Le Code*

Maurizio Palesi†

Sommario

bla bla bla.

Indice

1 Definizione di Coda

La *Coda* è una particolare lista in cui l'inserimento è consentito solo ad un estremo, detto *rear*, e la cancellazione è consentita solo all'altro estremo, detto *front*. Questa gestione è anche chiamata FIFO (First In First Out), in quanto il primo elemento inserito è il primo ad essere estratto.

Sia data una coda composta dagli elementi $a_1, a_2, a_3, \ldots, a_n$ e si supponga che il front sia rappresentata dalla posizione del primo elemento a_1 mentre il rear dalla

*Questo documento è stato redatto utilizzando come rifermiento l'ADT Coda realizzato dal Prof. Salvatore Cavalieri `http://www.diit.unict.it/users/scava`.

†Maurizio Palesi, DIIT—Università di Catania, Viale Andrea Doria 6, 95125 Catania, email:`mpalesi@diit.unict.it`, web:`http://www.diit.unict.it/users/mpalesi`, tel.0957382353, fax.095338280.

posizione dell'ultimo elemento a_n. L'inserimento di un elemento x produce la lista $a_1, a_2, a_3, \ldots, a_n, x$. Il nuovo rear diviene la posizione di x. La cancellazione dell'elemento di posizione front, produce la lista $a_2, a_3, \ldots, a_n, x$. Il nuovo front è relativo all'elemento a2.

2 Operazioni Primitive sulla Coda

La Coda è caratterizzata da alcune operazioni fondamentali. Sia Q una coda, ed x un generico valore. Le seguenti operazioni primitive vengono definite:

- $MakeNull(Q)$: La funzione restituisce una Coda vuota.
- $Empty(Q)$: La funzione restituisce 1 se la Coda è vuota.
- $Full(Q)$: La funzione restituisce 1 se la Coda è piena.
- $EndQueue(Q, x)$: Questa funzione inserisce l'elemento x nella posizione rappresentata dal rear. Se Q è la Coda $a_1, a_2, \ldots, a_n$ il rear è rappresentato dalla posizione dell'ultimo elemento a_n, l'inserimento dell'elemento x avviene in posizione successiva a tale elemento, e la funzione $EndQueue(Q, x)$ restituisce la nuova lista $a_1, a_2, \ldots, a_n, x$.
- $DeQueue(Q)$: Questa funzione elimina l'elemento della coda in posizione front. Se Q è la Coda $a_1, a_2, \ldots, a_n$ e il front è rappresentato dalla posizione del primo elemento, la funzione $DeQueue(Q)$ restituisce la nuova lista $a_2, a_3, \ldots, a_n$ e il nuovo front è rappresentata dalla posizione dell'elemento a_2.
- $Front(Q)$: La funzione restituisce l'elemento della coda in posizione front.

3 Rappresentazione di una Coda

Verranno considerate due rappresentazioni: con vettori (rappresentazione sequenziale) e con puntatori (rappresentazione collegata).

3.1 Rappresentazione Sequenziale

La rappresentazione con vettori può essere realizzata utilizzando due indici che chiameremo *front* e *rear*. front e rear, che indicano rispettivamente la posizione del primo elemento della coda e dell'ultimo elemento della coda. Inizialmente, quando la coda è vuota si assume che front=rear=-1. Quando viene inserito il primo elemento, rear viene posto a 0 e l'elemento viene inserito nel vettore nella componente di indice rear (ossia 0). Anche front viene assegnato a 0, in quanto è necessario rappresentare la presenza

dell'unico elemento in coda. Quando un nuovo elemento deve essere inserito successivamente, rear viene incrementato di 1 e l'elemento viene inserito nella nuova posizione fornita da rear. L'incremento di rear è circolare cioè:

$$rear = (rear + 1)\%N$$

Dove con N si è indicata la dimensione del vettore.

