UNIVERSIDADE PRESBITERIANA MACKENZIE

FACULDADE DE COMPUTAÇÃO E INFORMÁTICA

ESTRUTURA DE DADOS II – Prof. Jean Marcos Laine

Alan Meniuk Gleizer – RA

Caio Vinicius Corsini Fillho – RA

Gilberto De Melo Júnior – RA 10419275

TÍTULO A DECIDIR

São Paulo, SP

20 de novembro de 2024

Sumário

Introdução

5. Modelagem dos Dados em Árvores

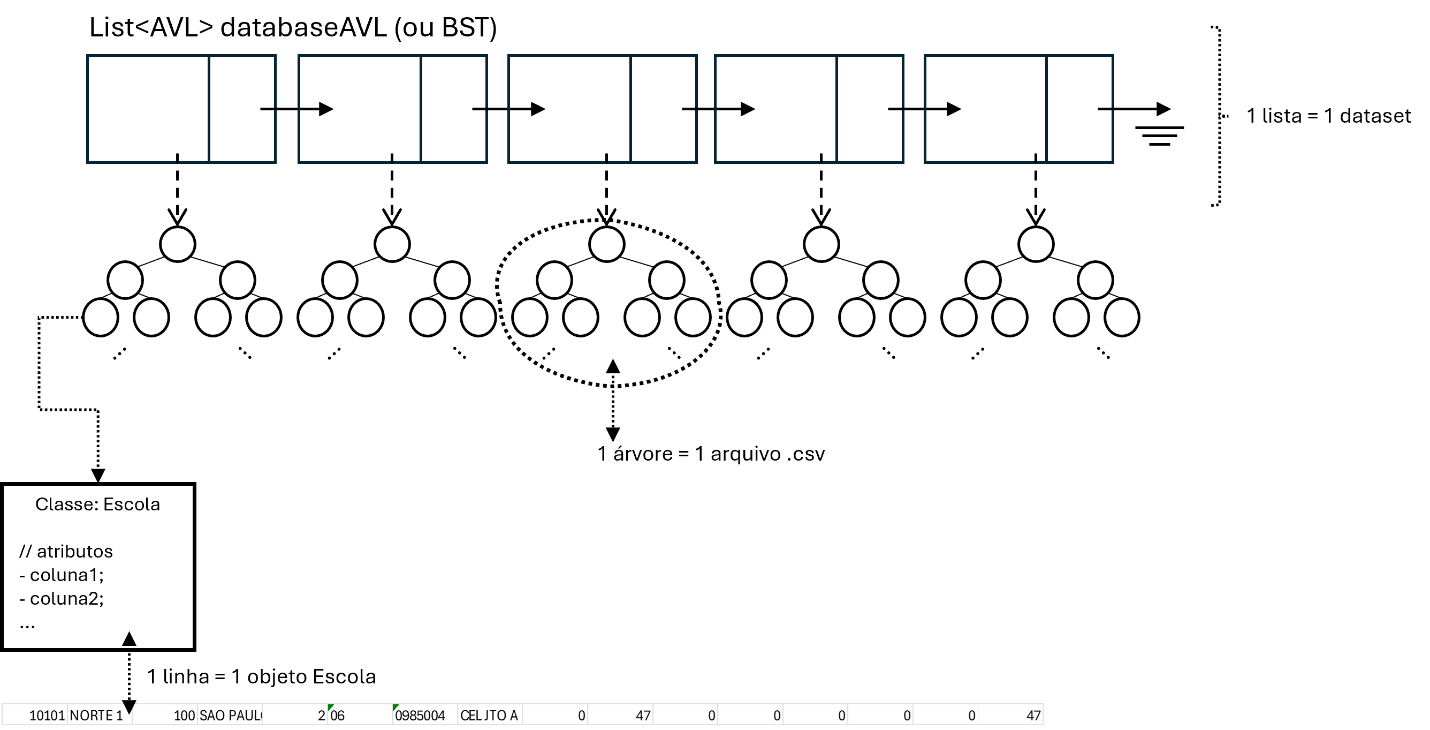
Neste projeto, o objetivo principal foi estruturar e organizar dados de instituições de ensino a partir de um conjunto de registros de escolas, de maneira que permitisse a busca e inserção eficientes. Para isso, utilizamos duas estruturas de dados baseadas em árvores: a árvore binária de busca (BST) e a árvore AVL, uma variação da BST que realiza auto-balanceamento. Na medida que o dataset selecionado armazena as informações segundo a lógica de uma escola por linha, e um arquivo .cvs por semestre, a organização dos dados nas árvores foi realizada de forma a espelhar a estrutura do dataset. Partindo de uma visão micro para macro, a organização do dataset se deu da seguinte forma:

A classe Escola é responsável por encapsular cada linha de dados, representando assim uma escola específica em um semestre específico. Cada coluna de dados de uma linha é representada por um atributo da Escola. Como o dataset possui um número grande de colunas, foi necessária uma pesquisa sobre a forma mais eficiente de lidar com todos esses atributos na codificação do projeto. Em alguns momentos, com na impressão dos atributos, a biblioteca de reflexão do Java foi usada para iterar sobre os atributos da classe Escola em tempo de execução, sem a necessidade de listá-los manualmente no código.

