AULA 22

LISTAS LINEARES DUPLAMENTE ENCADEADAS

Quando trabalhamos com listas lineares encadeadas, muitas vezes precisamos de manter um ponteiro para uma célula e também para a célula anterior para poder realizar algumas operações sobre a lista, evitando percursos adicionais desnecessários. Esse é o caso, por exemplo, da busca e da inserção. No entanto, algumas vezes isso não é suficiente, já que podemos precisar percorrer a lista linear nos dois sentidos. Neste caso, para solucionar eficientemente esse problema, adicionamos um novo campo ponteiro nas células da lista, que aponta para a célula anterior da lista. O gasto de memória imposto pela adição deste novo campo se justifica pela economia em não ter de reprocessar a lista linear inteira nas operações de interesse. Esta aula é baseada nas referências [7, 13].

22.1 Definição

As células de uma lista linear duplamente encadeada são ligadas por ponteiros que indicam a posição da célula anterior e da próxima célula da lista. Assim, um outro campo é acrescentado à cada célula da lista indicando, além do endereço da próxima célula, o endereço da célula anterior da lista. Veja a figura 22.1.

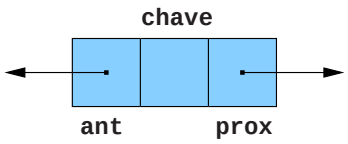

Figura 22.1: Representação de uma célula de uma lista linear duplamente encadeada.

A definição de uma célula de uma lista linear duplamente encadeada é então descrita como um tipo, como mostramos a seguir:

```
typedef struct cel {
   int chave;
   struct cel *ant;
   struct cel *prox;
} celula;
```

Uma representação gráfica de uma lista linear em alocação encadeada é mostrada na figura 22.2.

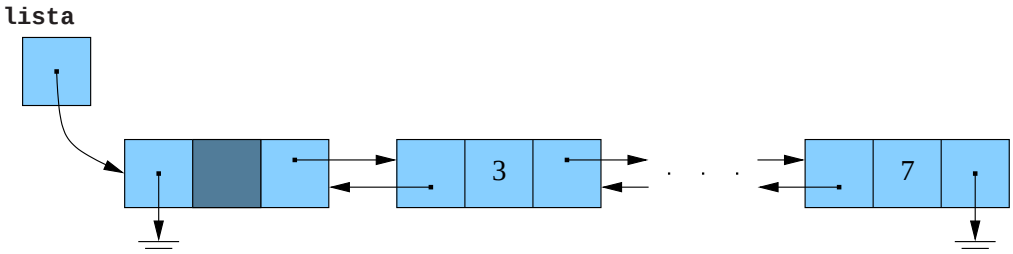

Figura 22.2: Representação de uma lista linear duplamente encadeada com cabeça **lista**. A primeira célula tem seu campo **ant** apontando para **NULL** e a última tem seu campo **prox** também apontando para **NULL**.

Uma lista linear duplamente encadeada com cabeça pode ser criada e inicializada da seguinte forma:

```
celula *lista;
lista = (celula *) malloc(sizeof (celula));
lista->ant = NULL;
lista->prox = NULL;
```

e sem cabeça como abaixo:

```
celula *lista;
lista = NULL;
```

A figura 22.3 ilustra a declaração e inicialização de uma lista linear duplamente encadeada (a) com cabeça e (b) sem cabeça.

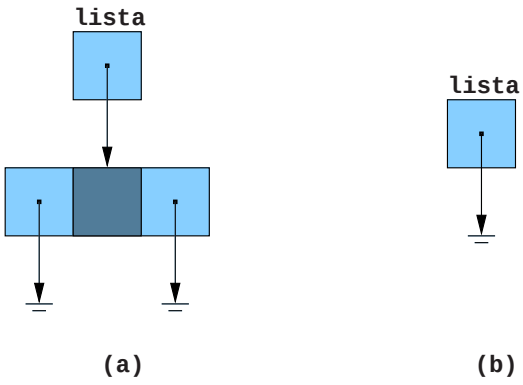

Figura 22.3: Lista linear duplamente encadeada vazia (a) com cabeça e (b) sem cabeça.

A seguir vamos discutir e implementar as operações básicas sobre listas lineares duplamente encadeadas com cabeça, deixando as liestas lineares sem cabeça para os exercícios.

22.2 Busca

A função `busca_dup_C` recebe uma lista linear duplamente encadeada `lst` e um valor `x` e devolve uma célula que contém a chave procurada `x`, caso `x` ocorra em `lst`. Caso contrário, a função devolve `NULL`.

```
/* Recebe uma chave x e um alista linear duplamente encadeada lst
   e devolve a célula que contém x em lst ou NULL caso contrário */
celula *busca_dup_C(int x, celula *lst)
{
   celula *p;

   p = lst->prox;
   while (p != NULL && p->chave != x)
      p = p->prox;

   return p;
}
```

Observe que a função `busca_dup_C` é praticamente uma cópia da função `busca_C`.

