Estruturas de dados elementares

Prof. Bruno de Castro Honorato Silva

November 30, 2020

1 Introdução

Uma estrutura de dados é uma maneira particular de organizar os dados em um computador de forma que os dados possam ser facilmente utilizados e manipulados no futuro. É uma forma eficaz de realizar várias operações relacionadas ao gerenciamento de dados. Com uma compreensão suficiente da estrutura de dados, os dados podem ser organizados e armazenados de maneira adequada. As estruturas de dados são projetadas para organizar os dados a fim de atender a propósitos específicos de modo a acessar e executar as operações de maneira adequada. Os tipos de estruturas de dados mais conhecidos são:

- Fila;
- Lista;
- Pilha.

Nas seções que se seguem, são descritas cada uma destas estruturas de dados. A ordem das seções é relacionada com a da abordagem sobre tais temas feita durante o curso.

2 Pilha

Pilha é um tipo de ED na qual as inserções e remoções são feitas na mesma extremidade, chamada topo. Nas pilhas elementos são adicionados no topo e removidos do topo política Last-In/First-Out (LIFO). Para lembrar o conceito de pilha, pode-se utilizar a associação com uma pilha de pratos. Se desejas desempilhar uma pilha composta de pratos, começarás removendo o prato que está parcialmente no topo, um de cada vez.

As principais operações desta estrutura de dados são:

- empilhar(P,x): insere o elemento x no topo de P
- desempilhar(P): remove o elemento do topo de P, e retorna esse elemento

Operações auxiliares são elencadas como se seguem:

- criar(P): cria uma pilha P vazia
- apagar(P): apaga a pilha P da memória

- topo(P): retorna o elemento do topo de P, sem remover
- tamanho(P): retorna o número de elementos em P
- vazia(P): indica se a pilha P está vazia
- cheia(P): indica se a pilha P está cheia (útil para implementações estáticas).

Um exemplo de aplicação é botão *voltar* de um *smartphone* android. Ao navegar pelas aplicações do *smartphone*, cada tela aberta é persistida na memória do *smartphone*. As telas são organizados no formato de pilha, pois, cada vez que o usuário clica no botão voltar, a tela atual é desempilhada e retrocedemos a tela anterior de tal forma que podemos seguir visitando outras telas. A Figura 1 ilustra uma estrutura de pilha que armazena números inteiros.

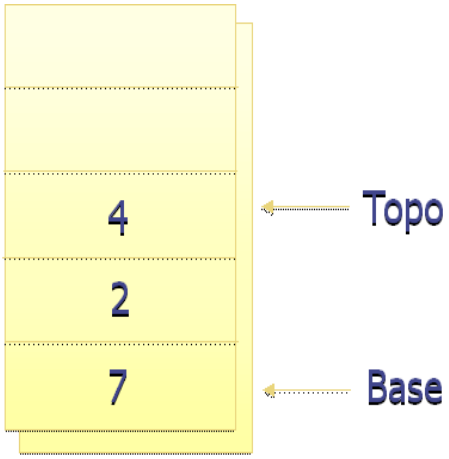

Figure 1: Ilustração de uma pilha de números inteiros.

3 Fila

Filas são estruturas nas quais as inserções são feitas em um extremo (final) e remoções são feitas no outro extremo (início) conforme a política *First-in, First-out* (FIFO). Modelos intuitivos de filas são as linhas para comprar bilhetes de cinema e de caixa de supermercado.

A seguir, são elencados alguns exemplos de aplicações de filas:

- Filas de espera e algoritmos de simulação;
- Controle por parte do sistema operacional a recursos compartilhados, tais como impressoras;
- *Buffers* de Entrada/Saída;
- Estrutura de dados auxiliar em alguns algoritmos como a busca em largura.

As principais operações desta estrutura de dados são:

- enfileirar(F,x): insere o elemento x no final da fila F. Retorna true se foi possível inserir false caso contrário
- desenfileirar(F): remove o elemento no inicio de F, e retorna esse elemento. Retorna null se não foi possível remover.

