Esercizio svolto 2.1 (libro di testo)

Un intervallo di interi $[a, b]$ rappresenta $\{a, a+1, \ldots, b\}$.

Per esempio $[15, 17] = \{15, 16, 17\}$. L'unione di intervalli può essere vista come l'unione dei corrispondenti insiemi di interi.

Per esempio l'unione di $[8, 9]$, $[10, 10]$, $[15, 17]$ $[16, 19]$ e $[18, 18] = \{8, 9, 10, 15, 16, 17, 18, 19\}$

L'unione di intervalli può essere sempre rappresentata in modo UNIVOCO come una lista ordinata L di intervalli DISGIUNTI e NON ADIACENTI.

Per esempio l'unione di $[8, 9]$, $[10, 10]$, $[15, 17]$, $[16, 19]$ e $[18, 18]$ è rappresentata da $L = [8, 10], [15, 19]$.

(disgiunti e non adiacenti)

Problema: dati in ingresso $2n$ interi $a_0, b_0, a_1, b_1, \ldots a_{n-1}, b_{n-1}$ interpretati come n intervalli di interi $[a_0, b_0], [a_1, b_1] \ldots [a_{n-1}, b_{n-1}]$ si vuole stampare la lista L che rappresenta la loro unione.

Prima soluzione di complessità $O(n^2)$.

Schema induttivo. Passo base $i = 0$: $L = [a_0, b_0]$.

Passo induttivo $i > 0$. Elabora $[a_i, b_i]$. Sia $L = [c_0, d_0], \ldots [c_{k-1}, d_{k-1}]$.

Due casi vengono considerati

- se $[a_i, b_i]$ è CONTENUTO in un intervallo di L, salta al passo $i+1$.

- altrimenti
  - ELIMINA da $\mathcal{L}$ tutti gli intervalli $[c_j, d_j] \subseteq [a_i, b_i]$ ottenendo la lista $\mathcal{L}'$
  - trova, se esiste, l'unico intervallo $[c', d']$ in $\mathcal{L}'$ che interseca $[a_i\ b_i]$ oppure è adiacente ad esso, dove $c' < a_i$; altrimenti poni $c' = a_i$

    $[c',\ d']$ $\qquad$ $[c',\ d']$
    $[a_i,\ b_i]$ $\qquad$ $[a_i,\ b_i]$

  - trova, se esiste, l'unico intervallo $[c'', d'']$ in $\mathcal{L}'$ che interseca $[a_i, b_i]$ oppure è adiacente ad esso, dove $b_i < d''$; altrimenti poni $d'' = b_i$

    $[c'',\ d'']$ $\qquad$ $[c'',\ d'']$
    $[a_i,\ b_i]$ $\qquad$ $[a_i,\ b_i]$

Se non ci sono intersezioni, inserisce $[a_i, b_i]$.
Se ci sono più inserire: $[c', b_i]$ oppure $[a_i, d'']$ oppure $[c', d'']$

  - AGGIORNA $\mathcal{L} = \mathcal{L}' \cup [c', d'']$

Infine, restituisci $\mathcal{L} = [c_0, d_0], [c_1, d_1], \ldots, [c_{u-1}, d_{u-1}]$, la lista ottenuta dopo il passo $i = u - 1$.

Ogni passo costa $O(u)$.

Possiamo ridurre il costo computazionale a $O(n \log n)$.

Abbiamo visto fino ad ora solo algoritmi di ordinamento di complessità $O(n^2)$. Vediamo se ne esistono di complessità $O(n \log n)$. Immaginiamo di usarne uno di essi.

Possiamo procedere come segue:

- Ordiniamo gli intervalli $[a_0, b_0]$ $[a_1, b_1]$ ... $[a_{n-1}, b_{n-1}]$ in base agli estremi sinistri (ipotizzando che siano tutti distinti)
- Scandiamo gli intervalli secondo l'ordinamento ottenuto e li "fondiamo" usando una pila.

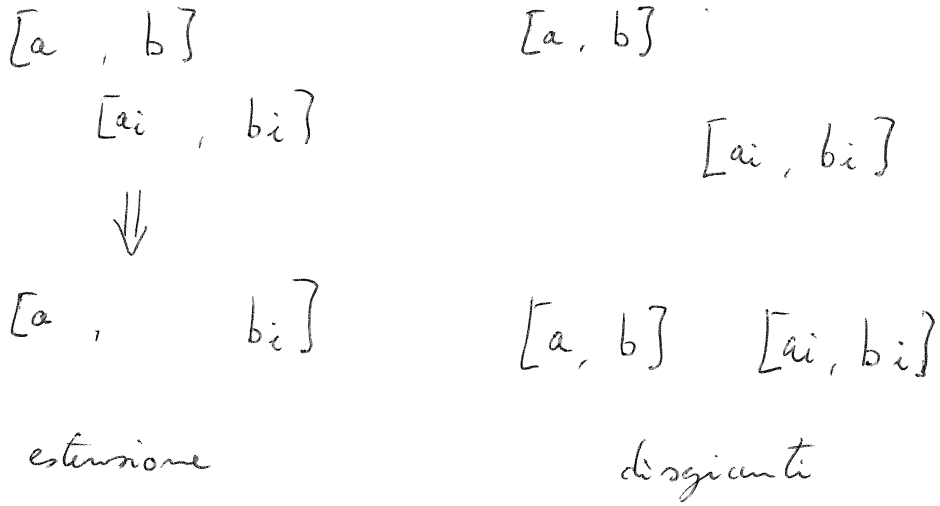
  Passo $i = 0$. Metti $[a_0, b_0]$ nella pila.

