Abstração de Dados

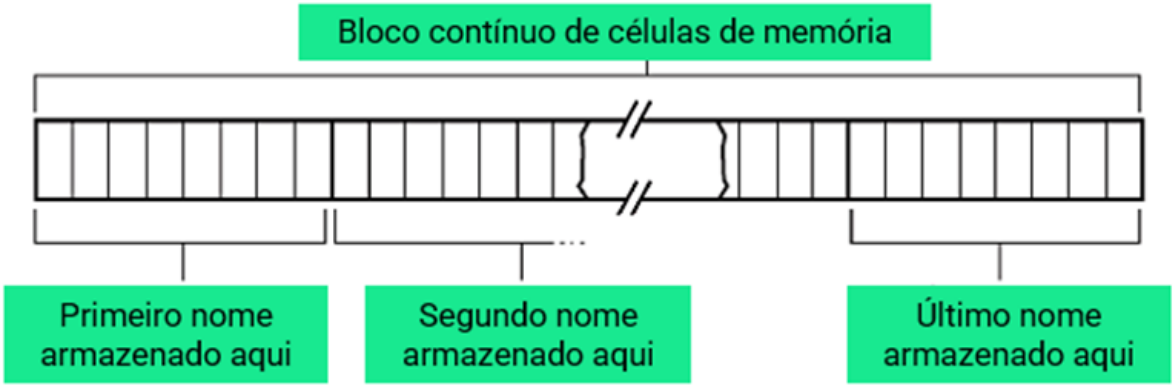
1 – Estruturas de dados básicas

Sabemos que as informações no computador estão armazenadas em células de memória com endereços específicos e muitos detalhes relacionados. Estruturas de dados são formas do cliente (programador, usuário, outro sistema) atuar com o sistema sem ficar preso aos detalhes do armazenamento da informação na memória do computador. Chamamos isso de "abstração de dados" que ganham forma nas estruturas de dados que são decisivas para que o programador expresse os algoritmos de forma mais eficiente.

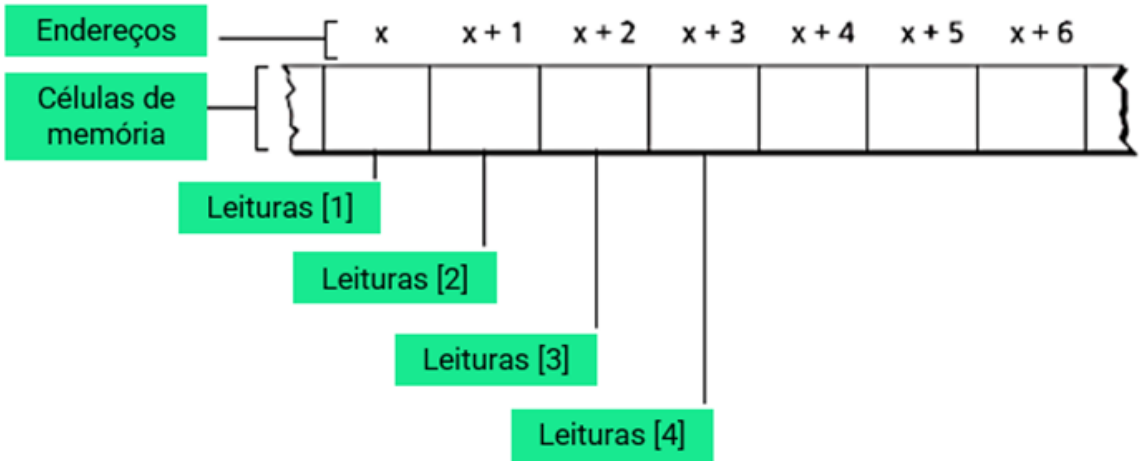
Vetor (array ou Listas contígua)

Uma lista é uma coleção de elementos que aparecem sequencialmente. São exemplos: listas de convidados para uma festa, lista de compras. Nomes também podem ser vistos como listas de letras. Pode-se abstrair uma lista:

Um vetor ou *array* permite acesso direto a qualquer elemento da lista, isto é, você não precisa percorrer todos os elementos de uma lista até encontrar o que você precisa, basta usar o índice da estrutura. Observe a figura abaixo uma parte da memória do computador, as informações estão uma "ao lado" da outra.



Abaixo temos uma representação semelhante.

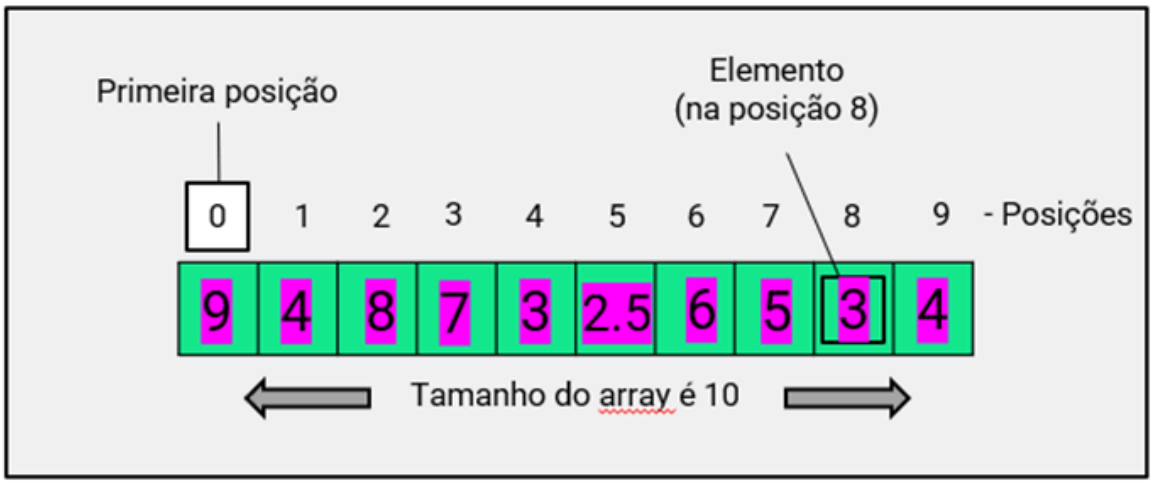


Considere que a lista se chame LEITURAS.