Operações auxiliares são elencadas como se seguem:

- frente(F): retorna o elemento no início de F, sem remover
- tamanho(F): retorna o número de elementos em F
- vazia(F): indica se a fila F está vazia
- cheia(F): indica se a fila F está cheia (útil para implementações estáticas)

Uma maneira simples de implementar uma fila é fixar o início da fila na primeira posição do vetor. As inserções (enfileira) fazem com que o contador de inserções seja incrementado (O(1)), mas as remoções (desenfileira) requerem remanejar todos os elementos (O(n)) pelo vetor de dados.

4 Lista

Listas são estruturas flexíveis que admitem as operações de inserção, remoção e recuperação de itens. É uma sequencia de zero ou mais itens $l = \{x_1, x_2, ..., x_n\}$ na qual x_i é de um determinado tipo e n representa o tamanho da lista. De forma geral as posições relativas dos itens na lista não tem importância ou significado para os dados.

Diversos tipos de aplicações requerem uma ED do tipo lista. Eis alguns exemplos:

- Lista de pedidos;
- Lista de contatos;
- Lista de tarefas.

Eis as operações básicas de uma lista:

- Criar lista: inicia a estrutura de dados.
- Inserir item: insere um item na lista, retorna *true* se a operação foi executada com sucesso, *false* caso contrário.
- Remover item (dado uma chave): remove um determinado item da lista dado uma chave, retorna *true* se a operação foi executada com sucesso, *false* caso contrário;
- Recuperar item (dado uma chave): recupera o item dado uma chave, retorna o item se a operação foi executada com sucesso, *null* caso contrário
- Contar número de itens: retorna o número de itens na lista;
- Verificar se a lista está vazia: retorna *true* se a lista estiver vazia e *false* caso-contrário;

- Verificar se a lista está cheia: retorna *true* se a lista estiver cheia e *false* caso-contrário;
- Imprimir lista: imprime na tela os itens da lista.

Um estrutura do tipo lista pode ser implementada de forma estática ou dinâmica. As subseções seguintes descrevem estas formas de implementação.

4.1 Lista estática

Nesta implementação, os itens da lista são armazenados em posições contíguas de memória, ou seja, utiliza-se um *array* comum parar guardar os dados desta estrutura. Desta forma, a quantidade de itens que podem ser inseridos na lista limita-se ao tamanho do vetor.

A lista pode ser percorrida em qualquer direção. A inserção de um novo item na última posição da lista pode ser realizada com complexidade constante. Porém, a inserção de um item em uma posição predefinida pode requerer tempo linear, visto que o vetor de dados precisará ter seus elementos remanejados.

Na implementação estática, todas as operações supracitadas devem ser implementadas.

4.2 Lista dinâmica

Considere a implementação de uma aplicação de carrinho de compras de um super-mercado. A lista de compras não possuí tamanho pre-definido. É comum iniciarmos uma compra de mercado planejando comprar x itens, porém, ao final, acabar comprando y itens. Desta forma, a estrutura de dados utilizada para armazenar os itens do carrinho não deve possuir limitação de tamanho, pois o espaço requerido para armazenar os dados pode aumentar em tempo de execução. Neste caso, pode-se optar por uma lista dinâmica para armazenar os itens do carrinho de compras virtual. O projeto de implementação deste tipo de estrutura de dados utiliza ponteiros e componentes simples chamados nó.

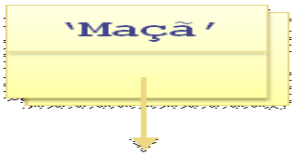

Figure 2: Ilustração de um nó.

Um nó possui uma seta apontando para fora. Essa seta representa um ponteiro que aponta para outro nó, formando uma lista ligada.

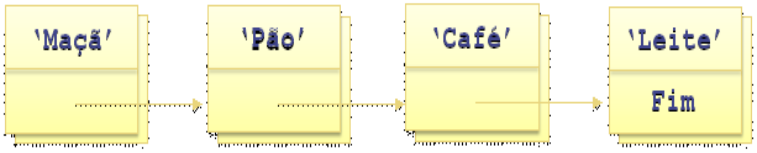

Figure 3: Ilustração de uma sequência de nós.

