Listas Lineares Encadeadas

Algoritmos e Estruturas de Dados

Prof. Dr. Luciano Demétrio Santos Pacífico

{luciano.pacifico@ufrpe.br}

UNIVERSIDADE
FEDERAL RURAL
DE PERNAMBUCO

Conteúdo

- **Introdução**
- **Alocação Encadeada**
- **Listas Encadeadas**

Introdução

UNIVERSIDADE
FEDERAL RURAL
DE PERNAMBUCO

Listas Lineares Sequenciais (Relembrando)

- Uma lista linear ou tabela é um conjunto de $n \geq 0$ elementos (nós) $L[1], L[2], ..., L[n]$, tais que suas propriedades estruturais decorrem, unicamente, da posição relativa dos mesmos dentro de uma sequência linear.

  - Se $n > 0$, $L[1]$ é o primeiro nó;

  - Para $1 < k \leq n$, o nó $L[k]$ é precedido por $L[k - 1]$.

Listas Lineares – Casos Particulares (Relembrando)

- ***Deques*** (*double ended queue*): remoções e inserções permitidas apenas nas extremidades da lista.

- **Pilhas**: remoções e inserções permitidas apenas em uma única extremidade da fila.

- **Fila**: inserções realizadas em um extremo, remoções no outro.

Listas Lineares Sequenciais - Relembrando

- Em uma lista linear alocada estaticamente (sequencialmente), há necessidade de saber exatamente quantos elementos máximos a aplicação suportará.

- Contudo, em uma aplicação real, é extremamente difícil saber exatamente quantos elementos estarão contidos na estrutura.

- Na alocação sequencial há ainda desperdício de memória, caso haja um dimensionamento exagerado do tamanho da estrutura.

Alocação Encadeada

UNIVERSIDADE
FEDERAL RURAL
DE PERNAMBUCO

Alocação Encadeada

- A **alocação encadeada** (ou **alocação dinâmica**) resolve o problema existente na alocação sequencial de necessidade da determinação, no início da execução do algoritmo, do número *n* máximo de elementos armazenados na estrutura de dados.

- Os nós em uma estrutura com alocação encadeada podem estar dispostos de forma aleatória na memória do computador onde o algoritmo é executado.

Alocação Encadeada

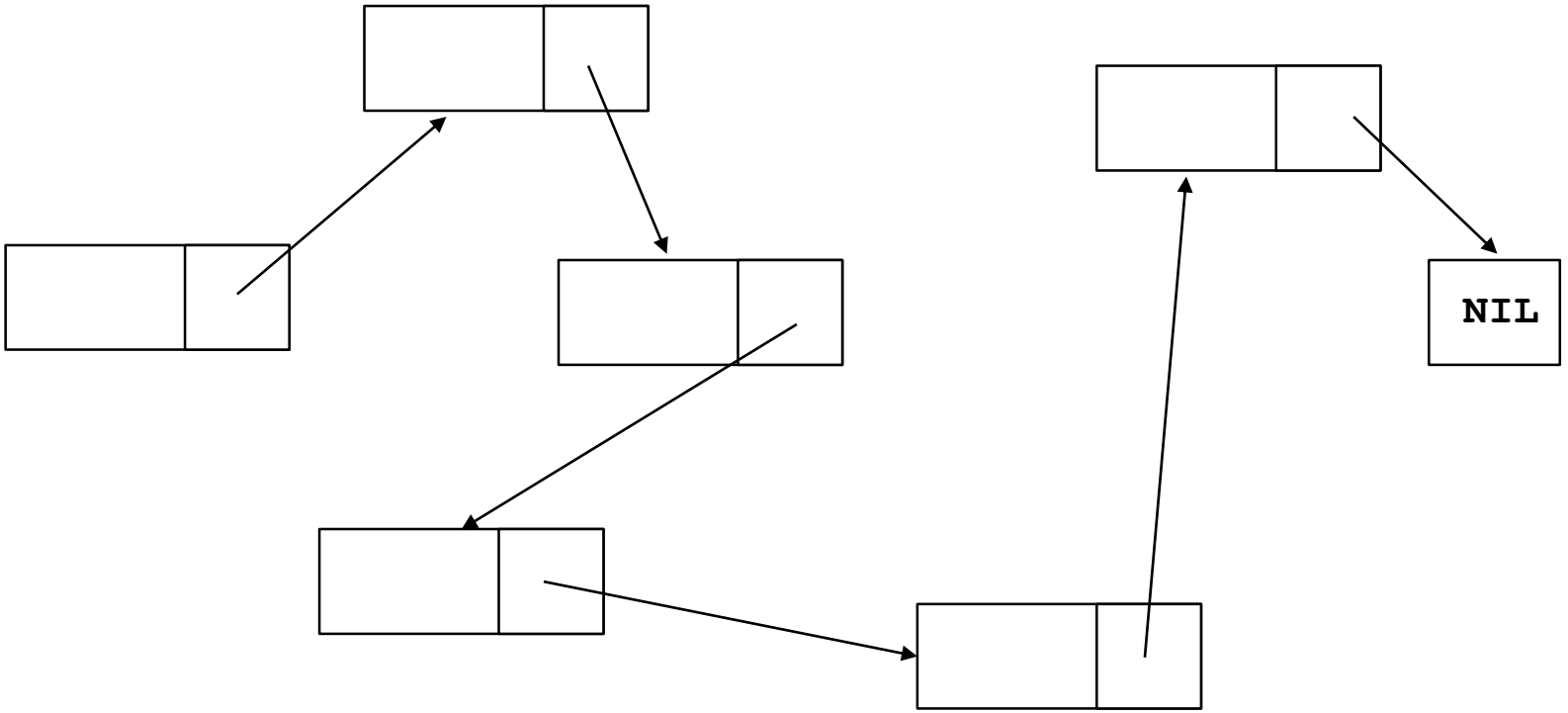
- Os nós de uma lista encadeada são ligados através de **ponteiros**, que indicam o próximo elemento na sequência.

