Aula 3

Estrutura de Dados

Prof. Vinicius Pozzobon Borin

Conversa Inicial

- O objetivo desta aula é apresentar os conceitos que envolvem a estrutura de dados do tipo lista
- Será mostrado como realizar as manipulações dos dados dentro de uma estrutura de lista, como a construção, inserção e manipulação de dados

- Os tipos de listas e suas características serão
  - Lista simplesmente encadeada (circular e não circular)
  - Lista duplamente encadeada (circular e não circular)

- Também serão apresentas outras duas estruturas de dados que podem ser construídas a partir de listas, mas que apresentam características únicas de operação
  - Pilhas
  - Filas

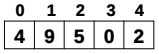
Conceitos de listas encadeadas

- Existem estruturas de dados com alocação sequencial na memória. Exemplo: vetores, matrizes e registros
- Existem estruturas de dados com alocação não sequencial, ou seja, cada dado da estrutura pode estar posicionado em qualquer parte da memória destinada ao programa em execução

- Como o programa consegue localizar cada dado da estrutura, uma vez que eles estão posicionados aleatoriamente na memória?
  - Solução: ponteiros

- Não existe o conceito de índice em uma lista
- Na lista, conhecemos somente o endereço do primeiro elemento
- Sem índice, como acessamos cada dado da lista? Na lista, cada elemento contém o endereço do próximo

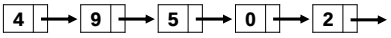
- Vetor



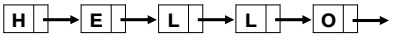
- Lista

- Vantagem da lista
  - Alocação dinâmica de memória
- Desvantagem da lista
  - Tempo de acesso ao elemento de um vetor: O(1)
  - Tempo de acesso ao elemento da lista: O(n)

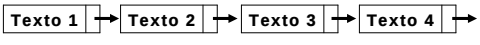
- Lista Homogênea Numérica



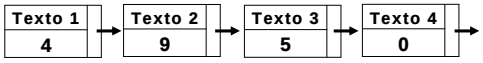
- Lista Homogênea Alfanumérica



- Lista Homogênea Alfanumérica



- Lista Heterogênea



- Aplicações
  - Agenda de contatos
  - Programa de solução de equações matemáticas
  - Reprodutor de música

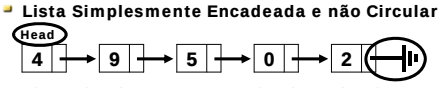
Lista simplesmente encadeada (*linked list*)

Listas encadeadas simples

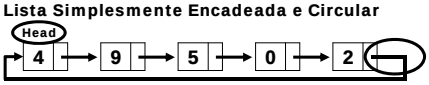
- Cada elemento (nó) de uma lista conterá um dado (ou um conjunto de dados) e um ponteiro contendo o endereço para o próximo elemento da lista
- A lista encadeada simples funciona como se fosse uma via de mão única
- Sendo assim, cada elemento conhece somente o seu subsequente, não sendo possível retornar para o elemento imediatamente anterior

Criação dos elementos da lista

- Lista Simplesmente Encadeada e não Circular

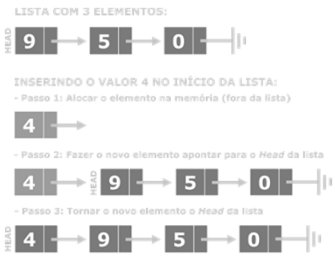


- Lista Simplesmente Encadeada e Circular




```
registro ElementoDaLista_Simples
        dado: inteiro
        prox: ElementoDaLista[->)
fimregistro
```

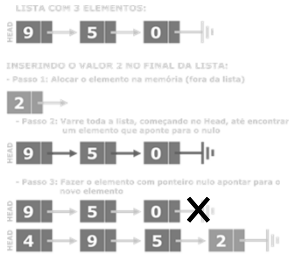
Inserindo no início da lista simples



Inserindo no início da lista simples

```
1  //Cria um Procedimento que recebe como parâmetro o dado a ser inserido no início
2  função InserirInício (numero: inteiro): sem retorno
3  var
4     //Declara um novo nó do tipo REGISTRO
5     NovoElemento[->]: registro ElementoDaLista_Simples        PASSO 1
6  início
7  //Insere no novo nó o dado recebido como parâmetro
8     NovoElemento->dado = numero
9
10    //Verifica se o HEAD está vazio
11    se (Head == NULO) então
12       //Se HEAD está vazio, a lista está vazia!
13       //Portanto, novo elemento será o HEAD!
14       Head = NovoElemento
15       Head->prox = NULO
16    senão
17       //Se HEAD não está vazio, existem dados na lista
18       //Portanto, novo elemento aponta para o HEAD
19       NovElemento->prox = Head                              PASSO 2
20       //Novo elemento vira o HEAD da lista
21       Head = NovoElemento                                   PASSO 3
22    fimse
23 fimfunção
```

Inserindo no final da lista simples



Inserindo no final da lista simples

Inserindo no meio da lista simples

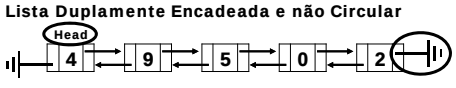
Lista duplamente encadeada (*doubly linked list*)

Listas encadeadas duplas

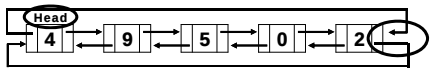
- Cada elemento (nó) de uma lista conterá um dado (ou um conjunto de dados) e um ponteiro contendo o endereço para o próximo e o elemento anterior da lista
- A lista encadeada dupla funciona como se fosse uma via de mão dupla
- Sendo assim, cada elemento conhece o seu subsequente e seu antecessor