L'inserimento di un elemento è possibile solo se la coda non è piena. La condizione di coda piena si ha quando:

$$front = (rear + 1)\%N$$

Per quanto riguarda la cancellazione dell'elemento di posizione front, essa viene semplicemente realizzata incrementando la variabile front. Dopo aver effettuato la cancellazione, è ovviamente necessario verificare che la coda non diventi vuota, perché in tal caso bisogna aggiornare la variabile rear. Se la coda diviene vuota, sia front che rear divengono pari a -1. La condizione di coda vuota viene controllata prima della cancellazione dell'elemento front. Se front e rear sono coincidenti, significa che la coda possiede un solo elemento, cioè quello che deve essere eliminato. Ciò significa che dopo l'inserimento la coda diverrà vuota. L'incremento della variabile front deve avvenire in modo circolare, analogamente a quanto visto per la variabile rear. Analogamente a quanto visto per la variabile rear, l'incremento della variabile front deve avvenire nella seguente maniera:

$$front = (front + 1)\%N$$

Nelle sottosezioni successive implementeremo le primitive di gestione prima descritte utilizzando il linguaggio C. Prima di tutto definiamo il tipo coda (Queue).

```
#define N 10

typedef int BaseType;

struct Queue {
   BaseType info[N];
   int      front, rear;
};
```

3.1.1 MakeNull(Q)

La funzione `MakeNull(Q)` restituisce una Coda vuota.

```
void MakeNull(struct Queue *q)
{
  q->front = -1;
  q->rear  = -1;
}
```

3.1.2 Empty(Q)

La funzione `Empty(Q)` restituisce 1 se la Coda è vuota.

```
boolean Empty(struct Queue q)
{
  return(q.front == -1);
}
```

3.1.3 Full(Q)

La funzione `Full(Q)` è necessaria perchè il vettore potrebbe saturarsi.

```
int Full(struct Queue q)
{
  return (q.front == ((q.rear+1)%N));
}
```

3.1.4 EndQueue(Q,x)

La funzione `EndQueue(Q,x)` inserisce l'elemento x nella posizione rappresentata dal rear.

```
void ENQUEUE(struct Queue q, BaseType x)
{
 if (!Full(q))
  {
    q.rear = (q.rear + 1) % N;
    q.info[q.rear] = x;
    if (q.front == -1)
      q.front = q.rear;
  }
}
```

3.1.5 DeQueue(Q)

La funzione `DeQueue(Q)` elimina l'elemento della coda in posizione front.

```
void Dequeue(struct Queue q)
{
  if (!Empty(q))
   {
     if (q.front == q.rear)
       MakeNull(q);
     else
       q.front = (q.front + 1) % N;
```

```
    }
}
```

3.1.6 Front(Q)

La funzione `Front(Q)` restituisce l'elemento della coda in posizione front.

```
BaseType Front(struct Queue q)
{
  if (!Empty(q))
        return(q.info[q.front]);
}
```

3.2 Rappresentazione Collegata con Puntatori

La rappresentazione collegata con puntatori di una Coda può avvenire secondo quanto descritto di seguito. Le variabili front e rear rappresentano i puntatori al primo e all'ultimo elemento della coda, rispettivamente. Adesso l'inserimento nella posizione $End(Q)$, può essere agevolmente realizzato tramite la conoscenza del puntatore rear all'ultimo elemento della coda. In tal caso la complessità computazionale non è più proporzionale al numero di elementi in coda, ma è costante. La precedente rappresentazione può essere definita in C nella seguente maniera.

```
struct Nodo
{
  BaseType     info;
  struct Nodo *next;
};

struct Queue
{
  struct Nodo *front;
  struct Nodo *rear;
};
```

La condizione di coda vuota è quando il puntatore al primo elemento della coda è NULL. La costante `CODAVUOTA` può essere rappresentata da:

```
#define CODAVUOTA NULL
```

Nelle sottosezioni successive implementeremo le primitive di gestione prima descritte utilizzando il linguaggio C. Prima di tutto definiamo il tipo coda (Queue).