- É necessário que cada elemento armazene um campo para informação do próximo nó da sequência, o que requer mais espaço de memória.

- Facilita operações como a concatenação de listas.

Alocação Encadeada

- Estrutura:



Listas Encadeadas

UNIVERSIDADE
FEDERAL RURAL
DE PERNAMBUCO

Listas Encadeadas

- Qualquer estrutura de dados, quando representada através da alocação encadeada, requer um ponteiro que indique o seu primeiro elemento.

- O percurso de uma lista é feito através deste ponteiro, muitas vezes chamado de **nó-cabeça**.

Listas Encadeadas

- As listas lineares encadeadas podem ser divididas em dois grupos principais:
  - Listas simplesmente encadeadas;
  - Listas duplamente encadeadas.
- As listas encadeadas podem ser ainda listas **circulares**.

Listas Encadeadas

- Os procedimentos vistos até agora para o caso de listas sequenciais podem ser facilmente adaptados para o contexto de listas encadeadas.

- Em determinadas situações, o uso de listas duplamente encadeadas pode representar ganhos em performance em comparação às listas simplesmente encadeadas, ao custo da necessidade de maior espaço de armazenamento (cada indivíduo deverá armazenar dois ponteiros).

  - Também requer cuidado adicional para que todos os ponteiros sejam mantidos atualizados corretamente.

Listas Simplesmente Encadeadas

- Cada nó possui apenas um único ponteiro, que serve para indicar o próximo elemento da sequência.

- A lista será percorrida apenas em um sentido.

- Cuidados especiais devem ser tomados pelos procedimentos de inserção e remoção para garantir a **integridade da estrutura**.

Lista Simplesmente Encadeada – Estruturas Básicas

- Por uma questão de organização do código, criaremos duas Estruturas de Dados para a representação das Listas Simplesmente Encadeadas.

```
1. registro NoListaSimplesEnc
2.      chave:inteiro,
3.      proximo:NoListaSimplesEnc
```

```
1. registro ListaSimplesEnc
2.      cabeca:NoListaSimplesEnc
```

Lista Simplesmente Encadeada

- Imprimindo todos os elementos contidos na lista:

```
1. //L -> ListaSimplesEnc
2. procedimento imprimirListaEnc(L)
3.       pt = L.cabeca
4.       enquanto pt != NIL
5.             imprimir(pt.chave)
6.             pt = pt.proximo
```

