UNIVERSIDADE FEDERAL DO PIAUÍ
CAMPUS SENADOR HELVÍDIO NUNES DE BARROS - CSHNB

Curso: Sistemas de Informação
Disciplina: Estruturas de dados 2
Docente: Juliana Oliveira de Carvalho
Discente: Marcos André Leal Silva

UNIVERSIDADE FEDERAL DO PIAUÍ - UFPI

Relatório técnico

Picos-PI
08 de novembro de 2023

Resumo: Este projeto, tem como objetivo demonstrar a prática e implementação de duas Estruturas de Dados: Árvores binárias de busca e árvores binárias AVL. Essas estruturas foram utilizadas para a resolução de alguns problemas práticos proposto no projeto através da Linguagem de programação C. O mesmo abordará as etapas de cada de cada questão e apresentará suas respectivas implementações, oferecendo uma visão abrangente das aplicações dessas estruturas de dados. Além do mais, serão feitos experimentos para verificar os tempos de execução tanto da árvore AVL quanto da árvore binária de busca.

Introdução: As estruturas de dados são um conceito muito importante para a programação e ciência de dados. Tratam-se de maneiras de agregar e organizar dados na memória de um computador ou dispositivo, de forma que façam sentido e proporcionem um bom desempenho ao serem processados. Nesse sentido, estamos considerando dados como sendo blocos de programação que representam algo e têm a função de resolver problemas computacionais. Para isso, eles devem ter a possibilidade de ser simbolizados, armazenados e manipulados.

Uma Árvore Binária de busca é uma estrutura de dados de árvore binária baseada em nós, onde a subárvore à esquerda de cada nó possui valores numéricos inferiores ao nó e a subárvore à direita de cada nó possui valores numéricos superiores ao nó.

Árvore AVL é uma árvore binária de busca balanceada, ou seja, uma árvore balanceada (árvore completa) são as árvores que minimizam o número de comparações efetuadas no pior caso para uma busca com chaves de probabilidades de ocorrências idênticas. Contudo, para garantir essa propriedade em aplicações dinâmicas, é preciso reconstruir a árvore para seu estado ideal a cada operação sobre seus nós (inclusão ou exclusão), para ser alcançado um custo de algoritmo com o tempo de pesquisa tendendo a O(\log n)}.

O objetivo desse projeto é através da estrutura de dados “Árvore binária de busca” e “Árvore binária AVL”, criar aplicações capazes de armazenar informações sobre diferentes tipos de series, suas temporadas e personagens, utilizando alguns parâmetros pré-determinados pelo enunciado. A partir dessa inserção de informações nas árvores binárias, o projeto tem que ser capaz de realizar buscar e apresentar as informações gravadas nessas estruturas. O usuário poderá, a partir dos conhecimentos de alguns dados cadastrados, realizar a busca dessas informações que serão apresentadas para o mesmo. Para a implementação do projeto, foi utilizado a linguagem de programação C, com seu paradigma de programação imperativo para a criação do código fonte. Através da ferramenta “Visual Studio Code (VS Code)”, esse programa foi criado, testado e executado. O programa apresenta duas estruturas de dados em formato de árvores binárias chamadas de “arvserie” e “arvtemporada”. A “arvserie” armazena uma variável do tipo Serie com algumas informações (id, número de temporadas e nome), um endereço para uma árvore binária de busca contendo informações de cada temporada e duas estruturas para direcionar o lado de cada nó da árvore (Dir e Esq). Já “arvtemporada” armazena uma variável do tipo Temporada com algumas informações (número da temporada, ano de lançamento, quantidade de episódios e título da temporada), um endereço para uma lista simples de participantes (nome do artista, nome do personagem, descrição do personagem e uma estrutura “prox” para apontar o próximo elemento da lista) e duas estruturas para direcionar o lado de cada nó da árvore (Dir e Esq).

A principal diferença entre árvores binárias de busca e árvores AVL está no balanceamento. As árvores binárias de busca tradicionais não garantem o equilíbrio, enquanto as árvores AVL são

projetadas para manter o equilíbrio automaticamente, resultando em um melhor desempenho previsível, especialmente em cenários de operações frequentes de inserção e exclusão. No entanto, o uso de memória adicional e a complexidade de implementação são algumas das considerações ao escolher entre essas estruturas de dados.

Essas árvores serão testadas através de um experimento que busca verificar e avaliar o tempo de inserção e de busca e cada árvore. O experimento é repetido 30 vezes para coletar dados em várias iterações e fornecer uma ideia de como o desempenho da árvore binária de busca e a árvore binária AVL varia ao longo das iterações. É uma maneira de avaliar a eficiência de operações de inserção e busca em árvores binárias, bem como entender como o tempo de execução pode variar em diferentes situações.

Seções específicas:

Informações técnicas: Para o desenvolvimento e testes deste projeto foi utilizado um notebook com um processador 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz, doze gigas de memória RAM ddr4, sistema operacional com a arquitetura de 64 bits, Windows 11 Home.

Os códigos foram feitos na IDE e editor de texto Visual Studio Code e compilados pelo compilador gcc (MinGW.org GCC Build-2) 9.2.0.

Questão 1:

O programa apresenta a implementação de árvore binárias de busca para gerenciar informações sobre séries, suas temporadas e participantes, permitindo que o usuário insira, visualize e pesquise informações relacionadas a esses elementos.

1. Estruturas de dados:
   - O programa utiliza algumas estruturas de dados no formato de árvore, incluindo “Serie”, “Temporada”, “Participantes”, “ArvSerie”, e “ArvTemporada” para representar séries, temporadas, participantes e árvores de séries e temporadas.