  Passo $i > 0$. Sieno $[a, b]$ l'intervallo in cima alla pila e $[a_i, b_i]$ l'intervallo corrente.

  - se $a_i \leq b + 1$, fai POP di $[a, b]$ e PUSH di $[a, \max\{b, b_i\}]$. (Poiché gli intervalli sono ordinati è sicuramente $a < a_i$).
  - se $a_i > b + 1$, fai PUSH di $[a_i, b_i]$

Infine, stampa il contenuto della pila alla fine dell'ultimo passo.

Nota: a seguito dell'ordinamento sono possibili solo "due configurazioni" al passo $i > 0$



Code

Coda: collezione di elementi in cui le operazioni disponibili sono definite da una specifica politica di accesso.

Estraiamo il primo elemento inserito nella coda "in testa" alla coda, mentre inseriamo un nuovo elemento "in fondo" alla coda.

Una politica d'accesso di questo tipo viene detta FIFO (First In First Out).

Le operazioni di inserimento ed estrazione richiedono tempo costante.

ENQUEUE: inserisce un elemento in fondo alla coda

DEQUEUE: estrae l'elemento dalla testa della coda

FIRST: restituisce l'informazione contenuta nell'elemento in testa alla coda

EMPTY: verifica se la coda è vuota.

Mentre in una pila c'è un unico punto di accesso su cui le operazioni vanno ad incidere, nel caso di una coda ne esistono due. FIRST e DEQUEUE operano "in testa", ENQUEUE opera "in fondo".

Implementazione tramite Array

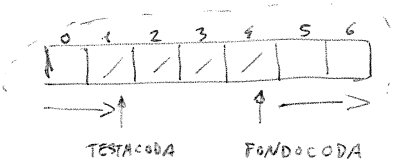
Gli elementi della coda sono memorizzati in sequenza nell'array, a partire dalla locazione associata all'inizio della coda, inserendoli man mano nella prima locazione disponibile.

La fine della coda corrisponde all'ultimo elemento inserito.

TESTACODA = indice contenente la locazione del primo elemento

FONDOCODA = indice contenente la locazione dell'ultimo elemento

I due indici vengono gestiti <u>in modo circolare</u>: <u>raggiunta una estremità</u>, <u>si riparte dall'altra</u>.




```
ENQUEUE (x):
VerificaRaddoppio();
carCoda = carCoda + 1;
fondoCoda = (fondoCoda + 1) % lunghezzaArray;
codaArray [fondoCoda] = x;
```

```
EMPTY ():
RETURN (carCoda == 0);
```

```
FIRST ():
IF (!EMPTY()) RETURN codaArray[testaCoda];
```

Variabili

<u>carCoda</u> = # elementi nella coda, inizializzata a 0.

<u>testaCoda</u> = inizializzata a 0

<u>fondoCoda</u> = inizializzata a -1

<u>lunghezzaArray</u> = memorizza la lunghezza dell'array codaArray

<u>codaArray</u> = array dinamico che memorizza gli elementi della coda

```
DEQUEUE ():
IF (!EMPTY()) {
  carCoda = carCoda - 1;
  x = codaArray [testaCoda];
  testaCoda = (testaCoda + 1) % lunghezzaArray;
  VerificaDimezzamento ();
  RETURN x;
}
```

Come anticipato gli incrementi di TESTACODA e FONDOCODA sono circolari. Pertanto si usa il modulo della divisione intera, assumendo che la lunghezza dell'array sia memorizzata nella variabile lunghezzaArray

Nota: le operazioni che implementano l'array dinamico

VerificaRaddoppio e VerificaDimezzamento

riallocano ordinatamente gli elementi della coda nelle prime celle dell'array e pongono testaCoda a 0 e fondoCoda a carCoda - 1

ContoModulare

$0+1 \bmod 7 = 1$

$1+1 \bmod 7 = 2$

$2+1 \bmod 7 = 3$

$3+1 \bmod 7 = 4$

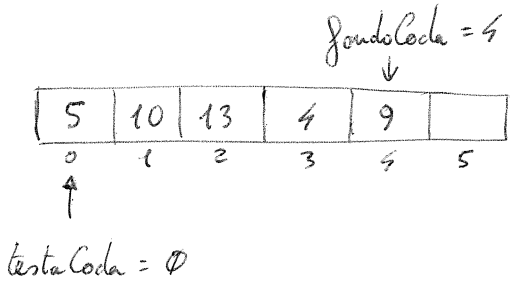
fondoCoda = (fondoCoda + 1) % lunghezzaArray

$4+1 \bmod 7 = 5$

$5+1 \bmod 7 = 6$

$6+1 \bmod 7 = 0$

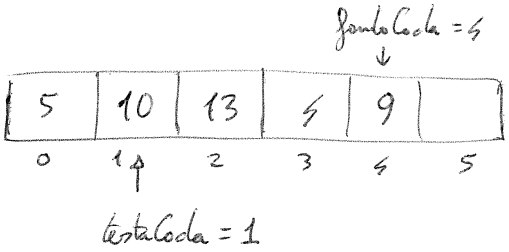
Esempio



L'operazione FIRST() restituisce il valore 5.

