Algoritmi e Strutture Dati

Esercitazione 2

Domenico Fabio Savo

Il tipo di dato Coda

tipo `Coda`:
dati:
una sequenza S di n elementi.

operazioni:

`isEmpty`$() \rightarrow result$
restituisce `true` se S è vuota, e `false` altrimenti.

`enqueue`$(elem\ e)$
aggiunge e come ultimo elemento di S.

`dequeue`$() \rightarrow elem$
toglie da S il primo elemento e lo restituisce.

`first`$() \rightarrow elem$
restituisce il primo elemento di S (senza toglierlo da S).

Coda: esercizio

Esercizio:

Scrivere lo pseudocodice che implementa le operazioni definite sul tipo di dato CODA (*isEmpty*, *enqueue*, *dequeue*, *first*), nel caso in cui questo è implementato tramite un array.

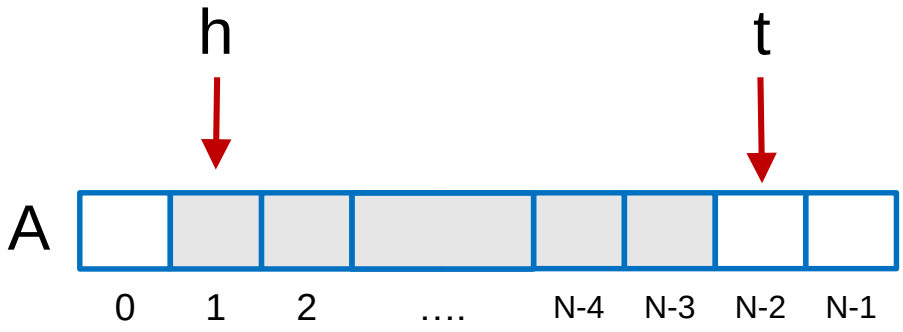
Coda: implementazione mediante lista semplice

La coda **C** sarà rappresentata come una tripla (**A**, **h, t**) dove:

- **A** è un array di dimensione N
- **h** è l'indice del primo elemento della coda
- **t** è l'indice della posizione successiva alla posizione dell'ultimo elemento della coda.

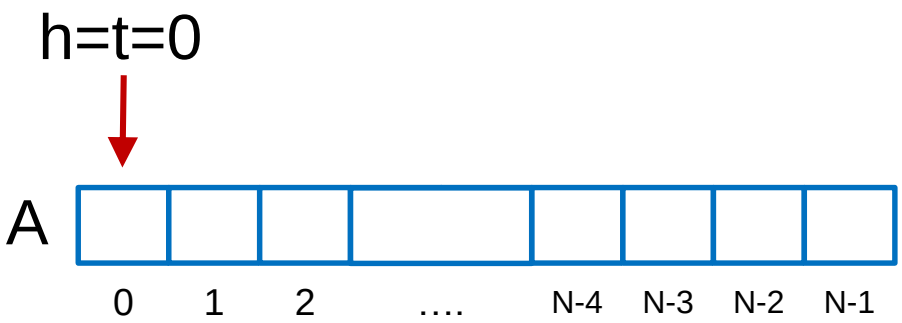
- L'array viene utilizzato in modo "**circolare**"

Coda: implementazione mediante lista semplice

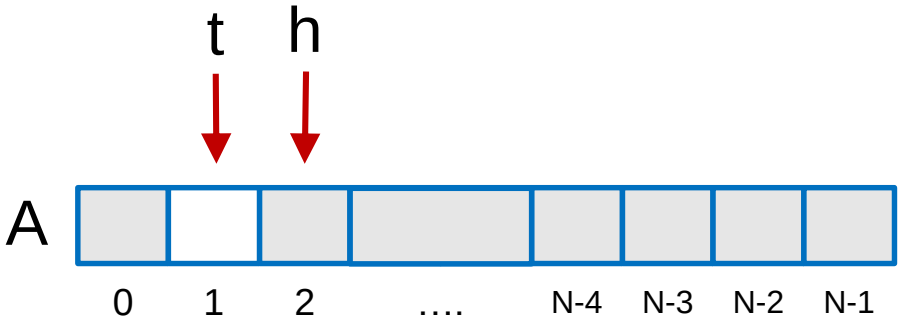


h punta al primo elemento della coda (testa della coda) mentre **t** punta alla cella successiva a quella dell'ultimo elemento della coda. **t** punta sempre ad una cella vuota. Un array di dimensione **N** potrà essere usato per realizzare una coda di dimensione **N-1**