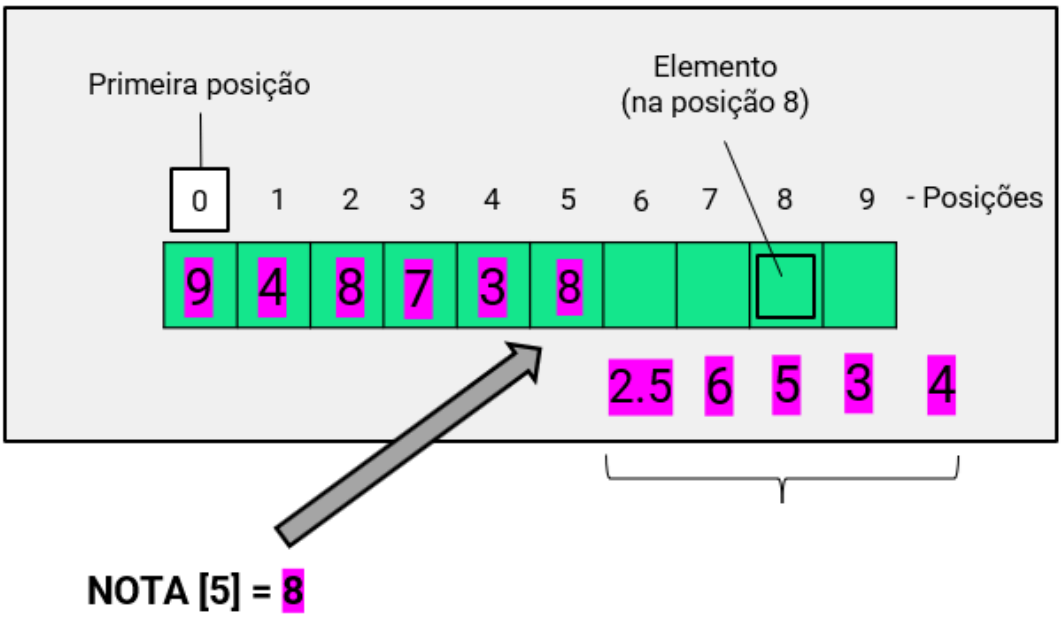
LEITURAS [1] indica o conteúdo na posição 1, LEITURAS [2], a posição 2 e assim por diante.

Essa forma de estrutura de dados em vetor permite acesso direto a uma determinada posição do vetor.

Considere abaixo a posição 5 (índice 5), onde temos o elemento 2.5.



A inserção de um novo elemento irá forçar que os demais elementos sejam deslocados pela estrutura exigindo um laço de programação.

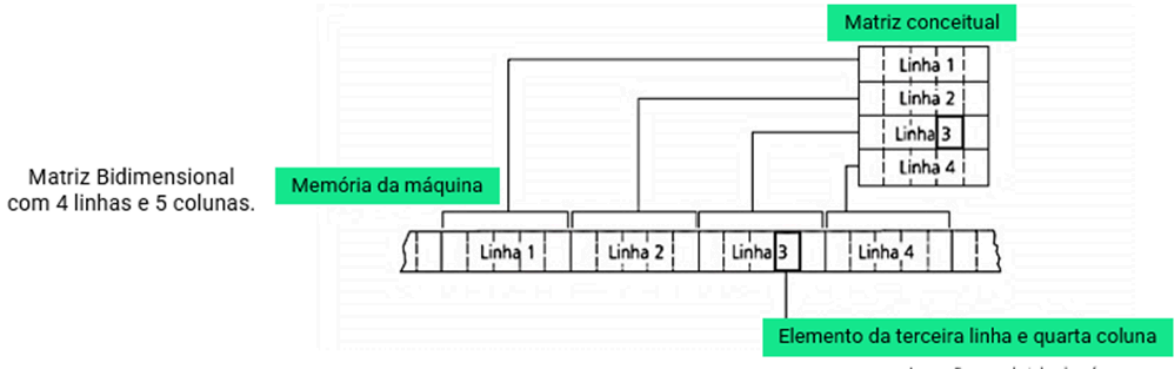


Em outras palavras, a inserção de um elemento no vetor sem copiar por cima de um elemento pré-existente requer movimentar os elementos restantes da lista. Decididamente incluir um elemento em um vetor sem apagar o anterior tem certo custo de programação! Isso é diferente da estrutura lista ligada que veremos mais adiante.

1.2 - Matrizes

Matrizes são estruturas importantes e comuns e consistem de vetores bidimensionais. Considere os dados como se estivesse todos dispostos em uma tabela, isto é, uma estrutura retangular contendo elementos de um mesmo tipo (no caso de matriz homogênea). Pense em uma forma de representar a ocupação de lugares em um teatro. A cada venda de um bilhete é preciso armazenar a informação evitando que duas pessoas tenham o mesmo número de assentos. Observe a praticidade de se marcar assento ocupado. Imagine que uma pessoa tenha reservado o assento 7 na 4ª. fileira.

Simplificando bastante o processo, considerando uma estrutura de dados chamada LUGARES contendo a representação de todos os assentos do teatro contendo 20 fileiras de bancos e 15 assentos em cada fileira, temos a representação abaixo:



Marcar o assento do assento 7 da 4ª. fileira poderia ser algo assim:

LUGARES [4][7] = 1; /* repare que a posição da poltrona linha 4, coluna 7 foi marcada */

1.3 – Listas ligadas

Se o armazenamento dos elementos da lista anterior estiver em blocos de letras espalhados pela memória e não "colados uns aos outros", temos uma **lista ligada.**

Uma lista ligada tem importante diferença com relação a vetores porque geralmente será utilizada quando o tamanho da memória do computador a ser alocado não é conhecido *a priori* e mais memória deve ser alocada dinamicamente, isto é, durante a execução do próprio programa.