2. Operações de Leitura:
   - O programa oferece funções “ler_serie” e “ler_temporada” para ler informações de séries e temporadas a partir da entrada padrão (teclado).

3. Inserção de Séries e Temporadas:
   - A função “inserirArvSerie” permite inserir séries na árvore de séries. A árvore de séries é organizada com base no código da série.
   - A função “inserirArvTemp” permite inserir temporadas na árvore de temporadas dentro de uma série. A árvore de temporadas é organizada com base no número da temporada.

4. Impressões:
   - Existem funções para imprimir informações sobre séries, temporadas e participantes: “imprimirSerie”, “imprimirTemp”, “imprimirParticipantes”.

5. Inserção de Participantes:
    - A função “inserirLista” permite adicionar participantes a uma temporada específica. Os participantes são mantidos em uma lista dentro de uma temporada. A inserção é feita em ordem alfabética com base no nome do artista.

6. Buscas:
    - As funções “buscaArvSerie” e “buscaArvTemp” permitem procurar séries e temporadas com base em seus códigos e números, respectivamente.

7. Impressão de Séries e Temporadas:
    - As funções “imprimirArvSerie” e “imprimirArvTemp” permitem imprimir todas as séries e temporadas em ordem, percorrendo as árvores em ordem.

8. Menu Interativo:
    - O programa oferece um menu interativo que permite ao usuário realizar várias operações, incluindo cadastro de séries, temporadas, participantes e impressão de informações.

9. Chamada de Funções com Argumentos:
    - O programa chama várias funções com argumentos para executar operações específicas, como inserir séries, temporadas, participantes e realizar buscas.

10. Manipulação de Listas:
    - A manipulação de participantes é realizada por meio de uma lista encadeada dentro de cada temporada.

11. Controle de Fluxo:
    - O programa utiliza estruturas de controle, como loops “while” e “switch”, para controlar o fluxo e a interação com o usuário.

12. Entrada e Saída:
    - O programa lida com entrada e saída de dados, permitindo ao usuário inserir informações e visualizar resultados na tela.

13. Tratamento de Erros:
    - O programa inclui tratamento de erros simples, como mensagens de aviso quando uma série ou temporada não é encontrada.

14. Liberar Memória:
    - Para evitar vazamento de memória, o programa oferece funções liberarArvTemp() e liberarArvSerie() para liberar a memória alocada para as árvores de temporadas e séries, respectivamente.

    - Essas funções são chamadas no final do programa em liberarTodasSeries() para liberar a memória de todas as séries cadastradas.

Questão 2:

O programa foi criado com o intuito de servir como uma ferramenta de teste que demonstra a criação de árvores binárias de busca, inserção de dados nesses nós e medição do tempo de busca. Seus resultados podem ser fundamentais para entender o tempo de execução da estrutura.

1. Estruturas de Dados:
    - O programa define as estruturas de dados Serie, Temporada, ArvSerie, e ArvTemporada para representar informações relacionadas a séries de televisão e as árvores binárias que as organizam.

2. Geração Aleatória de Dados:
    - O programa utiliza a função rand() da biblioteca stdlib.h para gerar dados aleatórios para as séries. Ele gera códigos de séries, números de temporadas e títulos aleatórios.

3. Criação de Árvores Vazias:
    - As funções criaArvSerie e criaArvTemp são responsáveis por criar árvores de séries e temporadas vazias, respectivamente. Essas funções inicializam as raízes como nulas.

4. Inserção em uma ABB:
    - A função inserirArvSerie insere uma nova série na árvore de séries (ArvSerie). A inserção segue as regras de uma árvore binária de busca, onde séries com códigos menores são inseridas à esquerda e séries com códigos maiores à direita.

5. Busca em uma ABB:
    - A função buscaArvTemp permite procurar por uma temporada específica em uma árvore de temporadas (ArvTemporada). Ela segue as regras da árvore binária de busca para encontrar a temporada desejada.

6. Medição de Tempo:
    - O programa mede o tempo gasto em operações de inserção e busca usando a função clock() da biblioteca time.h. O tempo é medido em nanossegundos e é calculado em relação à constante CLOCKS_PER_SEC, que representa o número de clocks por segundo.

7. Laço de Repetição:
    - O programa executa uma simulação de 30 iterações. Cada iteração envolve a criação de uma nova árvore de séries, seguida por 30.000 operações de inserção de séries aleatórias e 30.000 operações de busca por temporadas aleatórias.

8. Cálculo da Média de Tempo:
    - O tempo gasto em cada operação é acumulado ao longo das 30 iterações, e a média é calculada para a inserção e busca separadamente.

9. Apresentação de Resultados:
    - O programa imprime a média do tempo gasto nas operações de inserção e busca em nanossegundos após as 30 iterações. Os resultados são exibidos de forma clara e organizada

Questão 3.1:

O programa apresenta a implementação de árvore binárias AVL para gerenciar informações sobre séries, suas temporadas e participantes, permitindo que o usuário insira, visualize e pesquise informações relacionadas a esses elementos.

1. Definição de Estruturas de Dados:
    - Foram definidas várias estruturas de dados para representar séries, temporadas e participantes, incluindo Serie, Temporada, Participantes, ArvSerie e ArvTemporada.

2. Inserção de Séries:
    - A função inserirArvSerie é responsável por inserir uma série na árvore AVL de séries. O código da série é usado para determinar a posição correta na árvore.

3. Inserção de Temporadas:
    - A função inserirArvTemp é responsável por inserir uma temporada em uma árvore AVL de temporadas associada a uma série específica. O número da temporada é usado para determinar a posição correta na árvore.