Este tipo de implementação é útil do ponto de vista do desempenho computacional, pois as operações inserção e remoção no inicio ou no meio da lista podem ser mais eficientes. Uma segunda vantagem é o fato de não ser necessário informar o número de elementos em tempo de compilação.

É importante atentar para o fato de que a porção de memória dinâmica (*heap*) não é ilimitada e é sempre importante verificar se existe memória disponível ao ativar *malloc()*. Em C, o procedimento *malloc()* atribui o valor *NULL* à variável ponteiro quando não existe memória disponível.

Por exemplo, uma operação de remoção pode ser feita da seguinte maneira:

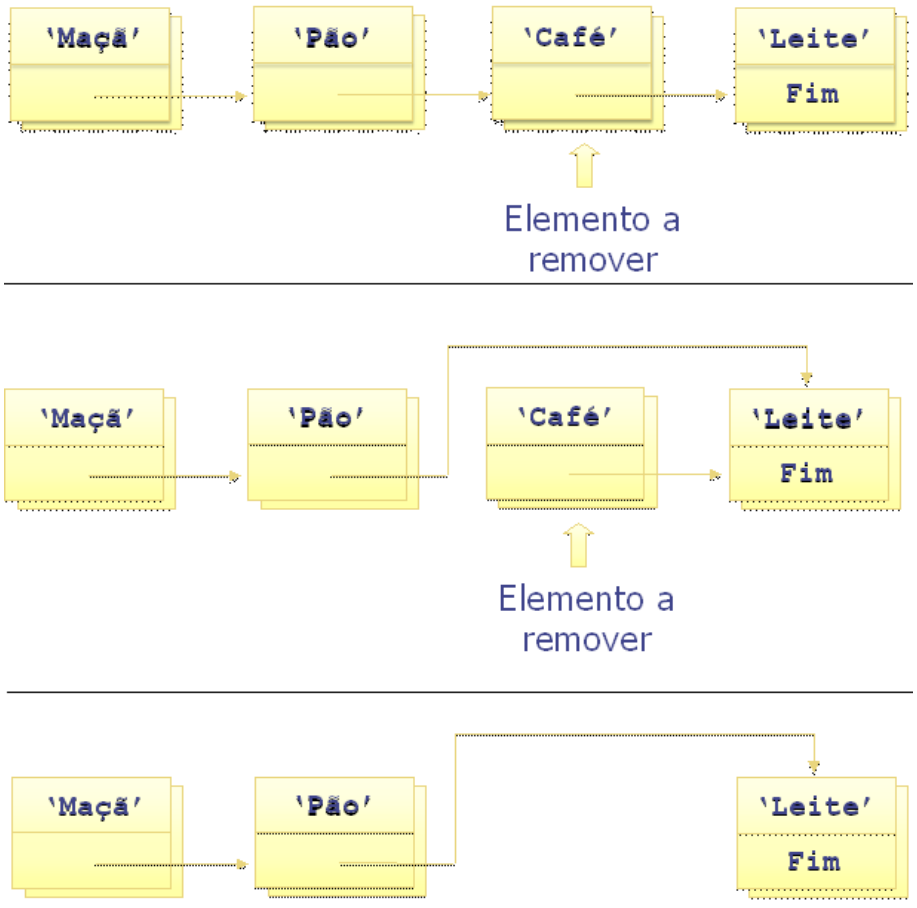

Figure 4: Ilustração da operação de remoção de nós.