Cada elemento da lista, neste caso o nome de uma pessoa, precisa ter um endereço apontando para o próximo elemento da lista. A lista ligada precisa de um ponteiro apontando para o início da lista e um ponteiro no final da lista apontando para o seu final.

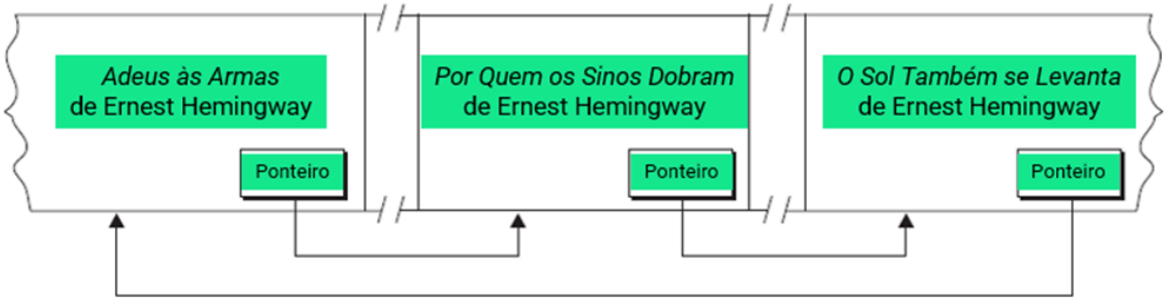
Importante perceber as diferenças nos métodos de busca e adição e eliminação de elementos em uma lista ligada. Depende somente da mudança de endereços de apontadores!

Uma lista ligada não permite acesso direto a qualquer elemento da lista (como em um vetor), isto é, você precisa percorrer todos os elementos de uma lista até encontrar o que você precisa.

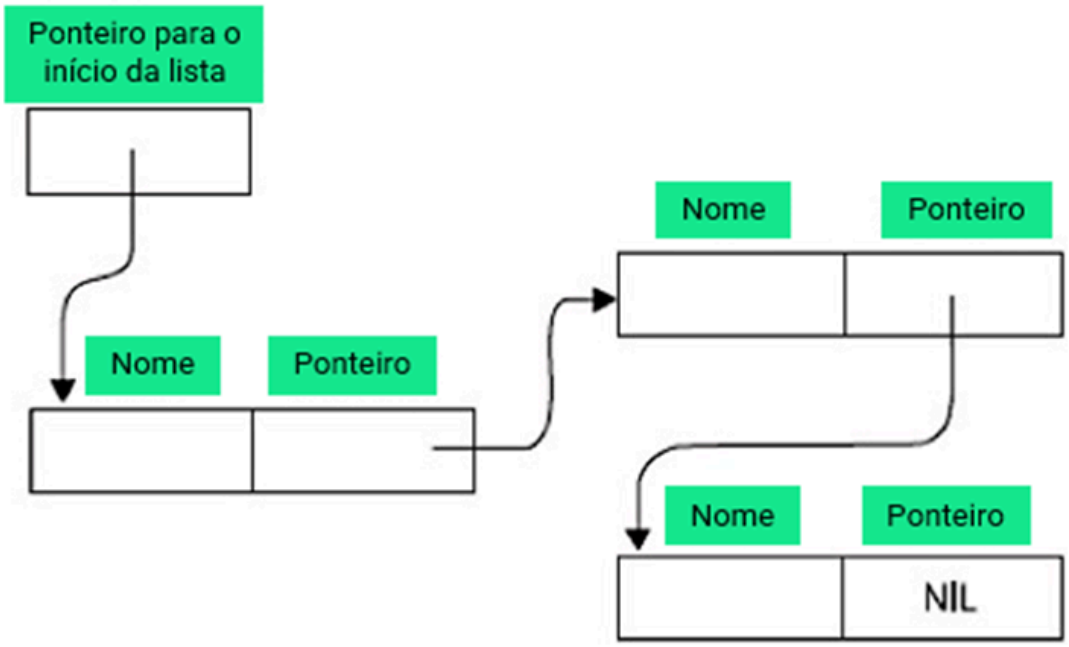
Listas ligadas são muito usadas quando não se sabe o tamanho da memória requerida para uma determinada aplicação.

Assim, a lista é preferível ao vetor (lista contígua) quando houver necessidade de inserir e remover elementos. Já em um vetor, é necessário fazer muitos deslocamentos de nomes se for necessário preencher a lista com novos elementos. Mas a vantagem do vetor é o acesso direto, lembre-se!