22.3 Busca seguida de remoção

No primeiro passo da busca seguida de remoção, uma busca de uma chave é realizada na lista. Se a busca tem sucesso, então o ponteiro para a célula que ser quere remover também dá acesso à célula posterior e anterior da lista linear duplamente encadeada. Isso é tudo o que precisamos saber para realizar o segundo passo, isto é, realizar a remoção satisfatoriamente. A função `busca_remove_dup_C` implementa essa idéia.

```
/* Recebe um número inteiro x e uma lista duplamente encadeada lst e remo-
   ve da lista a primeira célula que contiver x, se tal célula existir */
void busca_remove_dup_C(int x, celula *lst)
{
   celula *p;

   p = lst->prox;
   while (p != NULL && p->chave != x)
      p = p->prox;
   if (p != NULL) {
      p->ant->prox = p->prox;
      if (p->prox != NULL)
         p->prox->ant = p->ant;
      free(p);
   }
}
```

A função `busca_remove_dup_C` é muito semelhante à função `busca_remove_C`, mas não há necessidade do uso de dois ponteiros para duas células consectivas da lista. A figura 22.4

ilustra um exemplo de execução dessa função.

22.4 Busca seguida de inserção

A função `busca_insere_dup_C` a seguir realiza a busca de uma chave `x` e insere uma chave `y` antes da primeira ocorrência de `x` em uma lista linear duplamente encadeada `lst`. Caso `x` não ocorra na lista, a chave `y` é inserida no final de `lst`.

```
/* Recebe dois números inteiros y e x e uma lista duplamente
    encadeada lst e insere uma nova célula com chave y nessa
   lista antes da primeira que contiver x; se nenhuma célula
   contiver x, a nova célula é inserida no final da lista */
void busca_insere_dup_C(int y, int x, celula *lst)
{
   celula *p, *q, *nova;

   nova = (celula *) malloc(sizeof (celula));
   nova->chave = y;
   p = lst;
   q = lst->prox;
   while (q != NULL && q->chave != x) {
      p = q;
      q = q->prox;
   }
   nova->ant = p;
   nova->prox = q;
   if (p != NULL)
      p->prox = nova;
   if (q != NULL)
      q->ant = nova;
}
```

A função `busca_insere_dup_C` é também muito semelhante à função `busca_insere_C`, mas não há necessidade de manter dois ponteiros como na primeira função. Veja a figura 22.5.

Exercícios

22.1 Escreva funções para implementar as operações básicas de busca, inserção e remoção em uma lista linear duplamente encadeada sem cabeça.

22.2 Escreva uma função que receba um ponteiro para o início de uma lista linear em alocação duplamente encadeada, um ponteiro para o fim dessa lista e uma chave, e realize a inserção dessa chave no final da lista.

22.3 Listas lineares duplamente encadeadas também podem ser listas circulares. Basta olhar para uma lista linear duplamente encadeada e fazer o ponteiro para a céula anterior da célula cabeça apontar para a última célula e a ponteiro para a próxima célula da última