Já uma operação de inserção ocorre da seguinte forma:

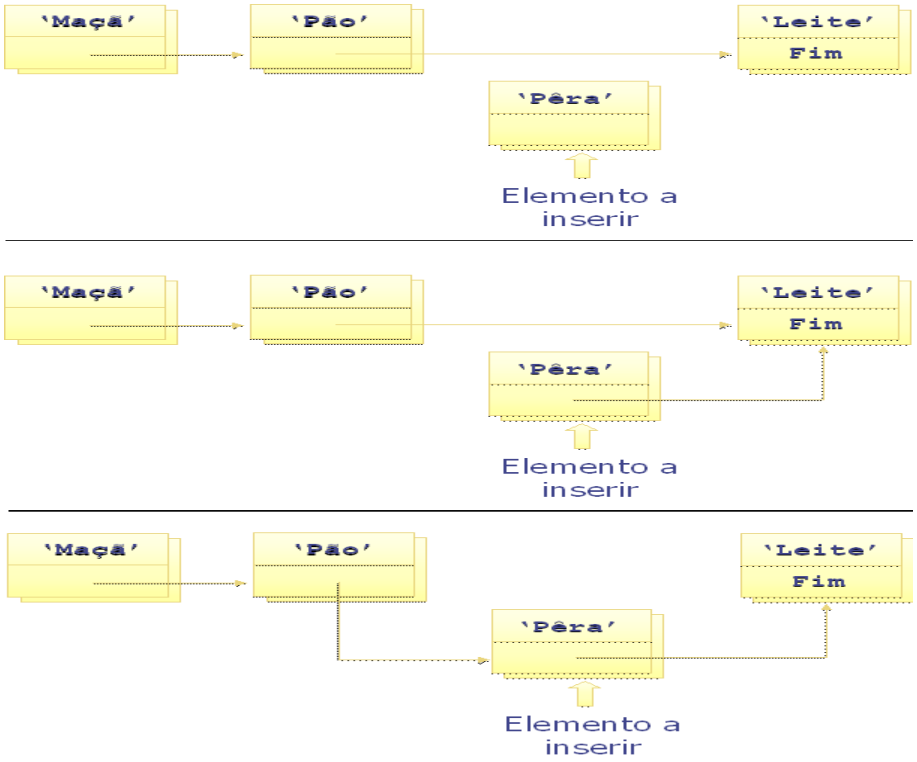

Figure 5: Ilustração da operação de inserção de nós.

As operações básicas de uma lista dinâmica são aquelas já descritas no inicio da seção. Porém, antes de começarmos, precisamos definir como a lista será representada.

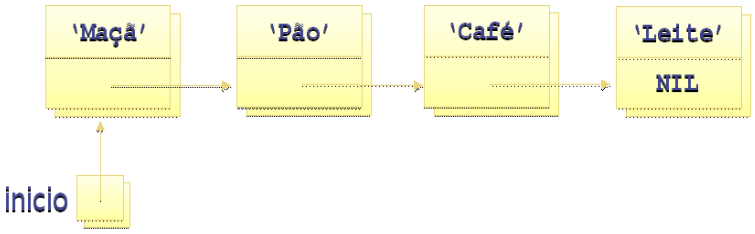

Figure 6: Ilustração da representação de uma lista dinâmica.

Uma forma bastante comum é manter uma variável ponteiro para o primeiro elemento da lista ligada. Convenciona-se que essa variável ponteiro deve ter valor *NULL* quando a lista estiver vazia. Portanto, essa deve ser a iniciação da lista e também a forma de se verificar se ela se encontra vazia.

Outro detalhe importante é quanto as posições. Na implementação com vetores, uma posição é um valor inteiro entre 0 e n. Com listas ligadas, uma posição passa ser um ponteiro que aponta um determinado nó da lista. Desta

forma, o projeto de implementação das operações deste tipo de ED pode ser descrito da seguinte forma:

- Criar lista: inicia a estrutura de dados.
- Inserir item: insere um item na última posição, retorna *true* se a operação foi executada com sucesso, *false* caso contrário;
- Remover item (dado uma chave): remove um determinado item da lista dado uma chave, retorna *true* se a operação foi executada com sucesso, *false* caso contrário;
- Recuperar item (dado uma chave): recupera o item dado uma chave, retorna o item se a operação foi executada com sucesso, *null* caso contrário
- Contar número de itens: retorna o número de itens na lista;
- Verificar se a lista está vazia: retorna *true* se a lista estiver vazia e *false* caso-contrário;
- Imprimir lista: imprime na tela os itens da lista.