Cada instância da classe Escola é armazenada em um nó (classe Node) de uma árvore BST e outra árvore AVL, e serve como dado central de cada nó. O código da escola (int codEsc) foi escolhido como chave primária para ordenação, busca e balanceamento das estruturas. A escolha pelo codEsc foi bastante intuitiva, na medida que se trata do único código garantidamente único entre todas as linhas de cada .csv, e as árvores trabalhadas não permitem/preveem uso de chaves duplicadas. Também foi decidido armazenar cada codEsc como um inteiro, ao invés de string, dado que operações de comparação entre inteiros são mais rápidas que entre objetos complexos.

Dessa forma, cada .csv, e consequentemente cada semestre, é representado por uma coleção de Nodes em uma árvore AVL e BST. Assim, foi necessário escolher ainda mais uma estrutura de dado para representar os diversos semestres que compõe o dataset como um todo. Como o dataset é limitado a 10 arquivos CSV, optamos pelo uso de uma simples lista encadeada, na qual cada elemento corresponde a uma árvore.

A estrutura final do dataset dentro do programa pode ser bem representada pelo esquema abaixo:

Figura 1: representação da estrutura de organização do dataset

Vale ressaltar que a inserção dos dados nos objetos Escola, e a subsequente construção das árvores e listas de árvores foi realizada manualmente, sem uso de bibliotecas dedicadas à leitura de .csv. Para isso, e com o objetivo de respeitar os princípios da programação orientada a objetos, foram utilizadas duas classes distintas:

A classe CSVreader é responsável por ler e interpretar o conteúdo de cada arquivo CSV. Esse processo é dividido em várias etapas, das quais é interessante destacar a abertura de cada arquivo e criação de uma lista de strings, e a conversão da lista de strings em uma lista de objetos Escola, por meio da tokenização com base no “;”.

Cada .csv requer a criação de um objeto CSVreader, de forma que uma segunda classe DatabaseManager é responsável pelo gerenciamento dos vários CSVreader e criação das árvores e listas de árvores. Neste processo, cabe destacar abertura do diretório que contém os arquivos .csv e criação de uma lista de strings com os nomes dos arquivos, a criação de um CSVreader para cada .csv, e a subsequente criação de árvores AVL e BST a partir das listas de Escola criadas nos CSVreaders. A classe DatabaseManager possui, dessa forma, dois atributos distintos, uma lista de AVLs e uma lista de BSTs que são passados como cópia para chamadas de fora da classe, afim de proteger a integridade dos dados originais.

A estrutura final dos dados na classe DatabaseManager pode ser representada conforme o esquema abaixo:

DatabaseManager

└── AVLdatabase: List<AVL>

├── AVL Semestre 1

│ ├── Node (Escola 1)

│ │ ├── Escola: { codEsc: 1001, nome: "Escola A", ... }

│ ├── Node (Escola 2)

│ │ ├── Escola: { codEsc: 1010, nome: "Escola B", ... }

│ └── ...

├── AVL Semestre 2

│ ├── Node (Escola 3)

│ │ ├── Escola: { codEsc: 1023, nome: "Escola C", ... }

│ ├── Node (Escola 4)

│ │ ├── Escola: { codEsc: 1035, nome: "Escola D", ... }

│ └── ...

└── AVL Semestre N

├── Node (Escola N1)

│ ├── Escola: { codEsc: 2001, nome: "Escola X", ... }

├── Node (Escola N2)

│ ├── Escola: { codEsc: 2010, nome: "Escola Y", ... }

└── ...

└── BSTdatabase: List<BST>

Figura 2: Esquema dos principais atributos de DatabaseManager

11. Reflexão Final

11.1. Alan Meniuk Gleizer

Este projeto apresentou uma oportunidade interessante de lidar com bases de dados reais e bastante relevantes em um contexto no qual foi possível e importante focar também na implementação de estruturas de dados cujos detalhes costumamos abstrair. Desde o princípio, ficou claro que as árvores com as quais trabalhamos, ainda que eficientes para buscas e ordenações, não eram necessariamente as estruturas mais adequadas para uma análise do conteúdo. Ainda assim, foi desafiador adequar essas estruturas ao nosso uso e pensar na melhor maneira de representar os vários .csv como um único banco de dados. A solução final de listas de árvores envolveu bastante discussão no grupo e diversas modelagens da classe Escola e seus atributos.

Também gostei muito de tentar aplicar os princípios de programação orientada a objetos e de modelagem de software, que estamos estudando na disciplina de Projeto de Software, ao desenvolvimento do projeto. Mesmo se tratando de um programa bastante pequeno e simples, conceitos como encapsulamento, modularidade e responsabilidade única realmente ajudaram a estruturar um sistema coeso e bastante fácil de alterar conforme novas necessidades do grupo surgiram. Especificamente, classes, como Escola, CSVreader e DatabaseManager, têm responsabilidades bem definidas: Escola armazena os dados de cada escola/linha, CSVreader cuida da leitura e processamento de um arquivo CSV (por instância), e DatabaseManager coordena o armazenamento em árvores e a organização geral dos dados. Os métodos das funções também foram modularizados de forma que, antes mesmo da seleção do dataset, seria possível adequá-las para a leitura de outros tipos e outras quantidades de arquivos.