ATIVIDADE DE REVISÃO

- **ARRAYS**

Arrays, ou vetores, são estruturas usadas para guardar vários valores do mesmo tipo em posições organizadas. Cada elemento é acessado por um índice que começa em zero, o que torna o acesso muito rápido. Porém, uma limitação dos arrays é que seu tamanho é fixo: depois que você cria, não pode aumentar ou diminuir a quantidade de posições. Quando precisamos armazenar dados de forma sequencial e sabemos exatamente quantos elementos teremos, os arrays funcionam muito bem.

- **MATRIZES**

As matrizes são parecidas com arrays, mas possuem duas dimensões (por isso também são chamadas de arrays bidimensionais), funcionando como uma tabela com linhas e colunas. Podemos imaginar uma matriz como um array que contém outros arrays dentro. Elas são úteis quando precisamos representar informações estruturadas, como mapas, tabelas numéricas, notas de alunos e até imagens. Assim como os arrays simples, as matrizes também costumam ter tamanho fixo, o que exige que a gente saiba antecipadamente a quantidade de linhas e colunas necessárias.

- **LISTAS**

Listas são estruturas mais flexíveis do que arrays. Elas podem crescer ou diminuir conforme adicionamos ou removemos elementos. Dependendo do tipo de lista (como listas encadeadas), a inserção e remoção podem ser muito rápidas, já que basta ajustar ligações entre elementos. Por outro lado, acessar um elemento específico pode ser mais lento do que em um array, porque em algumas listas é necessário percorrer os itens até chegar ao desejado. Listas são muito usadas quando não sabemos de antemão quantos dados precisaremos armazenar.

- **PILHAS**

Pilhas, ou stacks, seguem a lógica de LIFO, que significa “Last In, First Out”: o último elemento a entrar é sempre o primeiro a sair. A forma mais fácil de imaginar uma pilha é como uma pilha de pratos: você só mexe no que está no topo. As operações principais são o push, que adiciona um novo elemento no topo, e o pop, que remove o último elemento colocado. Pilhas são usadas em

várias situações, como desfazer ações em editores, controlar chamadas de funções e percorrer árvores.

- **FILAS**

As filas, ou queues, funcionam de maneira oposta às pilhas, seguindo a lógica FIFO: “First In, First Out”. O primeiro elemento que entra é o primeiro a sair, como acontece em uma fila de pessoas no supermercado. Os dois comandos mais comuns são enqueue (inserir no final) e dequeue (remover do início). Filas são muito úteis para controlar processos que precisam respeitar ordem de chegada, como tarefas na impressora ou requisições em sistemas.

- **MAPAS**

Mapas, também chamados de dicionários, ou dictionarys, são estruturas que armazenam dados no formato chave–valor. Em vez de buscar um item pela posição, como em arrays, você o encontra pela chave, que é única. Isso permite acessar valores de forma rápida e intuitiva. Um exemplo simples seria armazenar o nome de uma pessoa usando o CPF dela como chave.

- **TABELAS HASH**

As tabelas hash, ou hash tables, são uma forma muito eficiente de implementar mapas. Elas usam uma função hash para transformar a chave em um número, que é usado como índice para armazenar o valor internamente. Isso faz com que a busca, inserção e remoção sejam, na maioria das vezes, muito rápidas. Porém, às vezes duas chaves diferentes podem gerar o mesmo hash, o que é chamado de colisão. Para lidar com isso, existem métodos como listas encadeadas ou endereçamento aberto. Mesmo com esse problema, tabelas hash continuam sendo uma das estruturas mais rápidas para buscas.

- **ÁRVORES**

As árvores, ou trees, são estruturas organizadas em forma hierárquica. Cada elemento é chamado de nó, e pode ter “filhos”. O primeiro nó é chamado de raiz. Existem vários tipos de árvores, como as árvores binárias, onde cada nó pode ter até dois filhos. Árvores são muito úteis para organizar informações e realizar buscas rápidas. Elas aparecem em diversos lugares, como nos diretórios do computador, algoritmos de busca e até em bancos de dados

- **ALGORÍTMOS DE ORDENAÇÃO**

Algoritmos de ordenação são métodos usados para organizar uma lista de dados em uma determinada ordem, geralmente crescente ou decrescente. Eles são importantes porque muitos outros algoritmos funcionam melhor quando os dados já estão ordenados. Existem vários tipos de algoritmos de ordenação, cada um com suas vantagens e desvantagens. Um dos mais simples é o Bubble Sort, que compara pares de elementos e troca suas posições sempre que encontra algo fora da ordem. Embora seja fácil de entender, ele é lento para listas grandes. O Selection Sort funciona procurando o menor elemento de toda a lista e colocando-o na primeira posição, depois o segundo menor na segunda posição, e assim por diante. Ele também é simples, mas não é muito eficiente. Outro algoritmo básico é o Insertion Sort, que funciona como se você estivesse organizando cartas na mão: ele pega um elemento e o coloca na posição correta em relação aos elementos anteriores.

Existem também algoritmos mais avançados e muito mais rápidos, como o Merge Sort, que divide a lista em partes menores, ordena cada parte e depois junta tudo de volta. O QuickSort também divide a lista, mas escolhe um elemento chamado pivô para organizar os outros valores em dois grupos: os menores e os maiores que ele. Isso costuma ser bem eficiente. Em geral, esses algoritmos mais modernos têm desempenho muito melhor em listas grandes, porque reduzem bastante a quantidade de comparações e trocas. Apesar das diferenças, o objetivo de todos os algoritmos de ordenação é o mesmo: reorganizar os dados de forma rápida e correta.

- **ALGORÍTMOS DE BUSCA**

Algoritmos de busca são usados para encontrar um elemento específico dentro de uma estrutura de dados, como um array, lista ou árvore. O algoritmo mais simples é a Busca Linear, na qual percorremos a lista do início ao fim, verificando item por item até encontrar o valor desejado. Ela funciona em qualquer tipo de lista, mas pode ser lenta quando o número de elementos é grande, já que pode ser necessário verificar todos eles. Outro método muito mais rápido é a Busca Binária, mas ela só funciona em listas ordenadas. Nesse tipo de busca, em vez de olhar todos os itens, dividimos a lista ao meio repetidamente. Primeiro olhamos o elemento central: se ele for o valor procurado, o algoritmo termina; se o valor buscado for menor, buscamos na metade esquerda; se for maior, buscamos na metade direita. Esse processo se repete até encontrar o elemento ou até não sobrar mais nada para procurar.

Existem também buscas específicas para outras estruturas. Por exemplo, em árvores binárias de busca, cada comparação já indica se devemos ir para o filho esquerdo ou direito, o que torna a busca muito eficiente se a árvore estiver equilibrada. Em tabelas hash, a busca é ainda mais rápida, porque a função hash

leva diretamente à posição provável do elemento. Em todos os casos, o objetivo dos algoritmos de busca é o mesmo: localizar algo da maneira mais eficiente possível, evitando percorrer toda a estrutura quando existe uma forma mais inteligente de chegar ao resultado.

Aluno: Lucas Piva Bellaver