cui operano
 - $35 + 17 * (40 - 9) - 7$
 - $35\ 17\ 40\ 9 - * + 7 -$
- Questa notazione è definita polacca (si deve al matematico lukasiewicz)

Aritmetica postfissa

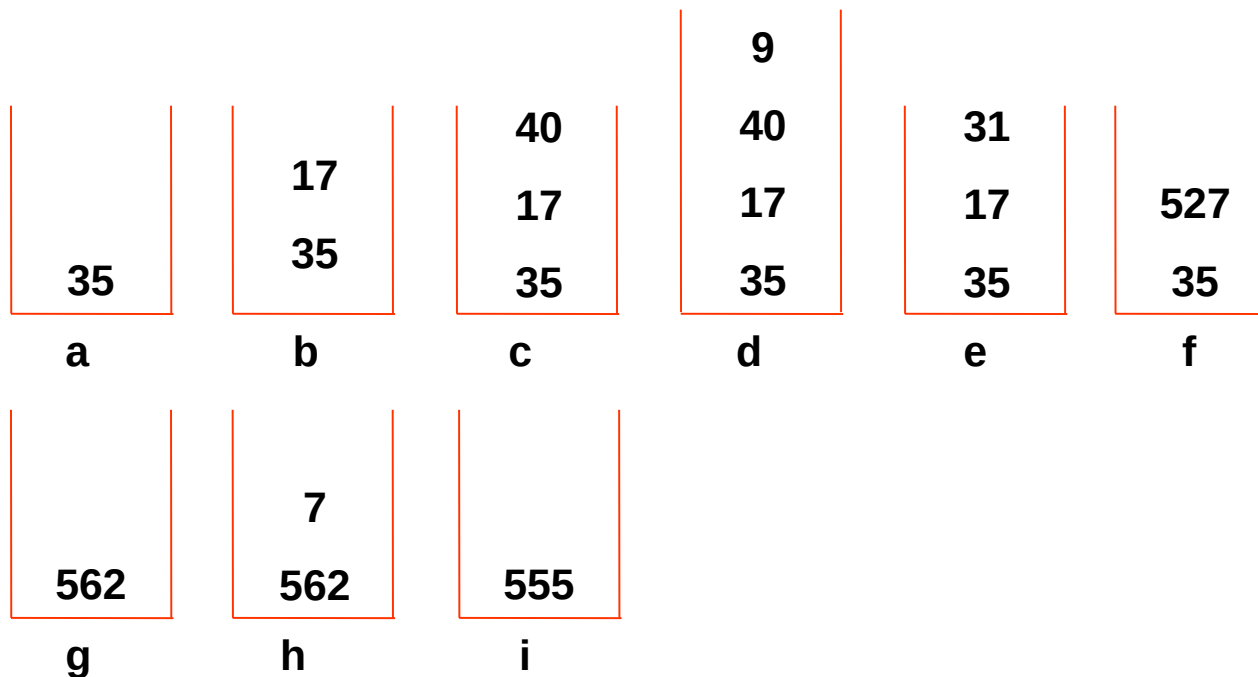
- Una notazione polacca è una qualunque serie di operandi aritmetici x, y, \dots e operatori binari $op (+, -, *, /)$ che si può formare mediante le regole seguenti:
 - ogni operando x è una notazione polacca
 - se $p1$ e $p2$ sono notazioni polacche allora $p1p2 op$ è una notazione polacca
 - $48+63$ corrisponde a $p1 \ 48 \ 63 \ +$
 - $52*4$ corrisponde a $p2 \ 52 \ 4 \ *$
 - $p1 \ p2 \ -$ è una notazione polacca

Aritmetica postfissa /2

- Le regole per calcolare l'espressione postfissa
 - scandisci l'espressione da sinistra a destra fino a che raggiungi il primo operatore
 - applica l'operatore ai due operandi che sono alla sua sinistra, ottieni il risultato che sostituirai nell'espressione al posto di operandi e operatore
-
- **35 17 40 9 - * + 7 -**
 - **35 17 31 * + 7 -**
 - **35 527 + 7 -**
 - **562 7 -**
 - **555**

Algoritmo per valutare una espressione in notazione polacca

- Scandire l'espressione da sinistra a destra
 - appena è raggiunto un operando in-pila l'operando nella pila degli operandi
 - appena è raggiunto un operatore rimuovi dalla pila i primi due operandi, applica l'operatore a questi e in-pila il risultato in cima alla pila



Algoritmo per valutare una espressione in notazione polacca /2

- Il programma seguente realizza l'algoritmo, ma rimanda alcune funzionalità (verifica che il simbolo sia un numero) ad un ulteriore sforzo di programmazione. Si dispone di una coda di simboli e si usa una pila per la valutazione della notazione polacca mediante pila di numeri reali.

```
double valuta_polacca (coda<char> post)
    creapila(val)
    while not codavuota (post) do
        t = leggicoda(post)
        canccoda(post)
        if ( t è un numero) then
            inpila (t, val)
        else
            numtop = leggipila (val)
            fuoripila(val)
            numsuc = leggipila (val)
            fuoripila(val)
            if (t == +) then ris = numsuc + numtop
            if (t == -) then ris = numsuc - numtop
            inpila (ris, val)
    return leggipila(val)
```

Notazione postfissa

- Disporre di un modo per calcolare espressioni postfisse serve a poco se non si dispone di un metodo che sia in grado di convertire la lista di simboli di una espressione infissa in quelli della corrispondente espressione postfissa.
- In questo caso è conveniente utilizzare una coda come risultato della conversione.

Conversione da infissa a postfissa

- Utilizziamo una pila per memorizzare i simboli degli operatori “in sospeso” e una coda per immagazzinare la espressione postfissa che viene costruita man mano.
- Ad ogni operatore nella espressione infissa viene assegnato un ordine di precedenza (moltiplicazione e divisione precedenza massima).
- Attribuiamo alla parentesi (la precedenza minima.

Conversione da infissa a postfissa /2

Simboli	Pila	Coda
35 + 17 * (40 - 9) - 7	<>	<>
+ 17 * (40 - 9) - 7	<>	35
17 * (40 - 9) - 7	+	35
* (40 - 9) - 7	+	17 35
(40 - 9) - 7	+ *	17 35
40 - 9) - 7	+ * (17 35
- 9) - 7	+ * (40 17 35
9) - 7	+ * (-	40 17 35
) - 7	+ * (-	9 40 17 35
- 7	+ * (- 9 40 17 35
7	-	+*- 9 40 17 35
	-	7+*- 9 40 17 35
	<>	-7+*-9 40 17 35

Conversione da infissa a postfissa /3

```
trasferisci(pila per riferimento s, coda per riferimento c)
  top_elem = leggipila(s)
  fuoripila(s)
  incoda(top_elem, c)

converti(lista infissa, coda per riferimento coda_post)
  creapila(pila_op)
  creacoda(coda_post)
  while not listavuota(infissa) do
    t = leggilista(infissa)
    if (t è un numero ) then incoda(t, coda_post)
    elseif pilavuota(pila_op) then inpila(t, pila_op)
    elseif (t = "(" ) then inpila(t, pila_op)
    elseif (t = ")" ) then top_elem = leggipila(pila_op)
      while (top_elem not = "(" ) do
        trasferisci(pila_op, coda_post)
        fuoripila(pila_op)
    else
      while (priorità t <= priorità top_elem ) do
        trasferisci(pila_op, coda_post)
        inpila(t, pila_op)
  while not finepila(pila_op) trasferisci(pila_op, coda_post)
```