Laboratório de Algoritmos e Técnicas de Programação II

Aula 08 - Listas lineares: formas de representação, alocação dinâmica, operações, listas encadeadas, duplamente encadeadas e circulares

Álvaro Magri Nogueira da Cruz

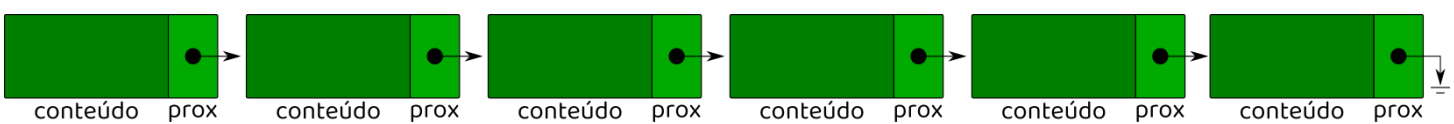
Agenda

1. Introdução
2. Estrutura de uma lista encadeada
3. Pilha
4. Fila
5. Lista duplamente encadeada
6. Exercícios
7. Referências

Introdução I

Observações

- Uma lista encadeada é uma representação de uma sequência de nós;
- Todos os nós são do mesmo tipo;
- Ficam armazenados na memória RAM do computador;
- Cada elemento da sequência é armazenado em uma célula da lista:
  - O primeiro elemento na primeira célula, o segundo na segunda, e assim por diante.



Estrutura de uma lista encadeada I

O nó

- Um nó de uma lista representa um tipo de dado;
  - Representado por uma *struct*.
- Toda lista é preenchida por elementos **do mesmo tipo**;
- Para unir os nós utiliza-se **ponteiros**;
- Os ponteiros guardam sempre a posição do **próximo elemento**.
  - Para lista encadeadas **simples**.

```
1 struct no{
2     int conteudo;
3     struct no *prox;
4 }no;
```

Estrutura de uma lista encadeada II

- É conveniente tratar as células como um novo tipo-de-dados e atribuir um nome a esse novo tipo:

```
1 typedef struct no{
2     int conteudo;
3     struct no *prox;
4 }no;
```

Pilha I

Observação

- Para criar um nó, devemos utilizar alocação dinâmica de memória;
  - *novoNo = (no*)* **malloc**(**sizeof**(*no*));
  - Dessa maneira agora podemos utilizar os campos da struct;
  - Neste exemplo: *conteudo* e **prox*.

A pilha

- Vamos olhar sobre a ótima de uma pilha;
- Devemos sempre **inserir** na **CABEÇA** da lista;
- Devemos sempre **remover** na **CABEÇA** da lista;

Pilha II

```
1  no *insere(int x, no *lista){
2      no *novoNo;
3      novoNo = (no*) malloc(sizeof(no));
4      novoNo->conteudo = x;
5      novoNo->prox = NULL;
6      if(lista==NULL){
7          lista = novoNo;
8      }
9      else{
10         novoNo->prox = lista;
11         lista = novoNo;
12     }
13     return lista;
14 }
```

Pilha III

```
1 no *remover(no *lista){
2     no *aux = lista;
3     if(lista==NULL) printf("Lista vazia!\n");
4     else{
5         lista = aux->prox;
6         free(aux);
7     }
8     return lista;
9 }
```

Fila I

A fila

- Agora vamos olhar sobre a ótima de uma fila;
- Devemos sempre **inserir** na **CAUDA** da lista;
- Devemos sempre **remover** na **CABEÇA** da lista;

```
1 no *insere(int x, no *lista){
2     no *novoNo;
3     no *aux = lista;
4     novoNo = (no*) malloc(sizeof(no));
5     novoNo->conteudo = x;
6     novoNo->prox = NULL;
7     if(lista==NULL){
8         lista = novoNo;
```

Fila II

```
9       }
10      else{
11          while(aux->prox!=NULL){
12                  aux = aux->prox;
13          }
14          aux->prox = novoNo;
15          aux = novoNo;
16      }
17      return lista;
18  }
```

Fila III

```
1 no *remover(no *lista){
2     no *aux = lista;
3     if(lista==NULL) printf("Lista vazia!\n");
4     else{
5         lista = aux->prox;
6         free(aux);
7     }
8     return lista;
9 }
```

Lista duplamente encadeada I

Definição

- As listas duplamente encadeadas são **mais complexas** que a simplesmente encadeada.
- A conexão entre os elementos é feita por meio de **dois ponteiros**:
  - Um para o elemento **anterior**;
  - Outro para o **seguinte**.
  - O ponteiro anterior ao primeiro elemento deve apontar para NULL (o início da lista), bem como o ponteiro seguinte ao último elemento.