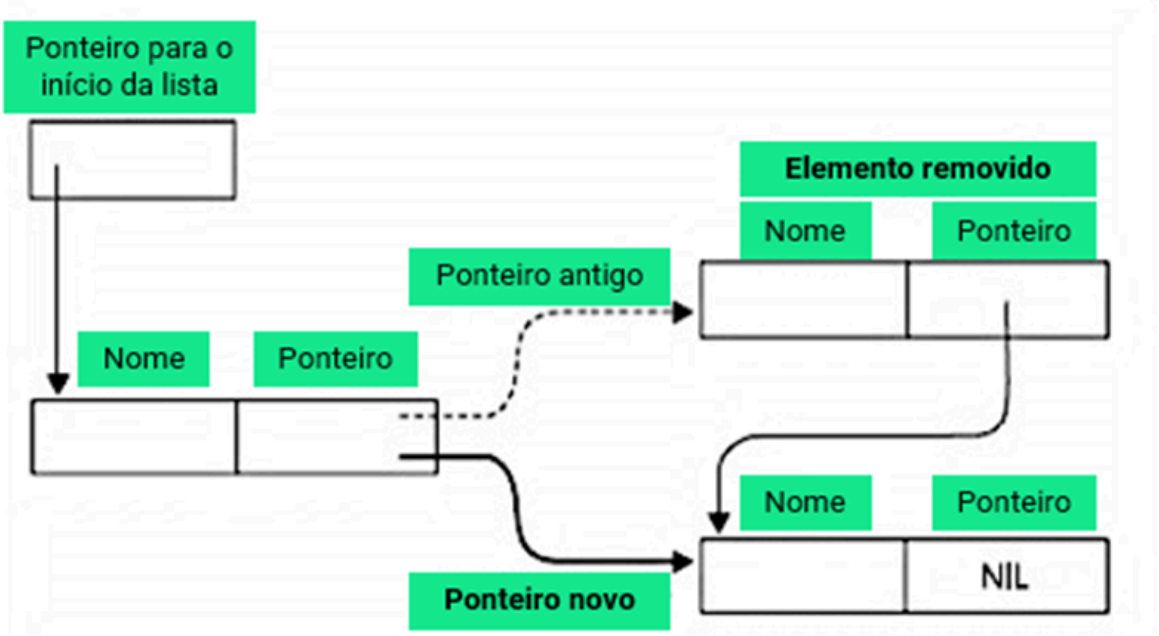
Repare abaixo a movimentação de ponteiros (dica: havendo ponteiros, é lista ligada, havendo índice, é vetor).



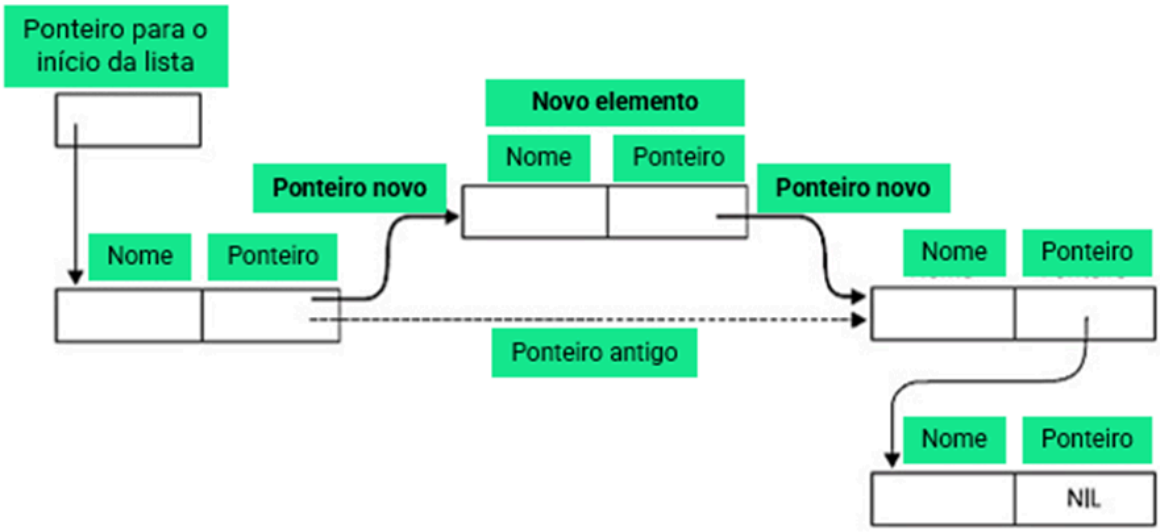
Visto de outra forma, a lista tem um início e cada elemento aponta para a localização do elemento seguinte (essa é a ideia do ponteiro representado por uma seta).



Quando queremos eliminar um elemento da lista, podemos simplesmente desviar a posição do ponteiro e nenhum elemento aponta para o elemento a ser eliminado. Isso equivale à sua eliminação.



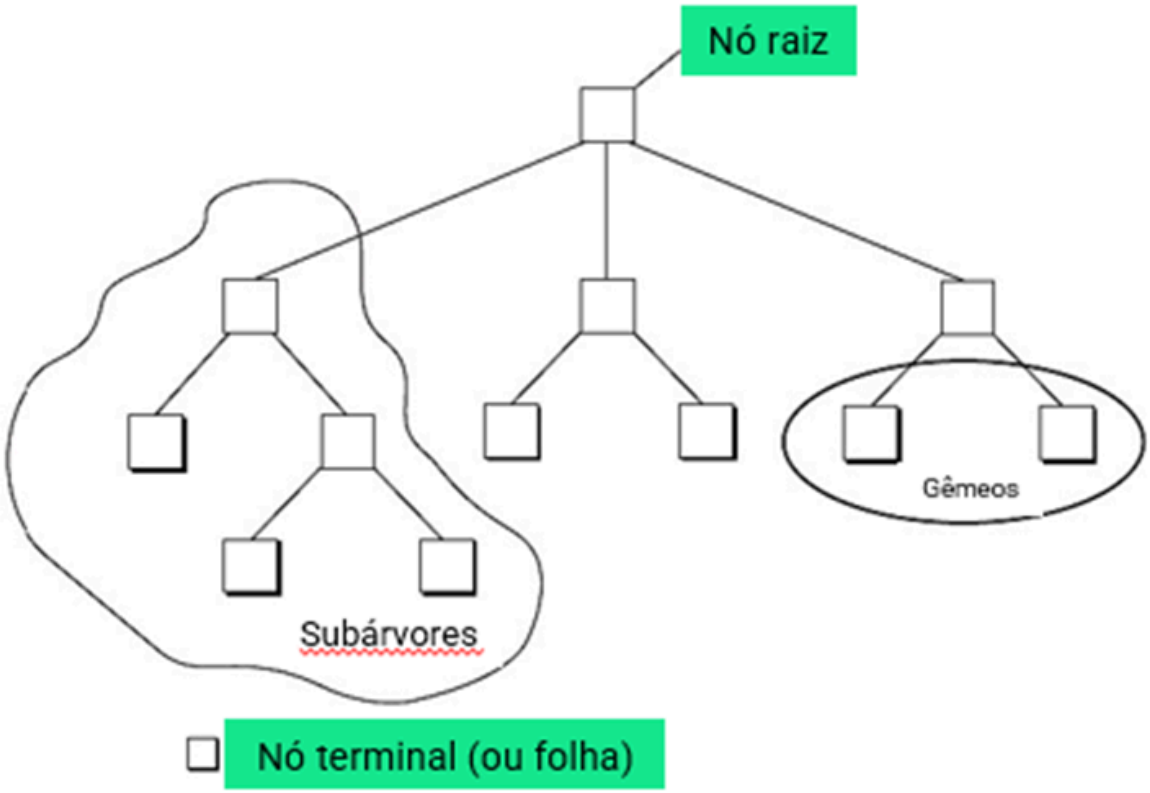
A inclusão de um elemento na lista mais uma vez significa ajuste de ponteiros. O novo elemento deve apontar para seu sucessor.



