14.12 Gestione di una coda mediante liste

Consideriamo una lista che contiene gli elementi della coda in sequenza. Il puntatore `puntTesta` si riferisce alla posizione del primo elemento che verrà estratto (Figura 14.17).

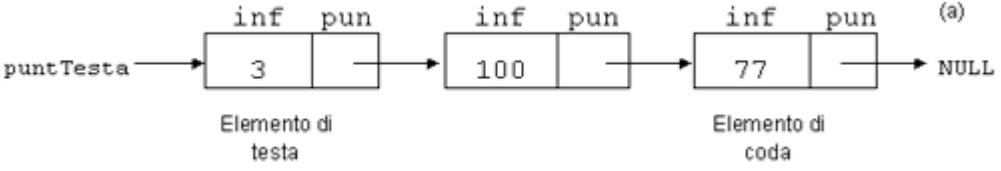

Figura 14.17 Coda implementata mediante una lista

Dunque nel caso dell'operazione di eliminazione si deve assegnare a `puntTesta` il valore del campo puntatore del primo elemento:

```
puntTesta = puntTesta->pun;
```

In questo modo viene scavalcato e dunque eliminato il primo elemento. Naturalmente si dovrebbe prevedere il rilascio della memoria corrispondente. Nel caso di inserimento si deve scorrere (scandire), mediante un puntatore ausiliario, tutta la lista, dopo di che si può effettuare l'inserimento in coda:

```
paus = puntTesta;
if(paus!=NULL) {          /* se non è vuota si visita la lista */
  while(paus->pun!=NULL) do
   paus = paus->pun;

   /* creazione del nuovo elemento      */
   paus->pun = (struct elemento *)malloc(sizeof(struct elemento));
   paus = paus->pun;      /* paus ora punta al nuovo elemento  */
   paus->inf = ele;       /* si inserisce l'informazione       */
   paus->pun = NULL;      /* si immette la marca di fine lista */
}
else {
  /* creazione primo elemento */
  puntTesta = (struct elemento *)malloc(sizeof(struct elemento));
  puntTesta->inf = ele;         /* inserisce l'informazione */
  puntTesta->pun = NULL;       /* marca di fine lista      */
}
```

Questo implica che per effettuare un inserimento si deve scandire la lista, il che è arduo se la coda contiene centinaia o migliaia di elementi. Per rendere più veloce l'operazione di inserimento si utilizza un ulteriore puntatore `punt_coda` che fa riferimento alla coda della sequenza (Figura 14.18).

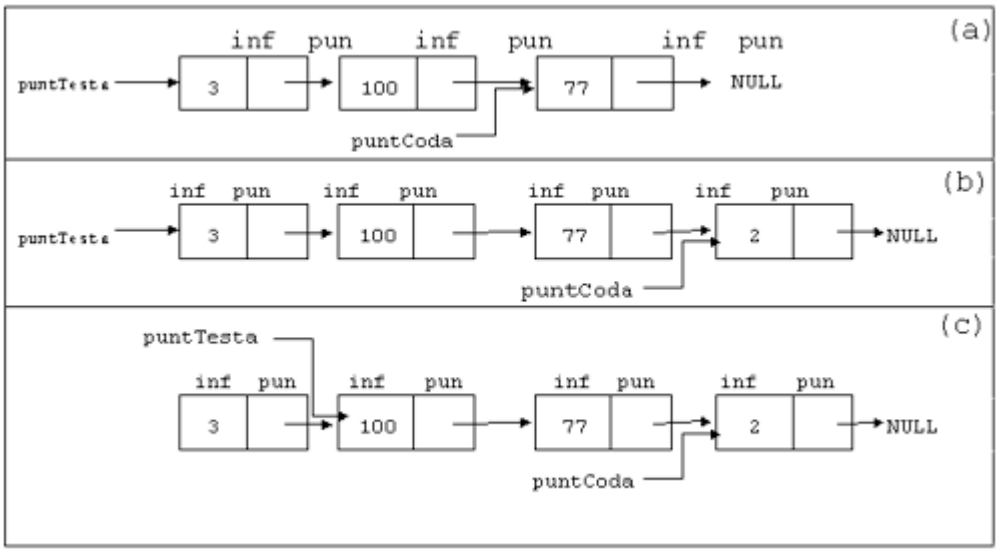

Figura 14.18 a) Implementazione di una coda mediante una lista con due puntatori. b) Inserimento di un elemento nella coda. c) Eliminazione di un elemento dalla coda