4. Inserção de Participantes em Temporadas:
    - A função inserirLista permite a inserção de participantes em uma temporada específica. Ela verifica se a temporada existe antes de inserir os participantes na lista.

5. Cálculo da Altura da Árvore:
    - A função alturaNOArvSerie() calcula a altura de um nó da árvore. A altura é usada para determinar o fator de balanceamento da árvore.

6. Fator de Balanceamento (FB):
    - A função fbArvSerie() calcula o fator de balanceamento de um nó, que é a diferença entre a altura da subárvore esquerda e da subárvore direita.

7. Busca por Série e Temporada:
    - As funções buscaArvSerie e buscaArvTemp permitem a busca de séries e temporadas específicas na árvore AVL, usando seus códigos ou números de temporada como critérios de busca.

8. Impressão de Séries e Temporadas:
    - As funções imprimirArvSerie e imprimirArvTemp percorrem a árvore AVL de séries e temporadas e imprimem os dados das séries e temporadas.

9. Impressão de Participantes em Temporadas:
    - A função imprimirTodosParticipantes permite imprimir todos os participantes de uma temporada específica.

10. Impressão de Artistas de um Personagem:
    - A função imprimirTodosArtistasPersonagem permite imprimir todos os artistas associados a um personagem específico em uma série específica.

11. Rotações AVL:
    - Foram implementadas várias funções de rotação (simples e duplas) para manter a árvore balanceada. Elas são usadas nas funções de inserção para garantir que a árvore permaneça balanceada.

12. Menu Interativo:
    - O programa oferece um menu interativo para o usuário interagir com as diferentes funcionalidades, como cadastro de séries, temporadas e participantes, bem como impressão de informações.

13. Liberar Memória:
    - O programa inclui funções liberarArvSerie e liberarArvTemp para liberar a memória alocada dinamicamente após a execução.

14. Manipulação de Entradas do Usuário:
    - O programa lê e manipula entradas do usuário, como códigos de série, temporadas, nomes de artistas, nomes de personagens, descrições, etc.

15. Tratamento de Casos Especiais:
    - O programa lida com casos especiais, como quando uma temporada não é encontrada ou quando uma série não existe.

16. Validação de Dados:
    - O programa inclui verificações para garantir que as inserções e buscas são feitas corretamente.

Questão 3.2:

O programa foi criado com o intuito de servir como uma ferramenta de teste que demonstra a criação de árvores binárias AVL, inserção de dados nesses nós e medição do tempo de busca. Seus resultados podem ser fundamentais para entender o tempo de execução da estrutura.

1. Estruturas de Dados:
    - O programa define as estruturas de dados Serie, Temporada, ArvSerie e ArvTemporada para representar informações sobre séries e suas temporadas, bem como as árvores AVL correspondentes.

2. Geração de Séries Aleatórias:

- A função ler_serie() é usada para gerar informações aleatórias sobre séries de TV, como um código de série, o número de temporadas e um título.

3. Criação da Árvore:
    - A função criaArvSerie() inicializa uma árvore vazia e a função criaArvTemp() inicializa uma árvore de temporadas vazia.

4. Inserção na Árvore:
    - A função inserirArvSerie() é responsável pela inserção de séries na árvore AVL de séries. Ela garante que a árvore permaneça balanceada por meio de rotações quando necessário.

5. Cálculo da Altura da Árvore:
    - A função alturaNOArvSerie() calcula a altura de um nó da árvore. A altura é usada para determinar o fator de balanceamento da árvore.

6. Fator de Balanceamento (FB):
    - A função fbArvSerie() calcula o fator de balanceamento de um nó, que é a diferença entre a altura da subárvore esquerda e da subárvore direita.

7. Rotações:
    - As funções rotacaoSDArvSerie(), rotacaoSEArvSerie(), rotacaoDEDArvSerie() e rotacaoDDEArvSerie() são utilizadas para realizar rotações simples e duplas em torno dos nós da árvore para restaurar o balanceamento da árvore AVL.

8. Busca na Árvore de Temporadas:
    - A função buscaArvTemp() é usada para buscar temporadas em uma árvore de temporadas.

9. Medição de Tempo de Execução:
    - O programa mede o tempo de execução de duas operações: inserção e busca.
    - Para medir o tempo, ele utiliza a função clock() para registrar o tempo de início e fim de cada operação.

10. Ciclo Principal:
    O programa possui um ciclo principal que realiza os seguintes passos:
    - Cria uma árvore AVL de séries vazia.
    - Gera aleatoriamente séries de TV e as insere na árvore, medindo o tempo de inserção.
    - Realiza buscas aleatórias em temporadas nas árvores de temporadas, medindo o tempo de busca.
    - Calcula médias dos tempos de inserção e busca com base em várias execuções.