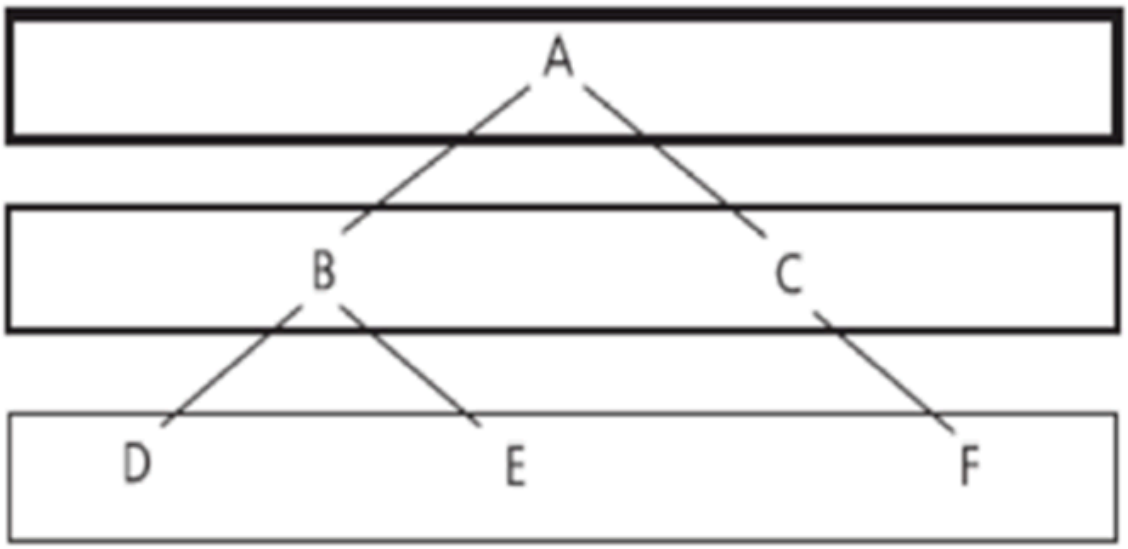
1.4 – Árvore

Em uma **árvore**, a organização dos elementos se dá de forma hierárquica, existindo um elemento que fica no topo da árvore, chamado de **raiz** e os elementos subordinados a ele, seus nós filhos.

Cada nó filho pode conter zero, um ou mais de um nó filho.

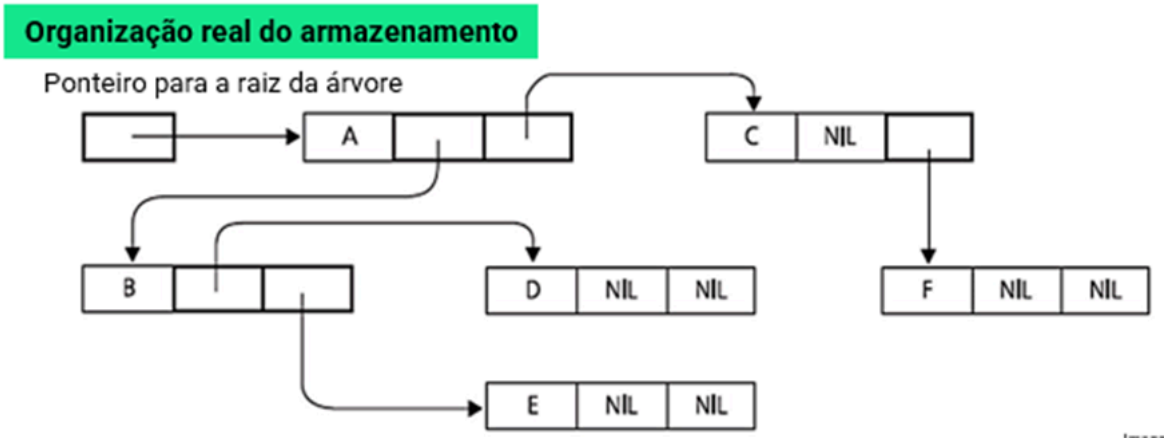


Olhando para uma árvore por camadas:



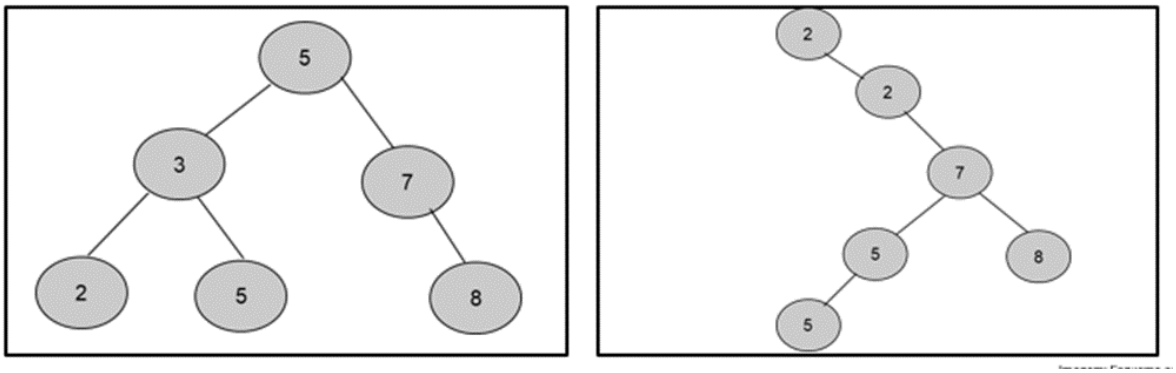
Note que a implementação da árvore se dá por lista ligada mas poderia ser por vetores também. Isto é uma lista ou um vetor pode representar uma árvore, embora a representação por listas seja mais fácil de se encontrar em algoritmos.

