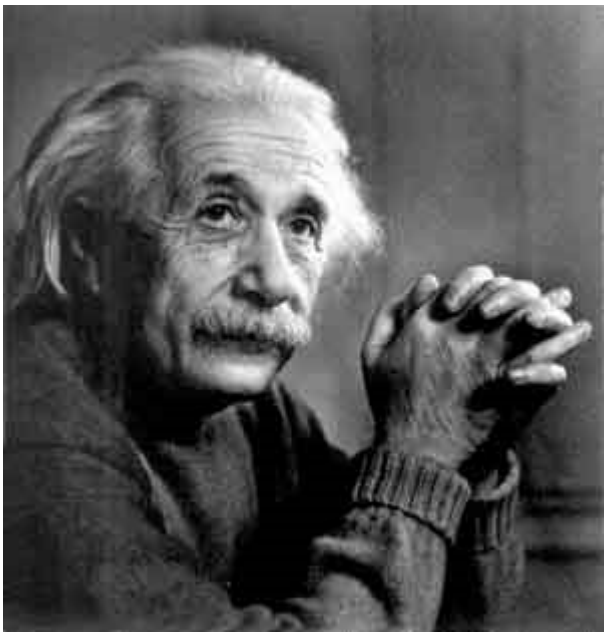
ESTRUTURA DE DADOS

Algoritmos são desenvolvidos para resolver problemas, assim expressam soluções computacionais que atendem a necessidades específicas. É desejável que os algoritmos façam um eficiente uso dos recursos computacionais (memória, armazenamento e processamento), para isso é necessário fazer a adequada escolha e implementação das estruturas de dados. Isto é importante, pois a forma como os dados estão organizados durante o processamento fazem com que o sistema tenha uma melhor ou pior performance. As estruturas de dados definem a organização dos dados e as funcionalidades de acesso e manipulação desses. Assim, o principal objetivo deste material é apresentar as estruturas de dados mais utilizadas no desenvolvimento de programas, descrever sua organização de dados, sua lógica de funcionamento e operações de manipulação.

INTRODUÇÃO



(14/03/1879 – 18/04/1955)

" Imagination is more important than knowledge. For knowledge is limited, whereas imagination embraces the entire world, stimulating progress, giving birth to evolution"[1] *(Albert Einstein)*

Você tem ideia de como são organizados os dados dos programas na memória?

Você já se perguntou como se organizam as relações entre os dados?

[1] Tradução: Imaginação é mais importante que conhecimento. Pois o conhecimento é limitado, enquanto a imaginação abraça o mundo inteiro, estimulando o progresso, dando origem à evolução.

A estrutura básica de organização dos dados nos programas se efetiva através das variáveis, que são vinculadas aos tipos de dados (numérico, alfanumérico, booleano, e seus derivados). Cada tipo de dado restringe os possíveis valores que as variáveis/constantes podem receber, além de determinar as operações que podem ser realizadas sobre esses dados.

Vamos analisar juntos alguns exemplos:

(1) Considere que você tem uma variável numérica (inteira) para armazenar a idade de uma pessoa. Você poderá fazer operações de soma, subtração, divisão, multiplicação com essa variável, pois são operações definidas para o tipo de dado inteiro. Além disso, essa variável somente poderá receber números inteiros (neste caso, valores reais – ponto-flutuante – não poderão ser atribuídos à variável, nem dados alfanuméricos - caracteres).

(2) Agora, se você tiver uma variável alfanumérica para guardar um endereço, as operações permitidas serão concatenar strings, buscar um substring, substituir elementos na string, enfim operações definidas para variáveis deste tipo. Essa variável pode receber informação textual (letras, dígitos numéricos e alguns caracteres especiais)

(3) E se você tiver uma variável do tipo booleano para testar se o usuário confirmou a opção de compra, esta variável somente poderá receber os valores Verdadeiro (true) ou Falso (false) e as operações normalmente se restringem às operações booleanas (conjunção “e lógico”, disjunção “ou lógico”, negação “não lógico”, entre outras).

A capacidade de armazenamento (tamanho), extensão dos valores permitidos e operações permitidas para cada tipo de dado podem variar de uma linguagem de programação para outra. Esses tipos de dados disponibilizados pela linguagem são denominados tipos primitivos.

Fonte: https://www.shutterstock.com

Além desses, as linguagens de programação possuem estruturas para definir conjuntos de dados. Esses podem ser homogêneos, que são representados pelos vetores, que são conjuntos de dados em que cada elemento possui um índice de acesso e todos os elementos possuem o mesmo tipo, assim podemos ter vetores com elementos numéricos, alfanuméricos, booleanos e também podem

ser conjuntos. Por exemplo, podemos ter vetores de temperaturas, vetores de idades, vetores de nomes, vetores de dados de pessoas.

Os conjuntos de dados também podem ser heterogêneos, ou seja, formado por diversos dados e cada dado definido com um tipo específico, esses conjuntos são normalmente denominados registros (estruturas). Por exemplo, podemos ter um registro com os dados cadastrais de uma pessoa, contendo código (numérico), nome (alfanumérico), idade (numérico), sexo (alfanumérico), entre outros dados.