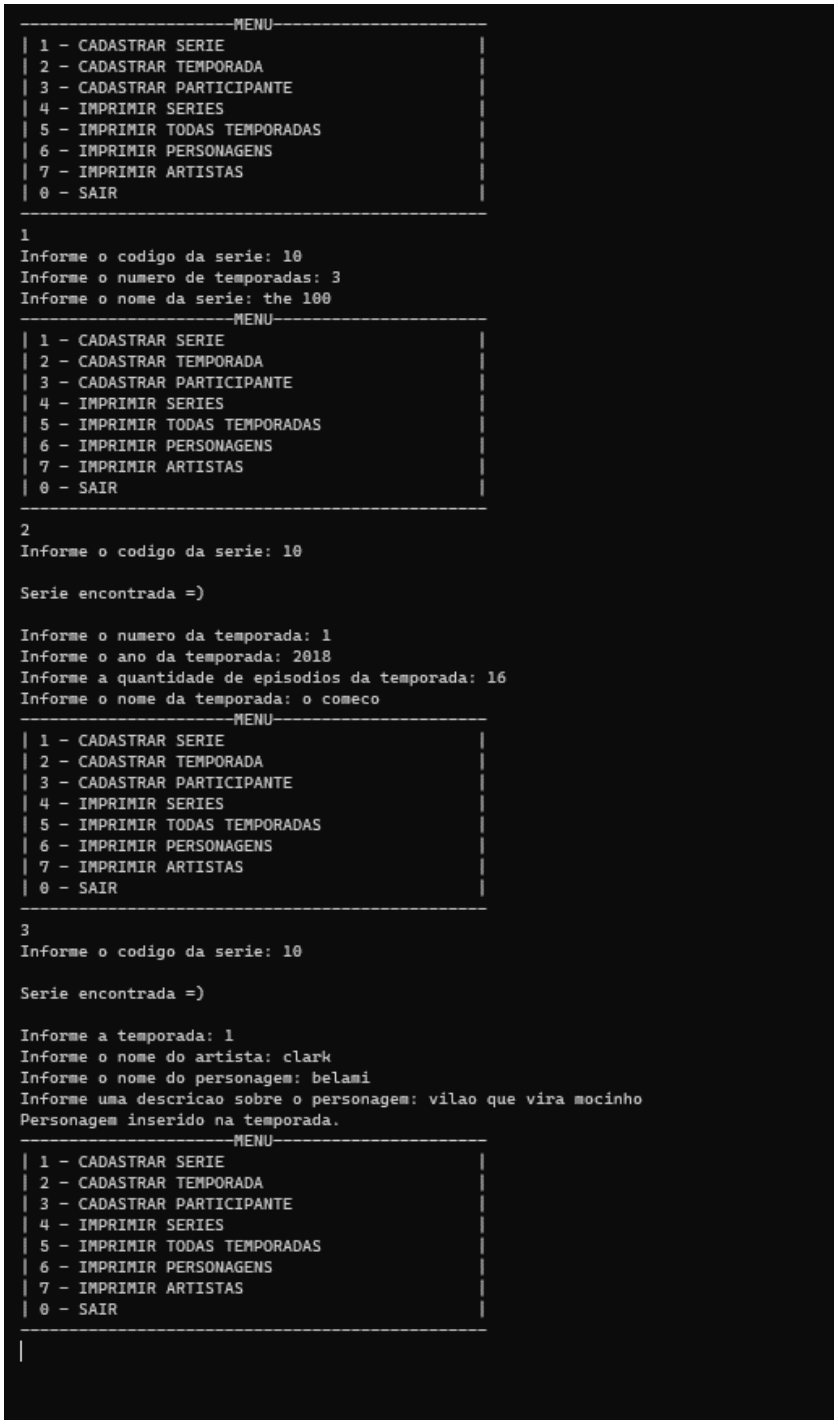
11. Liberação de Memória:
- O programa inclui funções para liberar a memória alocada para as árvores no final da execução.

12. Saída de Dados:

- O programa exibe o tempo médio de inserção e busca em nanossegundos para cada execução e, em seguida, passa para a próxima execução.

Resultados da execução do programa:

Questão 1:



```
----------------------MENU-----------------------
| 1 - CADASTRAR SERIE                           |
| 2 - CADASTRAR TEMPORADA                       |
| 3 - CADASTRAR PARTICIPANTE                    |
| 4 - IMPRIMIR SERIES                           |
| 5 - IMPRIMIR TODAS TEMPORADAS                 |
| 6 - IMPRIMIR PERSONAGENS                      |
| 7 - IMPRIMIR ARTISTAS                         |
| 0 - SAIR                                      |
-------------------------------------------------
4
Codigo da serie: 10
Quantidade de temporadas: 3
Titulo da serie: the 100

----------------------MENU-----------------------
| 1 - CADASTRAR SERIE                           |
| 2 - CADASTRAR TEMPORADA                       |
| 3 - CADASTRAR PARTICIPANTE                    |
| 4 - IMPRIMIR SERIES                           |
| 5 - IMPRIMIR TODAS TEMPORADAS                 |
| 6 - IMPRIMIR PERSONAGENS                      |
| 7 - IMPRIMIR ARTISTAS                         |
| 0 - SAIR                                      |
-------------------------------------------------
5
Informe o codigo da serie: 10
Ano: 2018
Temporada: 1
Quantidade de episodios: 16
Nome da temporada: o comeco

----------------------MENU-----------------------
| 1 - CADASTRAR SERIE                           |
| 2 - CADASTRAR TEMPORADA                       |
| 3 - CADASTRAR PARTICIPANTE                    |
| 4 - IMPRIMIR SERIES                           |
| 5 - IMPRIMIR TODAS TEMPORADAS                 |
| 6 - IMPRIMIR PERSONAGENS                      |
| 7 - IMPRIMIR ARTISTAS                         |
| 0 - SAIR                                      |
-------------------------------------------------
6
Informe o codigo da serie: 10

Serie encontrada =)
Informe a temporada: 1
Nome do artista: clark
Nome do personagem: belami
Descricao do personagem: vilao que vira mocinho

----------------------MENU-----------------------
| 1 - CADASTRAR SERIE                           |
| 2 - CADASTRAR TEMPORADA                       |
| 3 - CADASTRAR PARTICIPANTE                    |
| 4 - IMPRIMIR SERIES                           |
| 5 - IMPRIMIR TODAS TEMPORADAS                 |
| 6 - IMPRIMIR PERSONAGENS                      |
| 7 - IMPRIMIR ARTISTAS                         |
| 0 - SAIR                                      |
-------------------------------------------------
7
Informe o codigo da serie: 10
Informe o nome do personagem: belami

TEMPORADA 1

Nome do artista: clark

----------------------MENU-----------------------
| 1 - CADASTRAR SERIE                           |
| 2 - CADASTRAR TEMPORADA                       |
| 3 - CADASTRAR PARTICIPANTE                    |
| 4 - IMPRIMIR SERIES                           |
| 5 - IMPRIMIR TODAS TEMPORADAS                 |
| 6 - IMPRIMIR PERSONAGENS                      |
| 7 - IMPRIMIR ARTISTAS                         |
| 0 - SAIR                                      |
-------------------------------------------------
```