O NIL representa um ponteiro que não aponta para outro elemento.



Uma **árvore binária** é uma estrutura de dados específica, útil quando precisam ser tomadas decisões bidirecionais em cada ponto de um processo. Note que em uma árvore binária, à esquerda de um nó pai, os filhos da subárvore esquerda são menores e os filhos da subárvore direita são maiores.

Um exemplo de aplicação: encontrar todas as repetições em uma lista de números, verificar se um dado elemento está presente na árvore ou não está, qual o menor ou qual o maior elemento de uma lista.



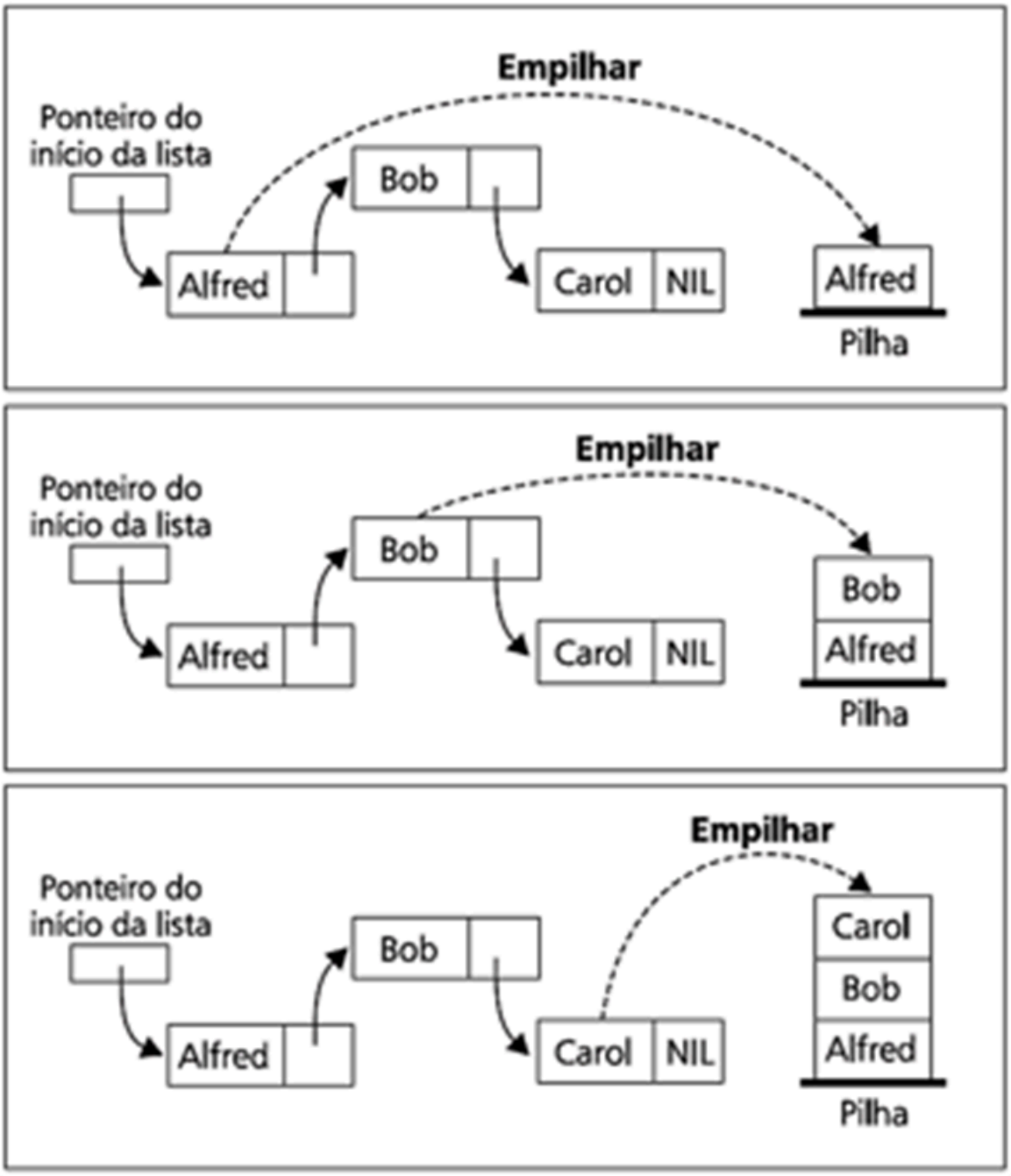
2 - Outros tipos de estruturas importantes