In questo caso l'operazione di inserimento è quella di Figura 14.18b; non è necessario scandire la lista poiché abbiamo attivato un puntatore alla coda:

```
/* creazione del nuovo elemento      */
puntCoda->pun = (struct elemento *)malloc(sizeof(struct elemento));
puntCoda = puntCoda->pun;  /* punt_coda al nuovo elemento */
puntCoda->inf = ele;       /* si inserisce l'informazione */
puntCoda->pun = NULL;      /* si immette la marca di fine lista */
```

Invece l'eliminazione è identica: è quella dell'implementazione precedente, che prevedeva un solo puntatore (Figura 14.18c).
Un altro metodo per gestire una coda prevede l'utilizzazione di una lista circolare nella quale l'elemento in coda punta a quello in testa (Figura 14.19a).

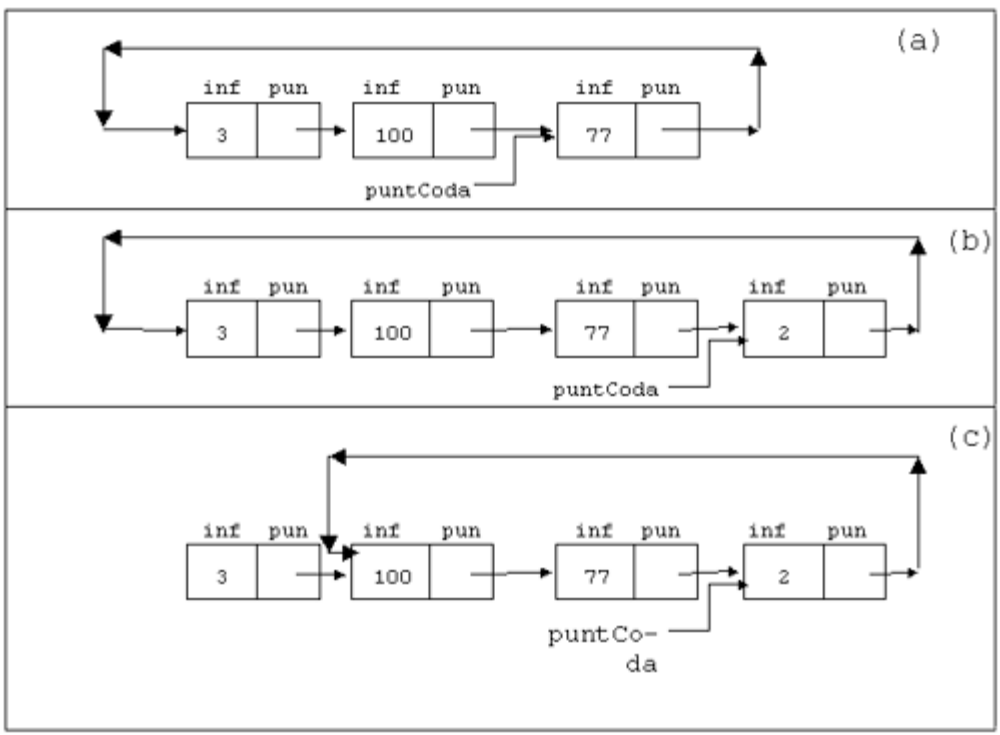

Figura 14.19 a) Implementazione di una coda mediante una lista circolare. b) Inserimento di un elemento nella coda. c) Eliminazione di un elemento dalla coda

In questo caso è necessario mantenere il solo puntatore alla coda `punt_coda`, per mezzo del quale possono essere effettuate le operazioni di inserimento (Figura 14.19b) ed eliminazione (Figura 14.19c).

```
/* INSERIMENTO creazione del nuovo elemento */
paus = (struct elemento *)malloc(sizeof(struct elemento));
paus->inf = ele;            /* inserimento dell'informazione */
paus->pun = puntCoda->pun;  /* nuovo elemento alla testa */
puntCoda->pun = paus;       /* punt_coda al nuovo elemento */
puntCoda = puntCoda->pun;   /* attualizzazione puntCoda */

/* ELIMINAZIONE */
paus = puntCoda->pun;       /* salvataggio puntatore alla testa */
puntCoda->pun = puntCoda->pun->pun;  /* eliminazione */

  free(paus);                          /* rilascio memoria */
```