	Algoritmos e Estruturas de Dados I	*5954009*
	Departamento de Computação e Matemática	1º Semestre/2024
	Prof. Dr. José Augusto Baranauskas Monitor: Joao Victor Lopes Pereira dos Santos	Trabalho 2 - *Trees*

A partir das diferentes implementações de árvores, como Árvores Binárias e Árvores Balanceadas, podemos utilizá-las em diferentes aplicações na área da computação. Uma das áreas mais comuns é o armazenamento e a indexação de itens em árvores, para posteriores consultas, inserções e remoções.

O objetivo deste trabalho é, a partir de uma tabela de valores separados por vírgula (*Comma-Separated Values* ou CSV), extrair informações de pacientes e inseri-las em diferentes implementações de árvores ABB e AVL, extraindo métricas de cada uma delas.

O arquivo "users.csv" mostra a relação de usuários em uma rede social, onde cada usuário é representado por uma tupla (UserID, Name, Birthday), onde "*UserID*" é um inteiro, "*Name*" é uma string e "*Birthday*" é uma data. Segue abaixo alguns exemplos da tabela:

```
ID,Name,Birthday
1,Richard Jones,1967-02-24
2,Barbara Thomas,1990-11-03
3,Robert Gonzalez,1966-04-12
4,Susan Martinez,1987-09-12
5,Elizabeth Thomas,1967-01-15
6,Patricia Jones,1987-02-22
7,Joseph Rodriguez,1991-12-14
8,Karen Williams,1986-04-26
9,Jessica Jackson,1989-08-21
```

Será fornecido um programa básico em C++ (FileReader.cpp) cuja funcionalidade é ler o arquivo e armazenar os usuários num vetor da *struct* apropriada. **Os grupos terão duas opções de entrega no trabalho**:

Opção 1:

1. Implementar as classes **ABB** e **AVL**, de forma *genérica,* de forma que possam indexar qualquer tipo de classe e por qualquer método de comparação. Por exemplo, uma ABB genérica pode indexar tanto a classe Fish = (FishName, FishAge, FishType) e por qualquer um dos 3 atributos de Fish, quanto a classe User, da mesma forma que em c++ você consegue declarar um vector<int> ou um vector<String>.

2. Implementar uma função main que, lendo as entradas do código, utilize as duas classes implementadas para instanciar 6 árvores diferentes, cada uma guardando individualmente todas as tuplas da tabela. Então serão uma ABB e AVL que guardam users por nome, uma ABB e AVL que guardam users por ID, e uma ABB e uma AVL que guardam users por birthday.

3. A partir dessas inserções, extrair as métricas descritas na tabela abaixo **para cada um dos tipos de indexação**, ou seja, para cada um dos 3 atributos.

4 - O programa main deve, além de encher as 6 árvores com todas as tuplas do arquivo, fornecer uma interface que permite ao usuário:

1 - Buscar uma tupla por qualquer um dos 3 índices.
2 - Adicionar uma nova tupla (deve ser adicionada em cada uma das 6 árvores, com as medidas de Nro de comparações da inserção sendo exibidas).
3 - Remover uma tupla (deve ser removida de cada uma das 6 árvores).

Para essa opção, os alunos podem consultar o arquivo "*Adendo"*, com explicações sobre como viabilizar a implementação.

Opção 2:

1. Implementar as classes **AVLID, BSTID, AVLName, BSTName, AVLBirthday, BSTBirthday**. Cada classe representa uma árvore (ABB (BST) ou AVL) que indexa os usuários por ID, Nome ou Entry, respectivamente.*****

Para **AVLBirthday** e **ABBBirthday**, será obrigatório o uso da biblioteca <ctime> para guardar e indexar as datas. Será fornecido um documento com uma explicação básica da biblioteca e fontes adicionais.

2. Implementar uma função main que, lendo as entradas do código, utilize as funções de insert implementadas para criar 6 árvores diferentes, cada uma guardando individualmente todas as tuplas da tabela.

3. A partir dessas inserções, extrair as métricas descritas na tabela abaixo **para cada um dos tipos de indexação**.

Tabela de métricas:

A	Número de Usuários (*Igual para todos*)
B	Tamanho da Árvore Mínima
C	Tamanho da AVL
D	Nro de Folhas da AVL
E	Tamanho da ABB
F	Nro de Folhas da ABB
G	Rotações na inserção da AVL
H	Nro de Comparações na Inserção na ABB
I	Nro de Comparações na Inserção na AVL

Média de Comparações por Usuário na ABB (H / A)
Média de Comparações por Usuário na AVL (I / A)
Média de Rotações por Usuário na AVL (G / A)

4 - O programa main deve, além de encher as 6 árvores com todas as tuplas do arquivo, fornecer uma interface que permite ao usuário:

1 - Buscar uma tupla por qualquer um dos 3 índices.
2 - Adicionar uma nova tupla (deve ser adicionada em cada uma das 6 árvores, com as medidas de Nro de comparações da inserção sendo exibidas).
3 - Remover uma tupla (deve ser removida de cada uma das 6 árvores).

**: Ler o adendo para mais informações sobre esse tipo de implementação em casos reais.*

Submissão

O trabalho será realizado em duplas ou trios. **Não serão aceitos trabalhos individuais.** Submeta a implementação da dupla ou trio (apenas um membro precisa submeter) na plataforma TIDIA-AE na Atividade "Trabalho 1 - Trees" até a data limite lá especificada. Coloque seus nomes completos em todos os arquivos sendo submetidos, na forma de comentário no início de cada arquivo (.h ou .cpp). Compacte os arquivos em um único arquivo .zip (não utilize espaços no nome do arquivo compactado, nem adicione pastas/diretórios no arquivo compactado).

Avaliação

Na nota do trabalho também serão considerados os seguintes critérios (além dos já mencionados nas Disposições Gerais entregues no início do semestre):

Correção: O programa faz o que foi solicitado? Faz tudo o que foi solicitado? Utiliza encapsulamento de informação? (i.e., acessa adequadamente os ADTs definidos?) Não há vazamento de memória?

Eficiência: As operações são executadas da maneira mais eficiente para cada estrutura de dados? Evita código duplicado/redundante/não atingível?

Interface: É simples de usar, genérico, prático, tolera os erros mais óbvios? O trabalho foi entregue dentro das especificações, implementando um ADT? Os arquivos estão em formato ZIP, com os nomes de arquivos solicitados?

- interface do programa;
- implementação dos ADTs e métodos;

Código fonte: é claro (*layout*, espaçamento, organização em geral), nomes de variáveis são sugestivos, e há documentação/comentários apropriados no código? Faz uso de pré- e pós-condições? Quando aplicável, faz uso de sublagoritmos (funções, procedimentos ou métodos) adicionais que melhoram a legibilidade do(s) método(s) solicitado(s) sem comprometer sua eficiência (por exemplo, na notação assintótica **O(*n*),** onde ***n*** representa o tamanho da entrada?)