Pilhas

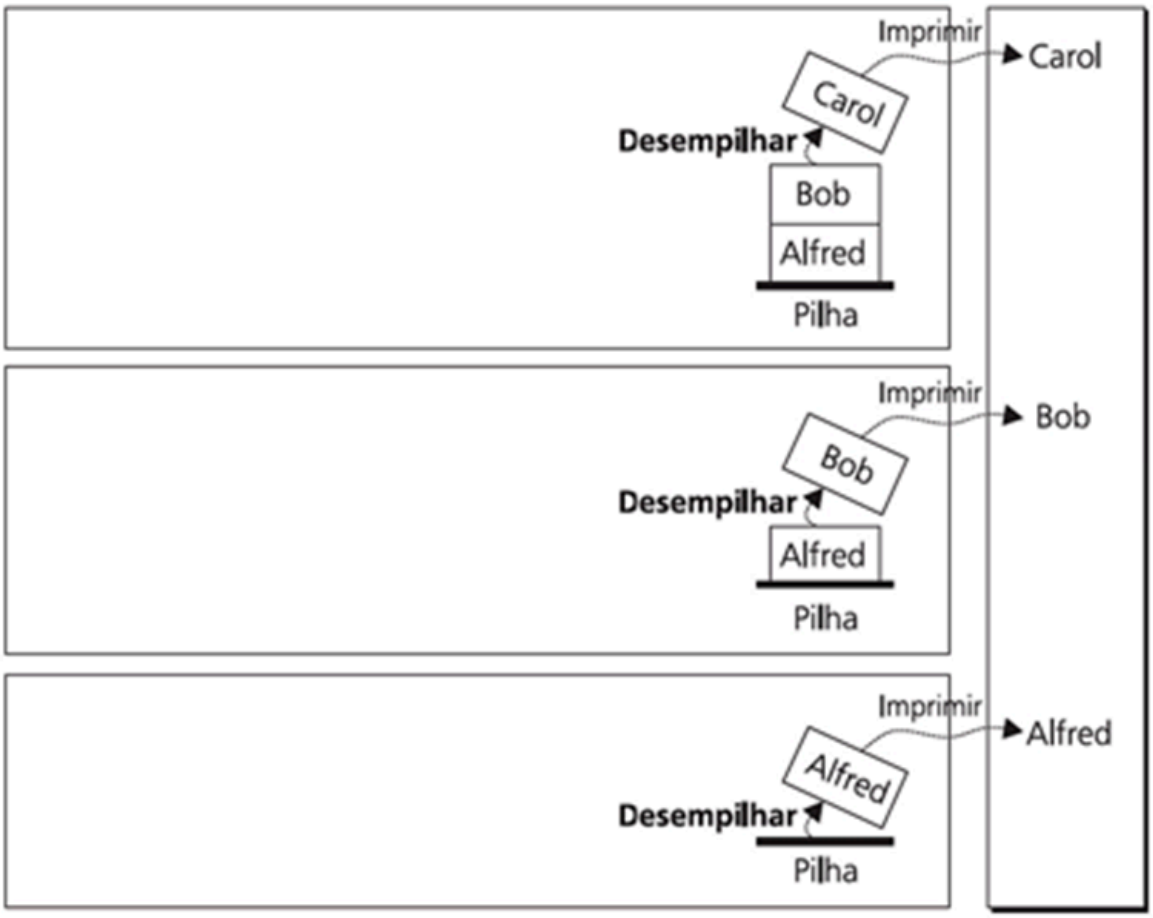
Quando operamos uma estrutura em que as operações de inserção ou remoção ocorrem só em um dos extremos da lista, chamamos essa estrutura de uma **pilha.**

A consequência dessa restrição é que o último elemento a entrar na estrutura é o primeiro a ser removido, isso leva as pilhas a serem conhecidas como estruturas "last-in, first-out – **LIFO**". A extremidade em que as operações ocorrem é chamada de topo da pilha, a outra é chamada de base da pilha. O processo de inserir um objeto no topo da pilha é chamado de empilhamento (**push**), o de remover, operação de desempilhamento (**pop**).

Veja na ilustração abaixo como os elementos de uma lista podem ser copiados a fim de formar uma pilha.



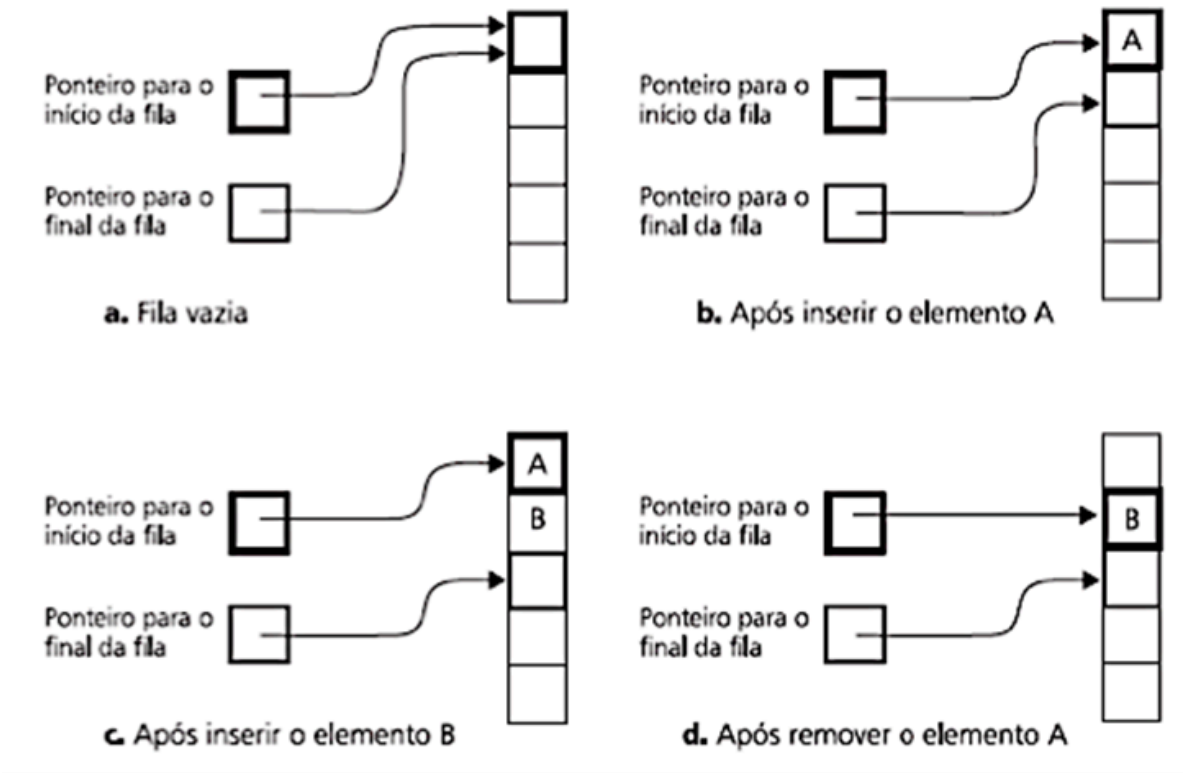
Uma vez criada a pilha, seus elementos podem ser descarregados mas sempre a partir do topo da pilha, conforme a ilustração a seguir:



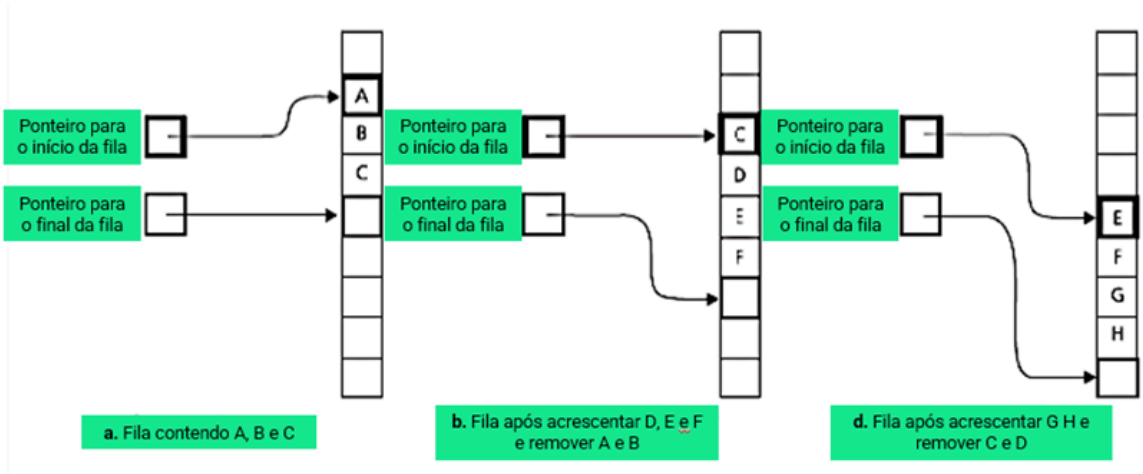
2.2 - Filas

Em uma fila, as inserções são realizadas em uma extremidade e as remoções na outra. É o esquema first-in, first-out – **FIFO.** Outra forma de considerar fila é pensar que nela os objetos são retirados de acordo com a ordem com que chegaram. A extremidade na qual são retirados os elementos é chamada de início da fila, onde eles são inseridos é chamado de fim da fila.

Abaixo uma fila em diferentes momentos de processamento:



Outro exemplo da dinâmica de uma fila:

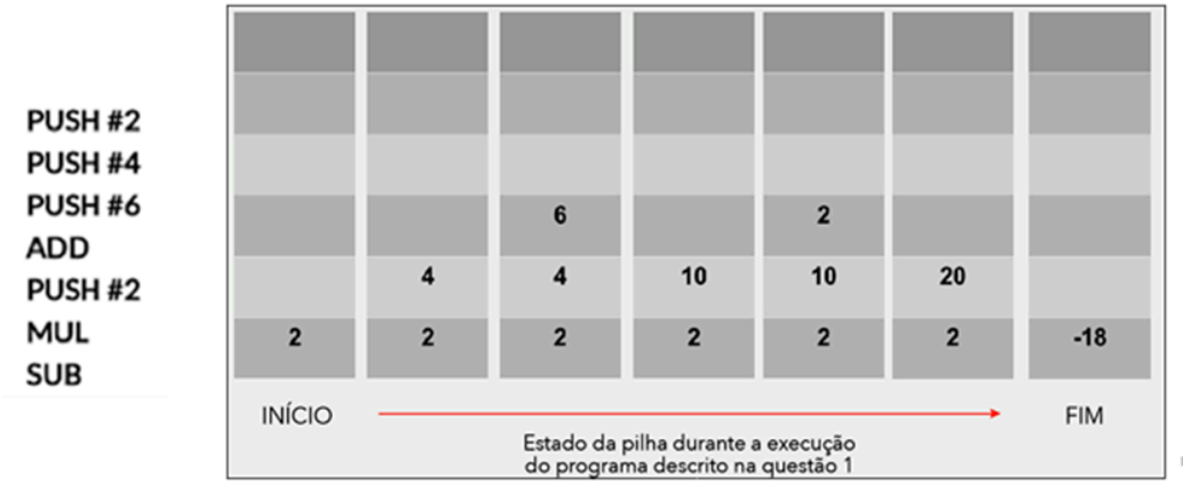


3 - Um exemplo de aplicação

Observe como as instruções são executadas, de forma ilustrativa, na memória do computador.

A instrução PUSH deposita um valor no topo da pilha, a instrução POP desempilha um valor do topo da pilha, ADD significa adicionar o topo ao vizinho imediato, de

forma análoga MUL é multiplicar e SUB é subtrair.



Ao término das execuções, o topo da pilha contém somente o número 18.

Referência Bibliográfica

BROOKSHEAR, J.G. **Ciência da Computação: uma visão abrangente.** Porto Alegre: Bookman, 2013.

DROSDEK, A. **Estruturas de Dados e Algoritmos em C++.** São Paulo: CENGAGE, 2018.

FORBELLONE, A.L.V. & EBERSPACHER, H. F. **Lógica de Programação – A Construção de Algoritmos e Estruturas de Dados.** 3ª. Edição. São Paulo, SP: Prentice Hall, 2005.

Ir para exercício