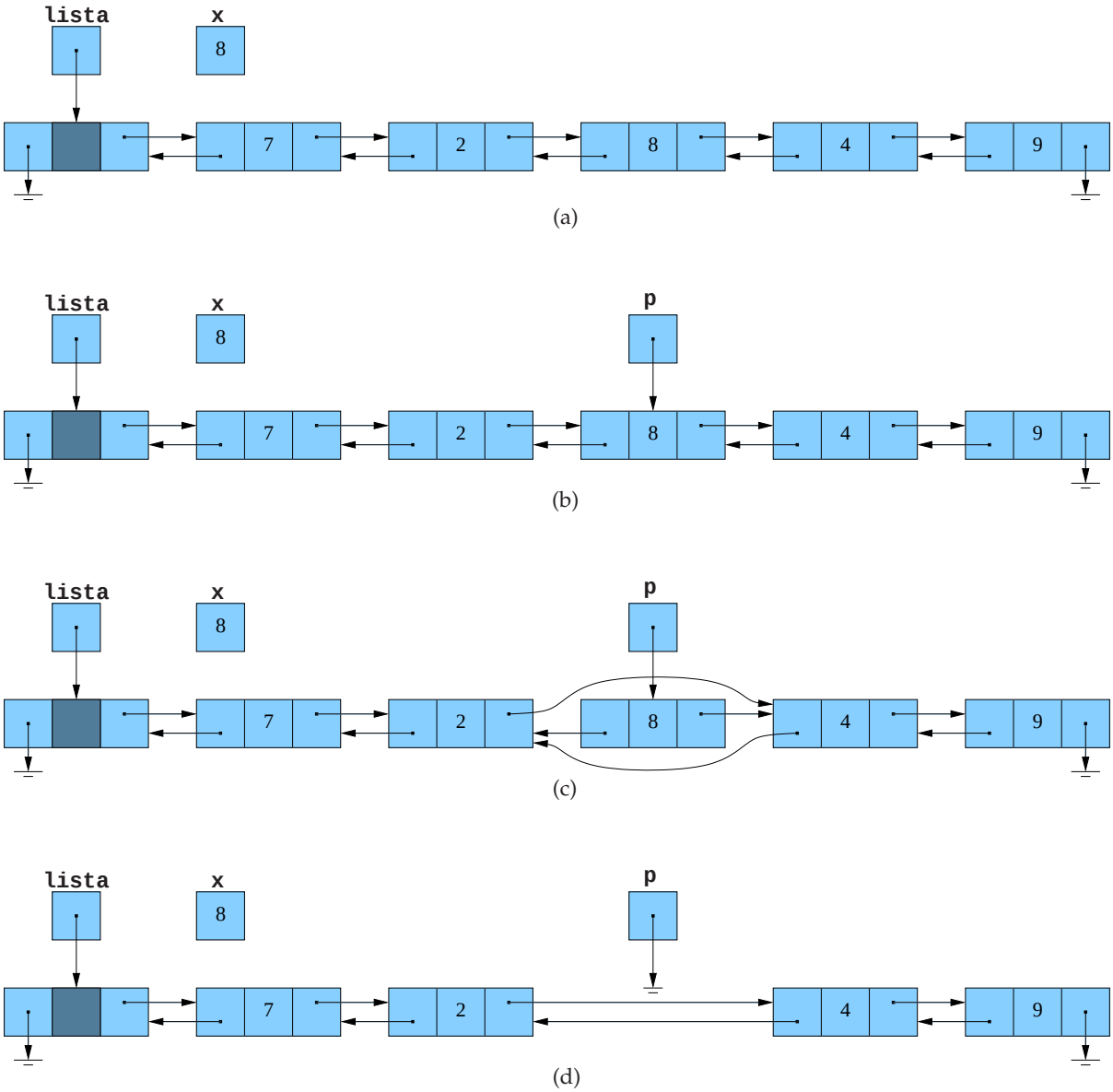

Figura 22.4: Remoção em uma lista linear duplamente encadeada com cabeça.