Questão 2:

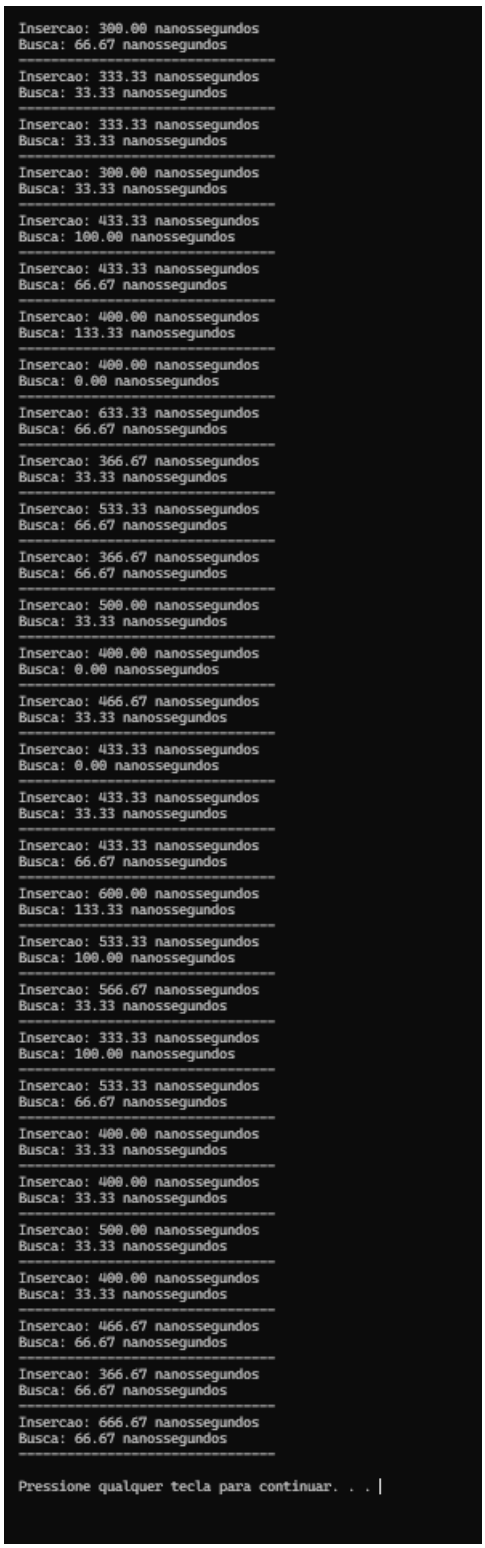

Insercao: 333.33 nanossegundos
Busca: 33.33 nanossegundos
Insercao: 266.67 nanossegundos
Busca: 0.00 nanossegundos
Insercao: 300.00 nanossegundos
Busca: 0.00 nanossegundos
Insercao: 300.00 nanossegundos
Busca: 33.33 nanossegundos
Insercao: 200.00 nanossegundos
Busca: 0.00 nanossegundos
Insercao: 166.67 nanossegundos
Busca: 0.00 nanossegundos
Insercao: 200.00 nanossegundos
Busca: 33.33 nanossegundos
Insercao: 166.67 nanossegundos
Busca: 33.33 nanossegundos
Insercao: 166.67 nanossegundos
Busca: 33.33 nanossegundos
Insercao: 200.00 nanossegundos
Busca: 0.00 nanossegundos
Insercao: 166.67 nanossegundos
Busca: 0.00 nanossegundos
Insercao: 166.67 nanossegundos
Busca: 33.33 nanossegundos
Insercao: 266.67 nanossegundos
Busca: 33.33 nanossegundos
Insercao: 300.00 nanossegundos
Busca: 33.33 nanossegundos
Insercao: 266.67 nanossegundos
Busca: 66.67 nanossegundos
Insercao: 233.33 nanossegundos
Busca: 33.33 nanossegundos
Insercao: 133.33 nanossegundos
Busca: 0.00 nanossegundos
Insercao: 133.33 nanossegundos
Busca: 0.00 nanossegundos
Insercao: 66.67 nanossegundos
Busca: 0.00 nanossegundos
Insercao: 200.00 nanossegundos
Busca: 0.00 nanossegundos
Insercao: 200.00 nanossegundos
Busca: 0.00 nanossegundos
Insercao: 100.00 nanossegundos
Busca: 0.00 nanossegundos
Insercao: 66.67 nanossegundos
Busca: 0.00 nanossegundos
Insercao: 66.67 nanossegundos
Busca: 0.00 nanossegundos
Insercao: 133.33 nanossegundos
Busca: 0.00 nanossegundos
Insercao: 166.67 nanossegundos
Busca: 0.00 nanossegundos
Insercao: 100.00 nanossegundos
Busca: 0.00 nanossegundos
Insercao: 66.67 nanossegundos
Busca: 0.00 nanossegundos
Insercao: 100.00 nanossegundos
Busca: 0.00 nanossegundos
Insercao: 100.00 nanossegundos
Busca: 0.00 nanossegundos
Pressione qualquer tecla para continuar. . .

