Universidade Federal Rural de Pernambuco
(UFRPE)
- Unidade Acadêmica de Garanhuns-

Listas Encadeadas

Estruturas de Dados

Prof. Priscilla Kelly Machado Vieira

Apresentação do Capítulo

- Introdução
- Conceitos Listas Simplesmente Encadeadas
  - Operações
- Conceitos Listas Circulares
- Conceitos Listas Duplamente Encadeadas
- Exercícios

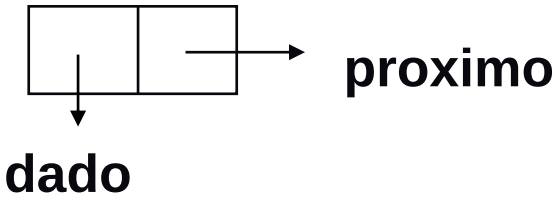
Introdução

- Coleção de nodos que juntos forma uma ordem linear
- Ordem é determinada como “siga o chefe”
- Cada nodo armazena o conteúdo a ser armazenado e uma referência para outro elemento, chamada *next* (próximo)
- E para inserir dados?

Qualquer posição que se deseja!

Conceitos

- Valores repetidos podem estar presentes em uma lista
- Listas podem ser implementadas com array ou com estruturas dinâmicas
- Listas possuem acesso sequencial



Operações

- Inserção
- Remoção
- Quantidade de elementos
- Retornar um elemento específico
- Verificar se está vazia

Estrutura da Lista


```
struct lista {
    int info;
    struct lista* prox;
};typedef struct lista Lista;
```

Inicializa Lista


```
/* função de inicialização: retorna uma lista vazia */
Lista* inicializa (void){
    return NULL;
}
```

Inserção


```
/* inserção no início: retorna a lista atualizada */

Lista* insere (Lista* l, int i){

    Lista* novo = (Lista*) malloc(sizeof(Lista));

    novo->info = i;

    novo->prox = l;

    printf("inserido!\n");

    return novo;

}
```

Percorre os Elementos da Lista


```
/* imprime valores dos elementos*/

void imprime (Lista* l){

    Lista*p; //variável auxiliar para percorrer a lista

    for (p = l; p != NULL; p = p->prox)

      printf("info = %d\n", p->info);

}
```

Busca


```
/* busca um elemento na lista*/

Lista* imprime (Lista* l, int v){

    Lista*p; //variável auxiliar para percorrer a lista

    for (p = l; p != NULL; p = p->prox)

        if (p->info == v)

            return p;

    }

      return NULL; //Não achou o elemento

}
```

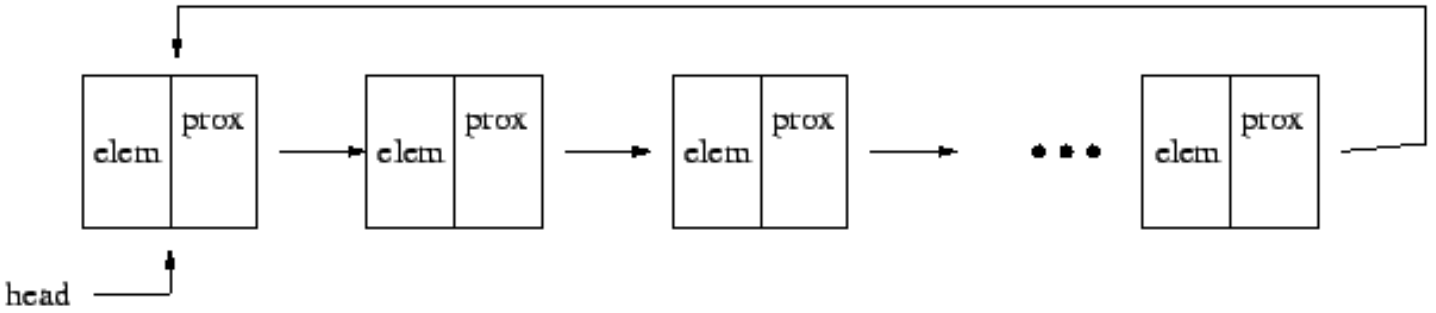
Libera


```
void libera (Lista* l){
    Lista* p = l;
    while (p != NULL) {
        Lista* t = p->prox; /* guarda referência para o próximo
    elemento */
        free(p); /* libera a memória apontada por p */
        p = t;  /* faz p apontar para o próximo */
    }
}
```

```
/* função retira: retira elemento da lista */
Lista* retira (Lista* l,int v) {
    Lista* ant = NULL;
    Lista* p = l;  /* procura elemento na lista, guardando anterior */
    while (p != NULL && p->info != v) {
        ant = p;
        p = p->prox;
    }    /* verifica se achou elemento */
    if (p == NULL)
        return l;    /* não achou: retorna lista original */
    if (ant == NULL) {    /* retira elemento */
        /* retira elemento do inicio */
        l = p->prox;
    }
    else {    /* retira elemento do meio da lista */
        ant->prox = p->prox;
    }
    free(p);
    return l;
}
```

Listas Circulares

- Sucessor do último elemento ser o primeiro elemento da lista
  - Não precisa considerar casos especiais de inserção e remoção



Listas Duplamente Encadeadas

- Ponteiro apontando para o elemento anterior
- Ponteiro apontando para o próximo elemento
  - Encadeamento simples dificulta remoção
  - Caminhamento inverso

Estrutura da Lista


```
struct lista {
    int info;
    struct lista* ant;
    struct lista* prox;
}; typedef struct lista Lista;
```

Inserção


```
Lista* lst_insere (Lista* l, int i){
    Lista* novo = (Lista*) malloc (sizeof(Lista));
    novo->info = i;
    novo->prox = l;
    novo->ant = NULL;
    if (l != NULL) //Verifica se a lista estava vazia
        l->ant=novo;

    return novo;
}
```

Remoção


```
Lista* lst_retira (Lista* l, int i){
    Lista* p = lst_busca(l, i);    //procura o elemento
    if (p == NULL)    //não encontrou o elemento
        return l;
    if (l == p)
        l = p->prox; // testa se é o primeiro elemento
    else
        p->ant->prox = p->prox;
    if (p->prox != NULL)    //verifica se é o último
elemento
        p->prox->ant = p->ant;
    free (p);
    return l;
}
```

Considerações Finais

- Além dessas estruturas, é possível criar muitas outras, inclusive juntando estruturas diferentes para se formar uma
- Pilhas e Filas
- O retorno das funções podem ser diferentes, de acordo com a necessidade, e assim por diante
- As estruturas de dados são “maleáveis” e você pode utilizá-las como quiser, desde que preserve os conceitos fundamentais de cada uma

Laboratório 10

- Crie uma função que insere inteiros de forma ordenada em uma lista simplesmente encadeada.