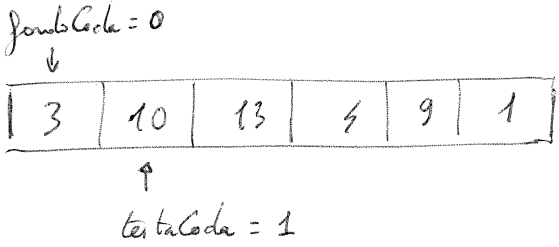
L'operazione EMPTY() restituisce il valore FALSE

L'esecuzione di DEQUEUE() trasforma la coda come segue:

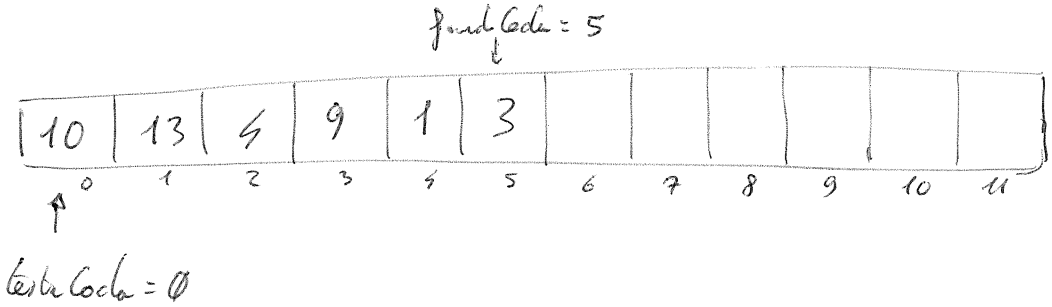


(l'elemento non viene fisicamente rimosso ma è "fuori coda")

Dopo le operazioni ENQUEUE(1), ENQUEUE(3), la coda appare come segue



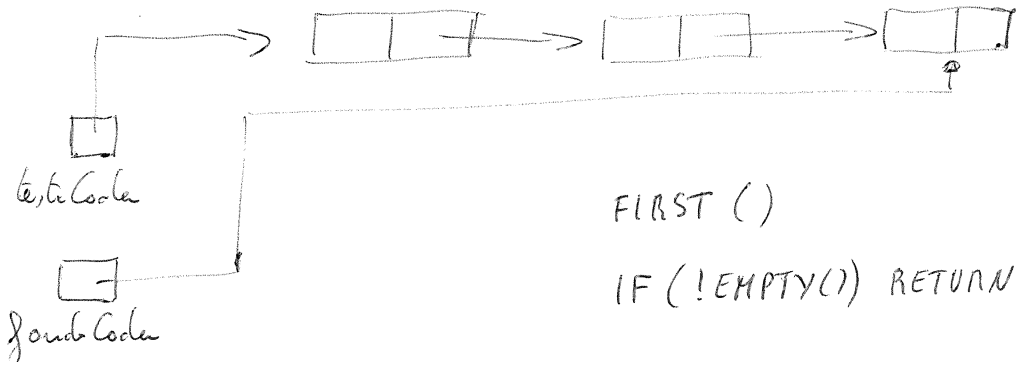
L'ulteriore inserimento tramite ENQUEUE(11) causa il ridimensionamento



Implementazione tramite una lista

Sequenza di nodi concatenati e ordinati in modo crescente secondo l'istante di inserimento (meno recente, tempo minore).

Il primo nodo corrisponde alla testa ed è il nodo da estrarre nel caso di una DEQUEUE(); l'ultimo nodo corrisponde al fondo ed è il nodo al quale concatenare un nuovo nodo tramite una ENQUEUE().



È corretta l'implementazione?

```
FIRST()
IF (!EMPTY()) RETURN testaCoda.dato;
```

```
ENQUEUE (x)
p = NuovoNodo(x);
p.succ = NULL;
IF (EMPTY()) {
  testaCoda = p;
  fondoCoda = p;
}
ELSE {
  fondoCoda.succ = p;
  fondoCoda = p;
}
```

PRIMO NODO IN CODA

```
EMPTY()
RETURN (testaCoda == NULL);
```

```
DEQUEUE()
IF (!EMPTY) {
  x = testaCoda.dato;
  IF (testaCoda == fondoCoda) {
    testaCoda = NULL;
    fondoCoda = NULL;
  }
  ELSE {
    testaCoda = testaCoda.succ;
  }
RETURN x;
}
```

ULTIMO NODO IN CODA