Questão 3.1:


----------MENU----------
| 1 - CADASTRAR SERIE |
| 2 - CADASTRAR TEMPORADA |
| 3 - CADASTRAR PARTICIPANTE |
| 4 - IMPRIMIR SERIES |
| 5 - IMPRIMIR TODAS TEMPORADAS |
| 6 - IMPRIMIR PERSONAGENS |
| 7 - IMPRIMIR ARTISTAS |
| 0 - SAIR |
1
Informe o codigo da serie: 20
Informe o numero de temporadas: 5
Informe o nome da serie: stranger things
----------MENU----------
| 1 - CADASTRAR SERIE |
| 2 - CADASTRAR TEMPORADA |
| 3 - CADASTRAR PARTICIPANTE |
| 4 - IMPRIMIR SERIES |
| 5 - IMPRIMIR TODAS TEMPORADAS |
| 6 - IMPRIMIR PERSONAGENS |
| 7 - IMPRIMIR ARTISTAS |
| 0 - SAIR |
2
Informe o codigo da serie: 20
Serie encontrada =)
Informe o numero da temporada: 3
Informe o ano da temporada: 2016
Informe a quantidade de episodios da temporada: 8
Informe o nome da temporada: piloto
----------MENU----------
| 1 - CADASTRAR SERIE |
| 2 - CADASTRAR TEMPORADA |
| 3 - CADASTRAR PARTICIPANTE |
| 4 - IMPRIMIR SERIES |
| 5 - IMPRIMIR TODAS TEMPORADAS |
| 6 - IMPRIMIR PERSONAGENS |
| 7 - IMPRIMIR ARTISTAS |
| 0 - SAIR |
3
Informe o codigo da serie: 10
Esta serie nao existe.
----------MENU----------
| 1 - CADASTRAR SERIE |
| 2 - CADASTRAR TEMPORADA |
| 3 - CADASTRAR PARTICIPANTE |
| 4 - IMPRIMIR SERIES |
| 5 - IMPRIMIR TODAS TEMPORADAS |
| 6 - IMPRIMIR PERSONAGENS |
| 7 - IMPRIMIR ARTISTAS |
| 0 - SAIR |
3
Informe o codigo da serie: 20
Serie encontrada =)
Informe a temporada: 3
Informe o nome do artista: millie
Informe o nome do personagem: onze
Informe uma descricao sobre o personagem: tem super poderes
Personagem inserido na temporada.
----------MENU----------
| 1 - CADASTRAR SERIE |
| 2 - CADASTRAR TEMPORADA |
| 3 - CADASTRAR PARTICIPANTE |
| 4 - IMPRIMIR SERIES |
| 5 - IMPRIMIR TODAS TEMPORADAS |
| 6 - IMPRIMIR PERSONAGENS |
| 7 - IMPRIMIR ARTISTAS |
| 0 - SAIR |

```
Personagem inserido na temporada!
----------------------MENU-----------------------
| 1 - CADASTRAR SERIE                           |
| 2 - CADASTRAR TEMPORADA                       |
| 3 - CADASTRAR PARTICIPANTE                    |
| 4 - IMPRIMIR SERIES                           |
| 5 - IMPRIMIR TODAS TEMPORADAS                 |
| 6 - IMPRIMIR PERSONAGENS                      |
| 7 - IMPRIMIR ARTISTAS                         |
| 0 - SAIR                                      |
-------------------------------------------------
4
Codigo da serie: 20
Quantidade de temporadas: 5
Titulo da serie: stranger things

----------------------MENU-----------------------
| 1 - CADASTRAR SERIE                           |
| 2 - CADASTRAR TEMPORADA                       |
| 3 - CADASTRAR PARTICIPANTE                    |
| 4 - IMPRIMIR SERIES                           |
| 5 - IMPRIMIR TODAS TEMPORADAS                 |
| 6 - IMPRIMIR PERSONAGENS                      |
| 7 - IMPRIMIR ARTISTAS                         |
| 0 - SAIR                                      |
-------------------------------------------------
5
Informe o codigo da serie: 20
Ano: 2016
Temporada: 3
Quantidade de episodios: 8
Nome da temporada: piloto

----------------------MENU-----------------------
| 1 - CADASTRAR SERIE                           |
| 2 - CADASTRAR TEMPORADA                       |
| 3 - CADASTRAR PARTICIPANTE                    |
| 4 - IMPRIMIR SERIES                           |
| 5 - IMPRIMIR TODAS TEMPORADAS                 |
| 6 - IMPRIMIR PERSONAGENS                      |
| 7 - IMPRIMIR ARTISTAS                         |
| 0 - SAIR                                      |
-------------------------------------------------
6
Informe o codigo da serie: 20

Serie encontrada =)
Informe a temporada: 3
Nome do artista: millie
Nome do personagem: onze
Descricao do personagem: tem super poderes

----------------------MENU-----------------------
| 1 - CADASTRAR SERIE                           |
| 2 - CADASTRAR TEMPORADA                       |
| 3 - CADASTRAR PARTICIPANTE                    |
| 4 - IMPRIMIR SERIES                           |
| 5 - IMPRIMIR TODAS TEMPORADAS                 |
| 6 - IMPRIMIR PERSONAGENS                      |
| 7 - IMPRIMIR ARTISTAS                         |
| 0 - SAIR                                      |
-------------------------------------------------
7
Informe o codigo da serie: 20
Informe o nome do personagem: onze

TEMPORADA 1

Nome do artista: millie

----------------------MENU-----------------------
| 1 - CADASTRAR SERIE                           |
| 2 - CADASTRAR TEMPORADA                       |
| 3 - CADASTRAR PARTICIPANTE                    |
| 4 - IMPRIMIR SERIES                           |
| 5 - IMPRIMIR TODAS TEMPORADAS                 |
| 6 - IMPRIMIR PERSONAGENS                      |
| 7 - IMPRIMIR ARTISTAS                         |
| 0 - SAIR                                      |
-------------------------------------------------
```