Lista Simplesmente Encadeada

- Como optamos por retornar no máximo **um valor por procedimento** (seguindo o exemplo de linguagens como **Java** e **C**), para o procedimento de busca em listas simplesmente encadeadas, precisaremos passar um ponteiro como **parâmetro** que será usado para recuperarmos a posição do elemento buscado (se existir) na estrutura de dados.
  - Linguagens que permitem que mais de um valor seja retornado por função (como **Python**) podem omitir esse novo parâmetro.

```
1.  //ant -> Nó auxiliar que armazenará posição anterior ao nó buscado
2.  procedimento buscarListaSimplesEnc(x, ant, L)
3.      pt = L.cabeca
4.      ant = NIL
5.      se pt != NIL
6.          enquanto (pt != NIL) e (pt.chave != x)
7.              ant = pt
8.              pt = pt.proximo
9.      senão
10.         imprimir("Lista Vazia")
11.     retorne pt
```

Lista Simplesmente Encadeada

- Inserção

```
1.  //X -> é o NoListaSimplesEnc a ser inserido
2.  procedimento inserirListaSimplesEnc(X, L)
3.      ant = NIL
4.      pt = buscarListaSimplesEnc(X.chave, ant, L)
5.      se pt == NIL //Não será permitida inserção de chaves iguais
6.          se ant == NIL // Só ocorre se a lista estiver vazia
7.              L.cabeca = X
8.          senão
9.              ant.proximo = X
10.     senão
11.         imprimir("Nó " + X.chave + " já existe!")
```

Lista Simplesmente Encadeada

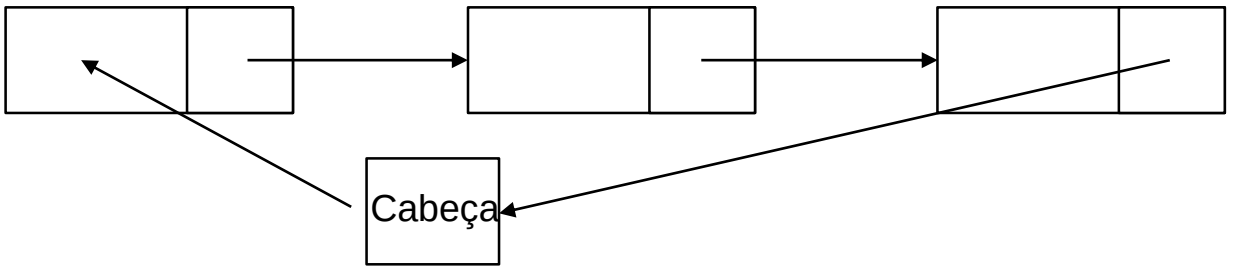
- Remoção

```
1.  //x -> chave do nó a ser removido
2.  procedimento removerListaSimplesEnc(x, L)
3.      ant = NIL
4.      pt = buscarListaSimplesEnc(x, ant, L)
5.      se pt != NIL
6.          se ant == NIL // Só ocorre se pt for o nó cabeça da lista
7.              L.cabeca = pt.proximo
8.          senão
9.              ant.proximo = pt.proximo
10.         pt.proximo = NIL //remove acesso de pt aos demais nós da lista
11.     senão
12.         imprimir("Nó " + x + " não existe!")
13.     retorne pt
```

Listas Circulares

- Uma lista circular nada mais é que uma lista simplesmente encadeada cujo último elemento aponta para o ponteiro cabeça da lista.

- Com esse tipo de estrutura a preocupação passa a ser uma função de parada que possa ser incorporada à busca.



Listas Duplamente Encadeadas

- Neste tipo de estrutura de dados, existe a necessidade de uma maior quantidade de memória de armazenamento que nas listas simplesmente encadeadas.