Nesta disciplina nós vamos estudar os tipos abstratos de dados, que são tipos de dados definidos pelo usuário (programador) utilizando os tipos primitivos e estruturas de conjuntos de dados disponibilizados pelas linguagens de programação.

Vamos entender o conceito de estrutura de dados?

Estruturas de dados (Data Structures) é a denominação da organização de dados e algoritmos de modo a otimizar o seu uso, dependendo desses elementos pode-se solucionar de forma simples problemas extremamente complexos. Existem diversos modelos de estruturas de dados, e novos modelos são criados constantemente pois acompanham também a evolução dos algoritmos e das linguagens de programação.

Você deve estar se perguntando em quais situações precisará utilizar estruturas de dados!

Na verdade todos programas possuem estruturas de dados, mesmo que sejam variáveis simples! E a seguir vamos analisar alguns exemplos de situações que exigem estruturas de dados específicas!

Considere que você precisa organizar um fichário com as fichas com dados de clientes. A forma mais prática é manter essas fichas em ordem alfabética, pois isso facilita a localização. As operações básicas que serão realizadas nesse fichário serão: incluir uma nova ficha (mantendo a ordenação), excluir fichas, consultar fichas, entre outras.

Neste caso, a estrutura de dados que vamos utilizar é denominada **LISTA** (*LIST*), que provê uma sequência de informações mantendo-as ordenadas por algum dos dados.

Outra situação é organizar as pessoas que precisam ser atendidas em um guichê. A forma mais racional de resolver essa situação é organizar as pessoas em fila, sendo que novas pessoas ficam no fim da fila e será atendido primeiro quem estiver no início da fila. As operações básicas serão: incluir nova pessoa na fila, consultar posição da fila, excluir pessoa da fila (em caso de alguém desistir).

Fonte: https://www.shutterstock.com

A estrutura de dados que utilizaremos tem o mesmo nome, chama-se **FILA** (*QUEUE*), que provê uma sequência de informações que são ordenadas pela ordem de entrada das informações na FILA. O critério de ordenação de uma FILA denomina-se **FIFO** (First-In-First-Out, o primeiro a entrar é o primeiro a sair).

Fonte: https://www.shutterstock.com

Agora, considere que você precisa organizar os pratos de um restaurante, a melhor forma é dispô-los em pilhas. Assim, o primeiro prato fica na base da pilha e o último prato é sempre colocado no topo da pilha. Para usar os pratos, pegaremos primeiro aquele que estiver no topo da pilha e assim subsequentemente, pois se tentarmos puxar o prato da base derrubaremos a pilha. As operações básicas serão colocar pratos na pilha, tirar pratos da pilha, consultar tamanho da pilha, entre outras.

Utilizaremos aqui uma estrutura de dados que se chama **PILHA** (*STACK*) que é uma estrutura sequencial que segue o critério **LIFO** de ordenação dos elementos (Last-In-First-Out, o último a entrar é o primeiro a sair).

Vamos analisar outra situação: em uma organização empresarial se deseja saber a estrutura dos cargos e as pessoas que ocupam cada um deles, considerando que cada cargo somente tem um superior direto. Nessa situação se constrói o organograma da empresa. As operações básicas serão a inclusão de novos cargos, a relação hierárquica de um cargo, a vinculação de uma pessoa a um cargo, a exclusão de um cargo, entre outras.

Fonte: https://www.shutterstock.com

Para atender a essa necessidade utilizaremos uma estrutura de dados hierárquica denominada **ÁRVORE** (*TREE*), que possui uma origem (raiz, no exemplo o cargo mais alto da organização) e todos os outros elementos são organizados hierarquicamente "abaixo" deste.

Fonte: https://www.shutterstock.com

Agora, se tivermos que organizar um trajeto para percorrer todos os pontos turísticos de uma cidade. Teríamos que consultar um mapa e estabelecer as possíveis rotas e depois escolher a melhor rota. As operações possíveis seriam indicar os destinos, calcular os tempos e distâncias para ir de um ponto a outro, calcular rotas alternativas, etc.

A representação de mapas e informações geográficas são normalmente construídas através de **GRAFOS** (*GRAPH*), que são estruturas genéricas que permitem organizar diversos elementos, estabelecendo relações entre eles, aos pares.

Esses exemplos de aplicações referem-se às estruturas de dados básicas: Pilhas, Filas, Listas, Árvores e Grafos, que são o foco de estudo desta disciplina.

Percebas que a maioria das estruturas de dados tem por princípio alguma forma de ordenação dos dados, principalmente porque isso facilita as consultas. Assim, também estudaremos diversos métodos de ordenação de dados.

Temos certeza que você gostará de aprender o funcionamento e o desenvolvimento computacional das estruturas de dados!

Amplie seus conhecimentos!

Pense (pesquise) outras aplicações que podem ser solucionadas com as estruturas de dados que estudaremos nesta disciplina: PILHAS, FILAS, LISTAS, ÁRVORES e GRAFOS.