Questão 3.2:



Analise de desempenho (Questões 2 e 3.2)

Para inserção de dados, a árvore AVL pode ser mais lenta devido às rotações necessárias para manter o equilíbrio. No entanto, garante que a árvore permaneça equilibrada, o que melhora o desempenho da busca posterior.

Para buscas, a árvore AVL é mais eficiente, especialmente em conjuntos de dados maiores, pois a árvore é mantida equilibrada, resultando em buscas com complexidade média O(log n). Na BST, a

eficiência da busca depende da distribuição dos elementos, e em casos desequilibrados, a busca pode ser muito mais lenta.

Para conjuntos de dados pequenos ou com distribuição de elementos favorável, a diferença de desempenho entre a árvore de busca binária e a árvore AVL pode não ser tão significativa.

Foram inseridos e buscados um total de 30.000 elementos em cada árvore, pode-se notar que em alguns casos de busca e inserção da árvore de busca essa media de tempo foi tão pequeno que chegou a 0.00000000... nanosegundos. Um dos fatores que podem ter levado a isso é a capacidade da máquina que o programa foi executado. Provavelmente, um a inserção e busca de uma quantidade de elementos superiores a 30.000 esse cenário seria muito diferente.

Conclusão: Os programas forneceram uma estrutura básica para gerenciar informações sobre séries de TV, temporadas e participantes, permitindo que o usuário insira, visualize e pesquise informações relacionadas a esses elementos. Além do mais, o experimento de verificação de tempo de inserção e de busca, se mostraram ser eficientes perante a máquina utilizada para testar os mesmos. Com tudo que já foi dito, podemos concluir que os programas se mostram eficiente nos resultados de saída exibidos. Todas demais informações já foram citadas, explanadas e detalhadas nos tópicos acima. Sendo assim, os programas conseguiram responder as problemáticas dos projetos e se mostram sem erros ou insuficiências. Com estes experimentos, foi possível desenvolver e manipular árvores binárias de busca e árvores binárias AVL através da Linguagem C, apesar de ser um projeto simples, a boa interpretação do problema é crucial para uma ótima aplicação.

Apêndice:

Questão 1:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


/*Prototipos*/
typedef struct arvtemporada ArvTemporada; //Arvore
typedef struct arvserie ArvSerie; //Arvore
typedef struct serie Serie;
typedef struct temporada Temporada;
typedef struct participantes Participantes;



ArvSerie* criaArvSerie();
ArvTemporada* criaArvTemp();
ArvSerie* buscaArvSerie(ArvSerie *raiz, int cod);
ArvSerie* buscaArvTempSerie(ArvSerie *raiz, int temp);

ArvTemporada* buscaArvTemp(ArvTemporada *raiz, int temporada);
void imprimirTodosArtistasPersonagem(ArvSerie *raiz, char *personagem);
```

```c
Serie ler_serie();
Temporada ler_temporada(int temp);

void imprimirSerie(Serie s);
void imprimirTemp(Temporada t);
void inserirArvSerie(ArvSerie **raiz, Serie s);
void imprimirArvTemp(ArvTemporada *raiz);


/*Structs*/

typedef struct serie
{
    int codigo;
    int num_temporadas;
    char titulo[50];
}Serie;

typedef struct temporada
{
    int num_temporada;
    int ano;
    int quantidade_ep;
    char titulo_temp[50];
}Temporada;

typedef struct participantes
{
    char nome_artista[50];
    char nome_personagem[50];
    char descricao_personagem[100];
    struct participantes *prox;
}Participantes;



/*Arvores*/

typedef struct arvserie
{
    Serie serie;
    ArvTemporada *arvTemp;
    struct arvserie *esq, *dir;
}ArvSerie;

typedef struct arvtemporada
{
    Temporada temp;
    Participantes *p; //lista
    struct arvtemporada *esq, *dir;
}ArvTemporada;
```

```c
/*Leituras*/

Serie ler_serie()
{
    Serie s;
    printf("Informe o codigo da serie: ");
    scanf("%d", &s.codigo);
    printf("Informe o numero de temporadas: ");
    scanf("%d", &s.num_temporadas);
    printf("Informe o nome da serie: ");
    //fflush(stdin); //utilize quando estiver em outro sistema sem ser o linux =)
    fflush(stdin); //utilize no sistema linux =)
    scanf("%[^\n]s", s.titulo);
    return s;
}