Criação dos elementos da lista

- Lista Duplamente Encadeada e não Circular



- Lista Duplamente Encadeada e Circular



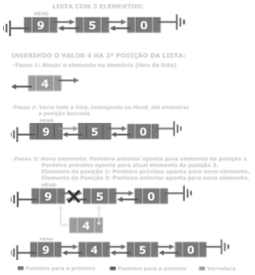
- Poteiro para o próximo
- Poteiro para o anterior

```
registro ElementoDaLista_Dupla
        dado: inteiro
        prox: ElementoDaLista[->)
        ant: ElementoDaLista[->)
fimregistro
```

Inserindo no início da lista dupla

Inserindo no final da lista dupla

Inserindo no meio da lista dupla

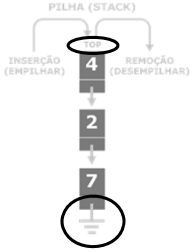


Pilha (*stack*)

First in last out (FILO)

- Em uma pilha, só é possível inserir um elemento ou remover esse elemento do topo da pilha
- O primeiro elemento que entra é o último que sai

First in last out (FILO)



Criação dos elementos da pilha

- Podemos implementar utilizando vetores ou listas
- Para lista, temos

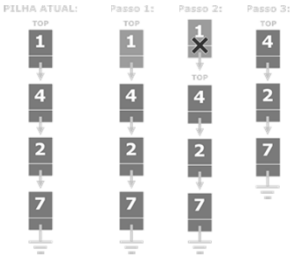
```
registro Pilha
        dado: inteiro
        prox: Pilha[->)
fimregistro
```

Inserindo na pilha *(push)*

Inserindo na pilha *(push)*

```
24  função push (numero: inteiro)
25  var
26      ElementoNovo: Pilha[->)        <-- PASSO 1
27  inicio
28      //Insere o valor no novo elemento que será inserido na pilha
29      NovoElemento->dado = numero
30      //Verifica que existe algo na pilha
31      se (Top == NULO) então
32          //Se a pilha está vazia, o novo elemento será a pilha
33          //e apontará para nulo
34          NovoElemento->prox = NULO
35      senão
36          //Se a pilha já tem algo, novo elemento aponta para o topo
37          NovoElemento->prox = Top   <-- PASSO 2
38      fimse
39      //Novo elemento vira o topo, pois a inserção é sempre no topo
40      Top = NovoElemento             <-- PASSO 3
41  fimfunção
```

Removendo da pilha (*pop*)



Removendo da pilha (*pop*)

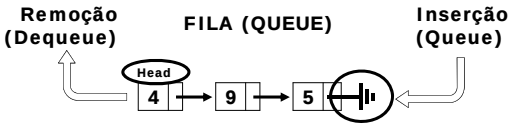
```
43  //Não passa nenhum valor como parâmetro,
44  //pois a remoção é sempre do valor do topo
45  função pop ()
46  var
47      ElementoParaRemover: Pilha[->)   <-- PASSO 1
48  inicio
49      //Verifica que existe algo na pilha
50      se (Top <> NULO) então
51          //Existe algo para remover então
52          //Salva temporariamente o topo atual
53          ElementoParaRemover = Top
54          //Incrementa o top (passa para o próximo nó)
55          Top = Top->prox              <-- PASSO 2
56          //Limpa da memória o topo antigo
57          libera(ElementoParaRemover)  <-- PASSO 3
58      fimse
59  fimfunção
```

Fila (*queue*)

First in first out (FIFO)

- Só é possível inserir um elemento no final da fila
- Só é possível remover um elemento do início da fila
- O primeiro elemento que entra é o primeiro que sai

First in first out (FIFO)



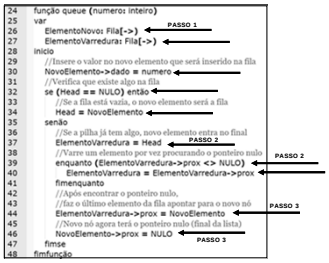
Criação dos elementos da fila

- Podemos implementar utilizando vetores ou listas
- Para lista, temos

```
registro Fila
      dado: inteiro
      prox: Fila[->)
fimregistro
```

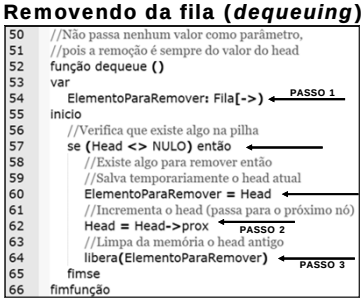
Inserindo na fila (*queuing*)

Inserindo na fila (*queuing*)



Removendo da fila (*dequeuing*)

Removendo da fila (*dequeuing*)



Referências

- ASCENCIO, A. F. G. Estrutura de dados: algoritmos, análise da complexidade e implementações em Java e C/C++. São Paulo: Pearson, 2011.
- _____. Fundamentos da programação de computadores: Algoritmos, Pascal, C/C++ (padrão ANSI) JAVA. 3. ed. São Paulo: Pearson, 2012.
- CORMEN, T. H. Algoritmos: teoria e prática. 3. ed. Rio de Janeiro: Elsevier, 2012.
- PUGA, S.; RISSETI, G. Lógica de programação e estrutura de dados. 3. ed. São Paulo: Pearson, 2016.
- MIZRAHI, V. V. Treinamento em linguagem C. 2. ed. São Paulo: Pearson, 2008.
- LAUREANO, M. Estrutura de dados com algoritmos e C. São Paulo: Brasport, 2008.