Lista duplamente encadeada II

```c
typedef struct no{
    int conteudo;
    struct no *prox;
    struct no *ant;
}no;
```

```c


no *insere(int x, no *lista){//Insere no inicio da lista
    no *novoNo;
    no *aux = lista;
    novoNo = (no*) malloc(sizeof(no));//Aloca memoria para o
     novo no
    novoNo->conteudo = x;
    novoNo->prox = NULL;
```

Lista duplamente encadeada III

```
9       novoNo->ant = NULL;
10      if(lista==NULL){//Verifica se a lista esta vazia
11          lista = novoNo;
12      }
13      else{
14          novoNo->prox = aux;
15          aux->ant = novoNo;
16          aux = novoNo;
17          lista = aux;
18      }
19      return lista;
20  }
```

Lista duplamente encadeada IV

```
1  no *remover(no *lista){ //Remove sempre a cabeca da lista
2      no *aux = lista;
3      if(lista==NULL) printf("Lista vazia!\n");//Verifica se a
       lista esta vazia
4      else if(aux->prox==NULL){//Se e o ultimo elemento,
       libera e lista=NULL
5          free(aux);
6          lista = NULL;
7      }
8      else{//Se tem mais de um elemento, apenas remove a
       cabeca
9          lista = aux->prox;
10         lista->ant = NULL;
11         free(aux);
12     }
13     return lista;
14 }
```

Exercícios I

1. Crie um programa que crie uma lista simplesmente encadeada e permita a busca de um elemento. E.g.: Caso o usuário queira buscar o número "5" na lista, o programa deverá percorrer a lista e verificar se este existe e qual posição se encontra.
2. No mesmo contexto do primeiro exercício, crie um programa que permita buscar e excluir um elemento da lista. E.g.: Caso o usuário queira excluir o número "5" da lista, o programa deverá percorrer a lista e verificar se este existe, caso exista, exclua-o.
3. Implemente os dos primeiros programas agora com listas duplamente encadeadas.

Exercícios II

4. Imagine que uma empresa queira um programa para controlar uma fila de um hospital. Este programa deverá controlar a ordem de chegada e também a possibilidade de uma pessoa prioritária passar na frente dos outros. Crie um algoritmo de fila que permita usuário prioritários, isto é, mesmo que não estejam na cabeça da fila, estes deverão ter prioridade de atendimento. **Algumas dicas:** Você pode ter **duas filas** distintas, uma para prioritário e uma de grande concorrência, toda vez que for solicitado um atendimento, você pode alternar entre as listas para fazer a exclusão na cabeça da fila.

Referências I

1. **Deitel H. M., e Deitel P.J; "C: Como programar". 6.ed. Pearson Prentice Hall, 2011. 818p.**
2. Jean Paul Tremblay & Richard P. Bunt. "Ciência dos Computadores - Uma abordagem algorítmica". McGraw-Hill.
3. Jaime Evaristo. "Aprendendo a Programar / Programando em Turbo Pascal". Edufal - Editora da Univ. Federal de Alagoas. Maceió, 1996.
4. Harry Farrer et al. "Pascal Estruturado (da série "Programação Estruturada de Computadores")". Editora Guanabara Dois. Rio de Janeiro, 1985.
5. Stephen O'Brien. "Turbo Pascal 6 Completo e Total". Makron Books.

Referências II

6. Celes, W., Cerqueira, R., Rangel, J.L. "Introdução a Estrutura de Dados". Elsevier, 2004.
7. Feofiloff, P. "Algoritmos em Linguagem C". Elsevier, 2009. 208p.
8. Schildt, H. "C Completo e Total". 3ª ed. Pearson. 1996. 852p.