Coda vuota e coda piena



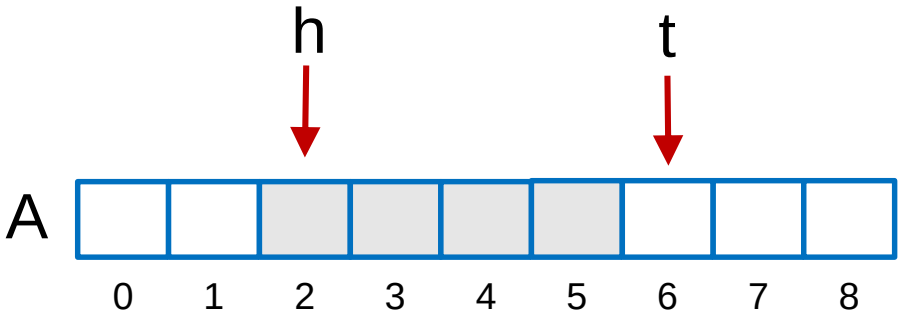
- **Coda vuota: h** e **t** puntano entrambi alla medesima posizione**.** Inizialmente puntano entrambi alla posizione **0**



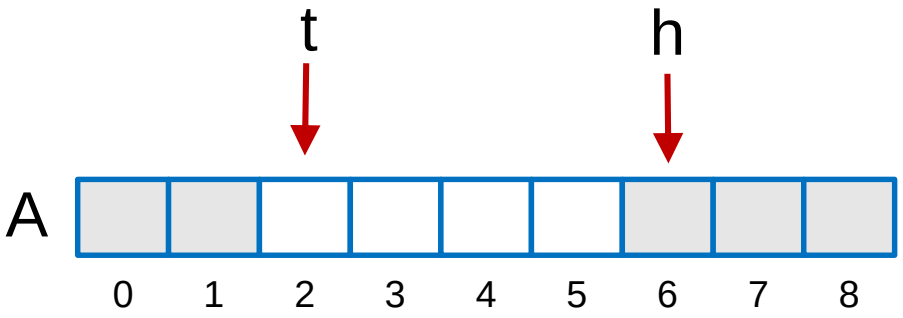
- **Coda piena:** le celle comprese tra quelle puntate da **h** e **t** sono **N-1**. Dato che l'array è ora trattato in modo "circolare", come possiamo contarle?

Coda vuota e coda piena (2)

Il numero di celle occupate è pari ad **(N + t – h) mod N**



(N + t – h) mod N =
(9 + 6 – 2) mod 9 =
13 mod 9 = 4



(N + t – h) mod N =
(9 + 2 – 6) mod 9 =
5 mod 9 = 5

Coda: isEmpty

- La coda è vuota quando **h** è pari a **t**.

isEmpty() → *true* or *false*

if (t = h) **then return** *true*

else return *false*

Coda: enqueue

```
enqueue(elem e)
        if((N+t-h mod N) = N-1)
                then "coda piena!"
        else { A[t] ← e;
                t ← (t+1) mod N;}
```

- Inseriamo l'elemento in posizione **t**.
- La funzione **enequeue** incrementa il puntatore **t.** La funzione modulo garantisce che **t** non superi mai **N-1**.

Coda: dequeue

```
dequeue() → elem e
        if (C.isEmpty()) then return null;
        else { x ← A[h];
                h ← (h+1) mod N;
                return x; }
```

- Restituiamo l'elemento in posizione **h**.
- La funzione **dequeue** incrementa il puntatore **h.** La funzione modulo garantisce che **h** non superi mai **N-1**.

Coda: first

first() → elem e

if (C.isEmpty()) **then return** *null*;

else return A[h];

- Il primo elemento è nella posizione A[h].

Calcolo profondità di un albero

Il seguente algoritmo mostra un applicazione della visita in profondità per calcolare il numero di livelli in un albero binario.

```
algoritmo profondità(nodo r) →intero
    if (r = null) return 0;
    sin ← profondita(figlio sx di r);
    des ← profondita(figlio dx di r);
    return 1 + max(sin,des);
```

Calcolo numero foglie di un albero

ESERCIZIO: prendendo spunto dall'algoritmo **profondita(nodo r)**, scrivere l'algoritmo **numFoglie** che calcola il numero di foglie (i.e., nodi senza figli) di un albero binario.

```
algoritmo profondità(nodo r) →intero
    if (r = null) return 0;
    sin ← profondita(figlio sx di r);
    des ← profondita(figlio dx di r);
    return 1 + max(sin,des);
```

Calcolo numero foglie di un albero

ESERCIZIO: prendendo spunto dall'algoritmo **profondita(nodo r)**, scrivere l'algoritmo **numFoglie** che calcola il numero di foglie (i.e., nodi senza figli) di un albero binario.

SOLUZIONE

```
algoritmo numFoglie(nodo r) →intero
    if (r = null) then return 0;
    if (figlio sx di r = null) && (figlio dx di r = null) then return 1;
    sin  ←  numFoglie (figlio sx di r);
    des  ←  numFoglie (figlio dx di r);
    return sin + des;
```