3.2.1 MakeNull(Q)

La funzione `MakeNull(Q)` restituisce una Coda vuota.

```
void MakeNull(struct Queue *q)
{
  q->front = CODAVUOTA;
  q->rear = CODAVUOTA;
}
```

3.2.2 Empty(Q)

La funzione `EMpty(Q)` restituisce 1 se la Coda è vuota.

```
ibt Empty(struct Queue q)
{
 return (q.front == CODAVUOTA);
}
```

3.2.3 Full(Q)

La funzionr `Full(Q)` restituisce sempre 0.

```
int Full(struct Queue q)
{
 return 0;;
}
```

3.2.4 EnQueue(Q,x)

La funzione `EnQueue(Q,x)` inserisce l'elemento `x` nella posizione rappresentata dal rear.

```
void EnQueue(struct Queue *q, BaseType x)
{
  struct Nodo *temp;

  temp = malloc(sizeof(struct Nodo));
  temp->info = x;
  temp->next = CODAVUOTA;
  if (Empty(*q))
      q->front = q->rear = temp;
  else
   {
      q->rear->next = temp;
      q->rear = temp;
```

```
    }
}
```

3.2.5 DeQueue(Q)

La funzione `DeQueue(Q)` elimina l'elemento della coda in posizione front.

```
void DeQueue(struct Queue *q)
{
  struct Nodo *temp;

  if (!Empty(*q))
   {
      temp = q->front->next;
      free(q->front);
      q->front = temp;
      if (q->front == CODAVUOTA)
         q->rear = CODAVUOTA;
   }
}
```

3.2.6 Front(Q)

La funzione `Front(Q)` restituisce l'elemento della coda in posizione front.

```
BaseType FRONT(struct Queue q)
{
  if (!Empty(q))
    return(q.front->info);
}
```

4 Un Esempio Applicativo

Il seguente programma gestisce una coda contenente elementi di tipo intero. Si suppone l'esistenza di un file di nome `"coda.h"` contenente l'implementazione della coda (con vettore o con puntatore) e la definizione del BaseType, che, come detto, viene scelto intero. Dunque, il file `coda.h` conterrà la seguente definizione:

```
typedef int BaseType;
```

Come si vede, il seguente programma include il file `"coda.h"`, in cui viene specificata l'implementazione della coda. Cambiando tale file può essere cambiata l'implementazione della coda, senza apportare alcuna modifica al seguente programma. Dunque,

il seguente programma prescinde completamente dalla realizzazione della coda, e può essere utilizzato sia per la rappresentazione sequenziale sia collegata.

```
#include <stdio.h>
#include <stdlib.h>
#include "coda.h"

void main()
{
  int           s;
  BaseType      elem;
  struct Queue  c;

  MakeNull(&c);

  do {
    printf("Menu di Operazioni\n");
    printf("1-Inserimento\n");
    printf("2-Cancellazione\n");
    printf("3-Inizializza\n");
    printf("4-Ispeziona Elemento in Testa\n");
    printf("5-Fine\n");
    printf("Inserisci la scelta ");
    scanf("%d", &s);

    switch (s)
      {
      case 1:
        if (!Full(c))
          {
            printf("Inserisci Elemento ");
            scanf("%d", &elem);
            EnQueue(&c, elem);
          }
        else
          printf("Coda Piena\n");
        break;

      case 2 :
        if (Empty(c))
          printf("Coda Vuota\n");
        else
```

```
            {
              DeQueue(c);
              printf("Primo Elemento Estratto\n");
            }
          break;

        case 3 :
          while (!Empty(c))
            DeQueue(c);
          break;

        case 4 :
          if (Empty(c))
            printf("Coda Vuota\n");
          else
            printf("Elemento in Cima = %d", Front(c));
          break;

        }
    } while (s < 5);
}
```