1 TIPOS ABSTRATOS DE DADOS (TAD)

Você já se deu conta que muitos termos da informática foram tirados no mundo real e adaptados ao contexto computacional?

Isso ocorre porque a informática busca imitar, simular, replicar o mundo real. Porém devido as limitações tecnológicas o que se faz é uma abstração da nossa realidade para que possamos capturar o que existe de mais relevante em uma situação real, possibilitando a construção de modelos que constituem soluções computacionais que possam ser implementadas.

Uma abstração é uma representação que construímos de uma determinada realidade. É desejável que sejam construídos bons modelos e para tanto é preciso conhecer bem os detalhes que formam a situação real e expressá-los através de uma estrutura de dados adequada, além de implementar os algoritmos necessários para sua manipulação.

Isso nos leva ao conceito de **Tipo Abstrato de Dados** (TAD), que é um i que podem ser utilizados para manipulação desses dados, tudo isso para atender uma necessidade específica.

Vamos entender melhor esse conceito de TAD!

Suponha que criamos uma variável do tipo ferramenta, ela poderia assumir os valores serrote, martelo, alicate, etc., e poderíamos fazer as seguintes operações com esta variável: comprar, afiar, utilizar, vender, etc. Este seria o nosso TAD ferramentas.

Ao definir um TAD não se leva em conta detalhes de implementação. Pode ser até que não seja possível implementar um TAD numa determinada linguagem, mas deve-se saber para que serve, que informações contém e quais operações/operadores podem ser utilizados.

Quando devemos identificar e projetar os Tipos Abstratos de Dados?

De forma resumida, podemos dizer que o **ciclo de vida de desenvolvimento de software** engloba as etapas de análise, projeto, implementação, teste e manutenção. Na etapa de **análise**, nós iremos determinar o que o sistema deverá fazer, qual seu foco, quais informações irá manipular, etc. Já na etapa de **projeto**, nós definiremos como o sistema será desenvolvido. Aqui a ideia é especificar uma definição geral, não os detalhe de desenvolvimento. Nessa etapa, decidimos a arquitetura do software, organizamos seus principais módulos, a partir disso podemos organizar a equipe de programadores para sua implementação.

As outras etapas são relativas à implementação: codificação em uma linguagem de programação; testes, para garantir que o software funcione adequadamente; e manutenção, que se refere aos ajustes que podem ser necessários a tempo de uso.

Assim, os TAD serão especificados (identificados e projetados) na etapa de projeto do software, ou seja, quando construímos um visão geral de como o software deverá ser desenvolvido, sem preocupar-se com detalhes de implementação. Tenha em mente que o **princípio básico do Tipo Abstrato de Dados** (TAD) é permitir que o programador possa separar o conceito (aquilo que o TAD deve ser e fazer) de sua implementação (detalhes de como deve ser desenvolvido, recursos, etc.). Essa separação possibilita que sejam realizadas mudanças na implementação e essas não alterarão o programa que utiliza o TDA.

Agora reflita: por que devemos utilizar Tipos Abstratos de Dados?

Podemos indicar algumas vantagens:

- É mais fácil programar, quando não é necessário se preocupar com detalhes de implementação.
- É mais fácil garantir a integridade dos dados, pois apenas as operações do TAD podem alterar os dados armazenados.
- Maior independência e portabilidade de código, pois alterações na implementação de um TAD não provocam alterações nas aplicações.
- Maior potencial de reutilização de código, pois alterações na lógica de um programa não implicam na reconstrução das estruturas de armazenamento.

Pilhas, Filas, Listas, Árvores e Grafos são tipos abstratos de dados!

A utilização de TAD está relacionada com o desenvolvimento de bons programas! Mas o que significa desenvolver um bom programa? É suficiente "funcionar"?

Um "programa" que não funciona, na verdade não é um programa!

Mas para ser um bom programa não é suficiente apenas funcionar! Um bom programa deve ter algumas características que os diferenciarão dos programas ruins:

- Um importante critério que caracteriza bons programas é sua **eficiência**, ou seja, programas que são rápidos e utilizam pouca memória são normalmente considerados melhores. Isso porque normalmente temos restrições de espaço de armazenamento e requisitos de rapidez!
- Outra característica relevante é a **reusabilidade**! Você sabe o que isso significa? De forma simplificada significa reutilizar, ou seja, você desenvolve um programa com um objetivo e este pode ser reutilizado dentro de outro programa/sistema. Esta é a ideia básica das bibliotecas de funções.
- Nós temos diversos tipos de computadores e diferentes sistemas operacionais, assim é desejável que os sistemas (programas) possam ser executados em diferentes plataformas. Essa característica de um programa é denominada **portabilidade de software** e é muito relevante para avaliar um bom programa.

Softwares que possuam portabilidade e reusabilidade tem custo mais baixo de desenvolvimento e manutenção.

A utilização de Tipos Abstratos de Dados (TAD) melhora a portabilidade e a reutilização do software, reduzindo os custos de desenvolvimento e manutenção.