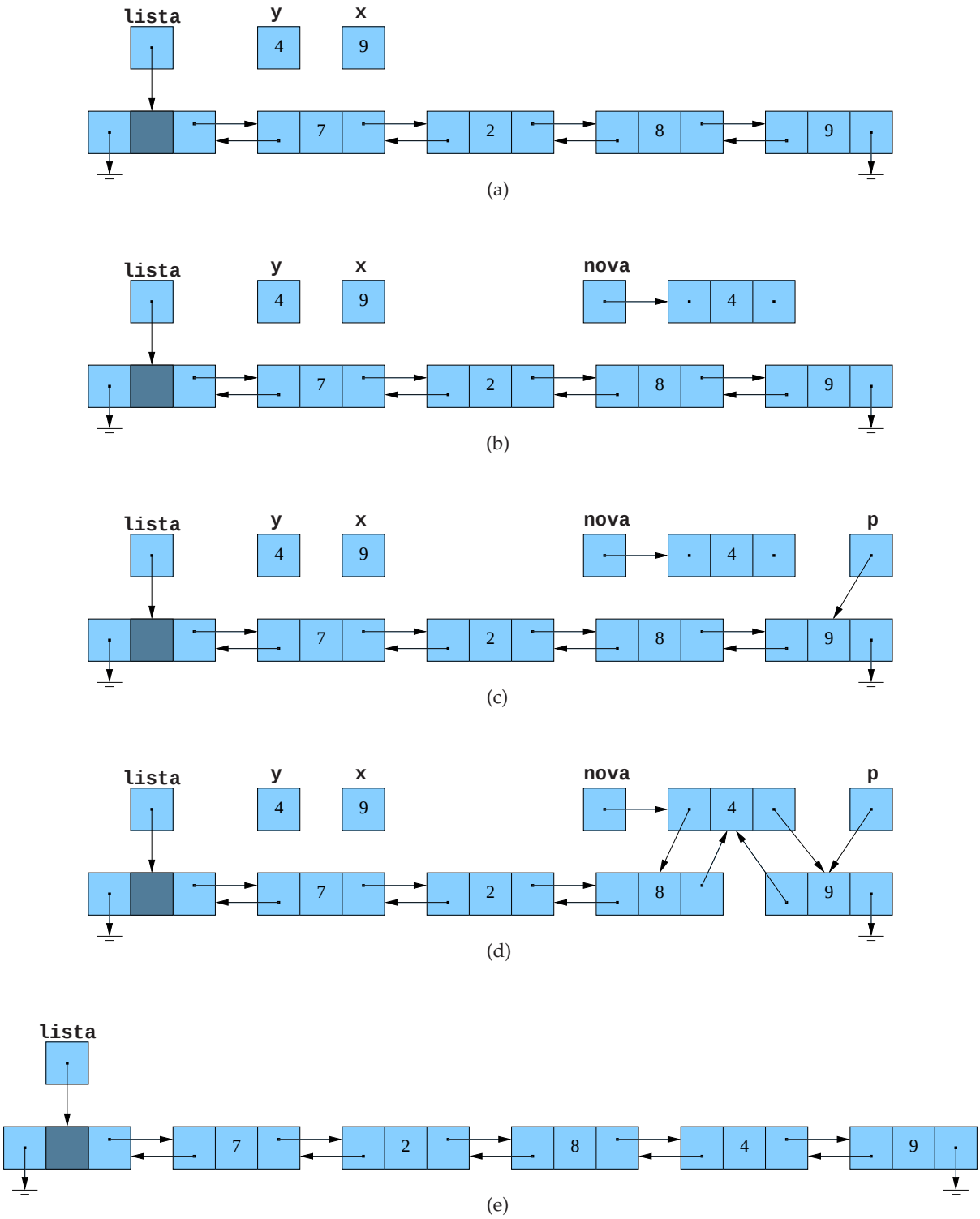

Figura 22.5: Inserção em uma lista linear duplamente encadeada com cabeça.

célula apontar para a célula cabeça. Escreva funções para implementar as operações básicas de busca, inserção e remoção em uma lista linear duplamente encadeada circular com cabeça.

22.4 Considere uma lista linear duplamente encadeada contendo as chaves $c_1, c_2, \ldots, c_n$, como na figura 22.6.

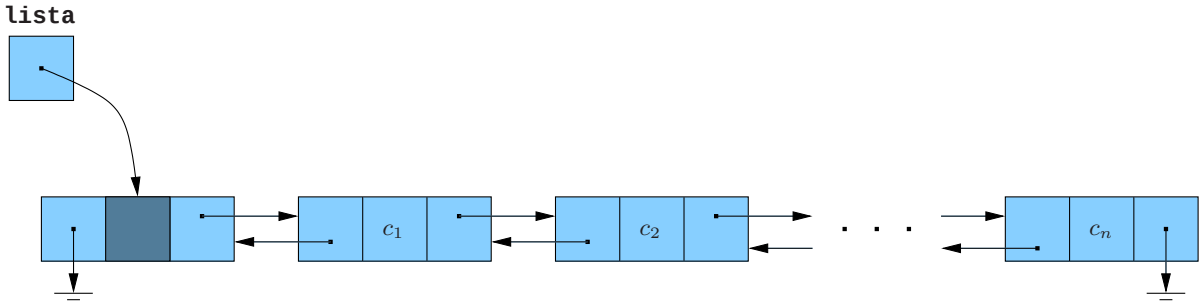

Figura 22.6: Uma lista linear duplamente encadeada.

Escreva uma função que altere os ponteiros `ant` e `prox` da lista, *sem* mover suas informações, tal que a lista fique invertida como na figura 22.7.

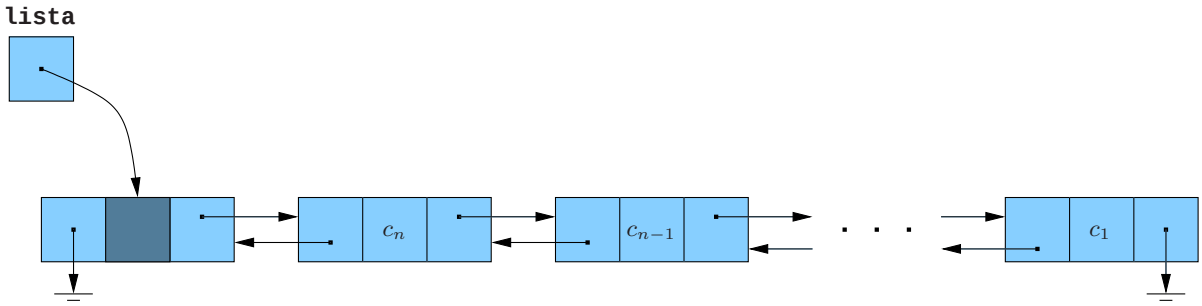

Figura 22.7: Inversão da lista linear duplamente encadeada da figura 22.6.

22.5 Suponha que você queira manter uma lista linear duplamente encadeada com cabeça com as chaves em ordem crescente. Escreva as funções de busca, inserção e remoção para essa lista.