TRABALHO FINAL
Medição de Tempo em Jogos Digitais

1. Objetivo

O objetivo do trabalho é comparar o desempenho de diferentes estruturas de dados vistas na disciplina em uma aplicação de estimativa de tempo de jogo com base em dados da plataforma Steam.

2. Especificação da Aplicação

Um jogador está montando uma lista de jogos que pretende jogar nas próximas semanas. Ele gostaria de saber quanto tempo, em média, precisará investir para jogar todos eles. Para isso, ele consulta um arquivo com estatísticas públicas de jogos da Steam, contendo o nome do jogo e o tempo médio de jogo efetivamente "jogado" pelos usuários, medido em horas (número float).

Esse processo de estimar o tempo total é demorado e sujeito a erros, especialmente quando há centenas de jogos disponíveis. Por isso, o jogador decidiu automatizar a tarefa.

★ Sua tarefa é projetar uma aplicação que recebe dois arquivos de entrada:

1. Um arquivo CSV contendo o nome dos jogos e o número médio de horas jogadas. Para teste, use o dataset disponível no Moodle que foi gerado a partir do dataset "steam video games" [1].
2. Um arquivo texto contendo uma lista de jogos que o jogador deseja jogar (um jogo por linha).

A tabela de jogos e suas horas médias devem ser armazenadas em duas estruturas de dados. Você deve criar diferentes versões da aplicação utilizando pelo menos duas das quatro árvores vistas em aula (ABP, AVL, Rubro-Negra ou Splay).

★ A tabela de jogos (arquivo CSV) deve ser carregada na árvore **na ordem** em que os **jogos aparecem** no arquivo. As **chaves** devem ser os **nomes** dos jogos**.** Assuma que não haverá repetição de nomes.

★ A aplicação não é *case sensitive*: letras maiúsculas e minúsculas devem ser tratadas como iguais.

★ A lista de jogos escolhidos pelo jogador **não deve ser armazenada em nenhuma estrutura**. Cada jogo deve ser usado apenas para consultar a árvore e encontrar seu respectivo tempo médio. Se um jogo da lista não se encontra na tabela de jogos, utilize zero como tempo médio e compute seu tempo.

★ Seu programa deverá ser chamado a partir da linha de comando (passando parâmetros para o `main`).

Exemplo de chamada: `C:\estimador_horas` **`steam_games.csv lista_jogos.txt saida_lista.txt`**

As entradas e saídas da sua aplicação são:
Entradas:
(i) nome do arquivo com a tabela de jogos e suas horas médias;
(ii) nome do arquivo com os jogos escolhidos pelo jogador.

[1] https://www.kaggle.com/datasets/tamber/steam-video-games

Saídas:
(i) arquivo de saída com a estimativa total de horas;
(ii) estatísticas sobre o processo de geração da árvore e das consultas (número de nós, altura da árvore, número de rotações e número de comparações realizadas durante as consultas). Considere rotações duplas de árvores como apenas uma rotação.

★ Os arquivos de entrada são arquivos de texto.
O arquivo com os dados da Steam segue o formato `<nome do jogo> , <horas medias>`:

```
Counter-Strike,1743
Dota 2,1150
Terraria,295
Left 4 Dead 2,230
Garry’s Mod,420
Stardew Valley,85
Portal 2,65
The Witcher 3,110
Cyberpunk 2077,70
Baldur’s Gate 3,120
...
```

O arquivo com a lista de jogos contém apenas os nomes desejados:

```
Stardew Valley
The Witcher 3
Cyberpunk 2077
Baldur’s Gate 3
Portal 2
```

Exemplo de saída:

Arquivo de saída: `saida_lista.txt`
Estimativa de horas totais para jogar os títulos da lista usando a tabela steam_games.csv.

```
Tempo total estimado: 450 horas

======== ESTATÍSTICAS ABP ============
Número de Nodos: 1000
Altura: 23
Rotações: 0
Comparações: 132

======== ESTATÍSTICAS AVL ============
Número de Nodos: 1000
Altura: 12
Rotações: 315
Comparações: 10
```

★ O número de comparações realizadas com os elementos da árvore deve ser calculado pela função a seguir (onde `comp` é uma variável global que acumula o número de comparações):

```
pNodoA* consulta(pNodoA *a, char *chave){
    while (a != NULL){
        comp++;
        if (!strcmp(a->jogo, chave)){
            return a;
        } else {
            if (strcmp(a->jogo, chave) > 0)
                a = a->esq;
            else
                a = a->dir;
        }
    }
    return NULL;
}
```

★ Arquivos de teste (entradas e saídas correspondentes) estarão disponíveis no Moodle. No dia da apresentação, novos conjuntos de arquivos serão fornecidos.

3. Requisitos

- É necessário elaborar um relatório detalhado com a análise comparativa do desempenho das duas árvores. Utilize recursos como tabelas e gráficos para dar suporte às suas conclusões. É importante **testar com diferentes tamanhos de arquivos e ordens de inserção** para observar o impacto no desempenho das árvores.
- O trabalho deve ser feito, preferencialmente, em duplas. Também serão aceitos trabalhos individuais e de duplas com integrantes de turmas diferentes.
- A linguagem de programação aceita é C (não é C++ nem C#).

4. Entrega e Apresentação

- **08 de dezembro de 2025** — apresentação em aula presencial e entrega pelo Moodle.
- Ambos os integrantes da dupla precisam estar presentes na apresentação.
- Somente os trabalhos apresentados serão corrigidos.
- Deve ser entregue via Moodle (após a apresentação) um arquivo .zip contendo: (i) todos os arquivos-fonte e executáveis necessários para execução do trabalho e (ii) o relatório em PDF.

5. Critérios de Avaliação

Para a avaliação serão adotados os seguintes critérios:
- Funcionamento (Peso: 30%);
- Organização e documentação do código (Peso: 30%);
- Relatório (Peso: 40%).

6. Dicas

- Há um exemplo de código no Moodle (com vídeo explicativo) mostrando como fazer a passagem de parâmetros para a função `main`.
- Para ler cada linha dos arquivos de entrada, utilize a função `fgets`.
- Para separar o nome do número de horas médias, utilize a função `strtok` (separador `,`).
- A ordem lexicográfica dos nomes dos jogos determinará a organização da árvore (utilize `strcmp` para comparar as strings).

Importante:

Este trabalho deverá representar a solução da dupla para o problema proposto. O plágio é terminantemente proibido e a sua detecção implicará nota zero. Além disso, é expressamente proibido o uso de ferramentas de inteligência artificial generativa, incluindo modelos de linguagem (LLMs) como ChatGPT, Copilot, Gemini ou similares, sob pena de anulação do trabalho. É permitido reusar código disponibilizado em aula ou de fontes públicas, desde que todas as referências sejam devidamente citadas no código e no relatório.