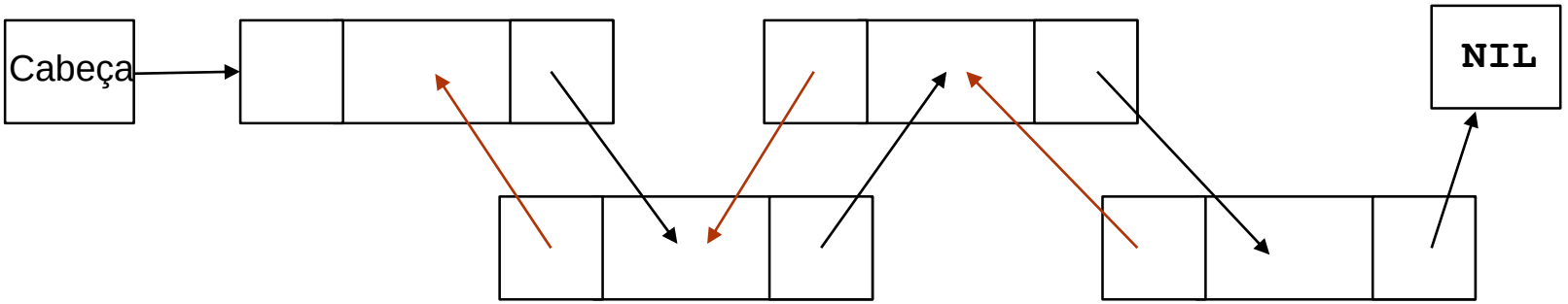
- O ganho será representado pela facilidade de execução das operações de busca, inserção e remoção.

- Além disso, agora outras operações serão permitidas, como percorrer a lista ao contrário.

Listas Duplamente Encadeadas

- O procedimento de busca agora retorna apenas o nó procurado, e a partir do mesmo podemos realizar a remoção e a inserção.

- Também podemos ter listas circulares duplamente encadeadas.



Lista Simplesmente Encadeada – Estruturas Básicas

- Por uma questão de organização do código, criaremos duas Estruturas de Dados para a representação das Listas Duplamente Encadeadas.

```
1. registro NoListaDuplaEnc
2.      chave:inteiro,
3.      anterior:NoListaDuplaEnc,
4.      proximo:NoListaDuplaEnc
```

```
1. registro ListaDuplaEnc
2.      cabeca:NoListaDuplaEnc
```

Lista Duplamente Encadeada

- Busca
  - O código da busca será simplificado com o uso da estrutura de Lista Duplamente Encadeada:

```
1.  //x -> chave do nó buscado
2.  procedimento buscarListaDuplaEnc(x, L)
3.      pt = L.cabeca
4.      se pt != NIL
5.          enquanto (pt != NIL) e (pt.chave != x)
6.              pt = pt.proximo
7.      senão
8.          imprimir("Lista Vazia")
9.      retorne pt
```

Lista Duplamente Encadeada

- Inserção

```
1.  //X -> é o NoListaDuplaEnc a ser inserido
2.  procedimento inserirListaDuplaEnc(X, L)
3.      pt = buscarListaDuplaEnc(X.chave, L)
4.      se pt == NIL //Não será permitida inserção de chaves iguais
5.          X.proximo = L.cabeca //Inserção no início da lista
6.          se L.cabeca != NIL //Ocorre se a lista não estiver vazia
7.              L.cabeca.anterior = X
8.          L.cabeca = X
9.      senão
10.         imprimir("Nó " + X.chave + " já existe!")
```

Lista Duplamente Encadeada

- Remoção

```
1.  //x -> chave do nó a ser removido
2.  procedimento removerListaDuplaEnc(x, L)
3.      pt = buscarListaDuplaEnc(x, ant, L)
4.      se pt != NIL
5.          se pt.anterior == NIL //Só ocorre se pt for o nó cabeça da lista
6.              L.cabeca = pt.proximo
7.          senão
8.              pt.anterior.proximo = pt.proximo
9.          se pt.proximo != NIL //o nó não era o último da lista
10.             pt.proximo.anterior = pt.anterior
11.         pt.proximo = NIL //remove acesso de pt aos demais nós da lista
12.         pt.anterior = NIL
13.     senão
14.         imprimir("Nó " + x + " não existe!")
15.     retorne pt
```

Referências

- SZWARCFITER, J.; MARKENZON, L. Estruturas de Dados e seus Algoritmos, 3ª ed. Rio de Janeiro: LTC, 2010.

- CORMEN, H. T.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. Introduction to Algorithms, 3rd ed., *Boston: MIT Press*, 2009.

- FEOFILOFF, Paulo. Algoritmos em Linguagem C. Editora Campus/Elsevier, 2009.

Listas Lineares Encadeadas

Algoritmos e Estruturas de Dados

Prof. Dr. Luciano Demétrio Santos Pacífico

{luciano.pacifico@ufrpe.br}

UNIVERSIDADE
FEDERAL RURAL
DE PERNAMBUCO