Temporada ler_temporada(int temp)
{

    Temporada t;
    t.num_temporada = temp;
    printf("Informe o ano da temporada: ");
    scanf("%d", &t.ano);
    fflush(stdin);
    printf("Informe a quantidade de episodios da temporada: ");
    scanf("%d", &t.quantidade_ep);
    fflush(stdin);
    printf("Informe o nome da temporada: ");
    //fflush(stdin); //utilize quando estiver em outro sistema sem ser o linux =)
    //utilize no sistema linux =)
    scanf("%[^\n]s", t.titulo_temp);
    fflush(stdin);

    return t;
}



/*Impressões*/
void imprimirSerie(Serie s)
{
    printf("Codigo da serie: %d\nQuantidade de temporadas: %d\nTitulo da serie:
%s\n\n", s.codigo, s.num_temporadas, s.titulo);
}

void imprimirTemp(Temporada t)
{
```

```
    printf("Ano: %d\nTemporada: %d\nQuantidade de episodios: %d\nNome da temporada:
%s\n\n",t.ano ,t.num_temporada, t.quantidade_ep, t.titulo_temp);
}

void imprimirParticipantes(Participantes* aux)
{
    printf("Nome do artista: %s\nNome do personagem: %s\nDescricao do personagem:
%s\n\n", aux->nome_artista, aux->nome_personagem, aux->descricao_personagem);
}


/*Implementação das Arvores*/

/*Series*/

ArvSerie* criaArvSerie()
{
    return NULL;
}

void inserirArvSerie(ArvSerie **raiz, Serie s)
{
    if (*raiz == NULL)
    {
        *raiz = (ArvSerie*)malloc(sizeof(ArvSerie));
        (*raiz)->esq = NULL;
        (*raiz)->dir = NULL;
        (*raiz)->arvTemp = criaArvTemp();
        (*raiz)->serie = s;
    }else{
        if ((*raiz)->serie.codigo > s.codigo)
            inserirArvSerie(&(*raiz)->esq, s);
        else if ((*raiz)->serie.codigo < s.codigo)
            inserirArvSerie(&(*raiz)->dir, s);
    }
}

ArvSerie* buscaArvSerie(ArvSerie *raiz, int cod)
{
    while (raiz)
    {
        if (raiz->serie.codigo > cod)
            raiz = raiz->esq;
        else if(raiz->serie.codigo < cod)
            raiz = raiz->dir;
        else
            return raiz;
    }

    return NULL;
}
```

```
ArvSerie* buscaArvTempSerie(ArvSerie *raiz, int temp)
{
    for (int i = 1; i <= raiz->serie.num_temporadas; i++)
    {
        if (i == temp)
        {
            return raiz;
        }

    }

    return NULL;
}


void imprimirArvSerie(ArvSerie *raiz)
{
    if (raiz)
    {
        imprimirArvSerie(raiz->esq);
        imprimirSerie(raiz->serie);
        imprimirArvSerie(raiz->dir);
    }

}

/*Temporada*/

ArvTemporada* criaArvTemp()
{
    return NULL;
}

void inserirArvTemp(ArvTemporada **raiz, Temporada t)
{
    if ((*raiz) == NULL)
    {
        *raiz = (ArvTemporada*)malloc(sizeof(ArvTemporada));
        (*raiz)->esq = NULL;
        (*raiz)->dir = NULL;
        (*raiz)->temp = t;
        (*raiz)->p = NULL;
    }else
        if ((*raiz)->temp.num_temporada > t.num_temporada)
            inserirArvTemp(&(*raiz)->esq, t);
        else if ((*raiz)->temp.num_temporada < t.num_temporada)
            inserirArvTemp(&(*raiz)->dir, t);
}

ArvTemporada* buscaArvTemp(ArvTemporada *raiz, int temporada)
```

```
{
    while(raiz)
    {
        if (raiz->temp.num_temporada > temporada)
            raiz = raiz->esq;
        else if (raiz->temp.num_temporada < temporada)
            raiz = raiz->dir;
        else
            return raiz;
    }

    return NULL;
}

void imprimirArvTemp(ArvTemporada *raiz)
{
    if (raiz)
    {
        imprimirArvTemp(raiz->esq);
        imprimirTemp(raiz->temp);
        imprimirArvTemp(raiz->dir);
    }

}

/*Implementação da lista*/

/*Participantes*/

ArvTemporada* criarLista()
{
    return NULL;
}

void inserirLista(ArvTemporada* lista, int temp)
{
    ArvTemporada* temp_serie = buscaArvTemp(lista, temp);
    if (temp_serie != NULL)
    {
        fflush(stdin);
        Participantes* aux = (Participantes*)malloc(sizeof(Participantes));
        printf("Informe o nome do artista: ");
        scanf("%[^\n]s", aux->nome_artista);
        fflush(stdin);
        printf("Informe o nome do personagem: ");
        scanf("%[^\n]s", aux->nome_personagem);
        fflush(stdin);
        printf("Informe uma descricao sobre o personagem: ");
        scanf("%[^\n]s", aux->descricao_personagem);
        fflush(stdin);
```

```
        if (temp_serie->p == NULL || strcmp(aux->nome_artista, temp_serie->p-
>nome_artista) < 0)
        {
            aux->prox = temp_serie->p;
            temp_serie->p = aux;
        }else
        {
            while (temp_serie->p->prox != NULL && strcmp(aux->nome_artista,
temp_serie->p->prox->nome_artista) > 0)
            {
                temp_serie->p = temp_serie->p->prox;
            }
            aux->prox = temp_serie->p->prox;
            temp_serie->p->prox = aux;
        }

    } else {
        printf("Temporada nao encontrada.\n");
    }
}


void imprimirTodosParticipantes(ArvTemporada *temp_serie_busca, int temporada)
{
    ArvTemporada *aux = buscaArvTemp(temp_serie_busca, temporada);
    if (aux != NULL) {
        Participantes *participante = aux->p;
        while (participante != NULL)
        {
            imprimirParticipantes(participante);
            participante = participante->prox;
        }
    } else
    {
        printf("Essa temporada nao existe.\n");
    }
}

void imprimirTodosArtistasPersonagem(ArvSerie *raiz, char *personagem)
{
    ArvTemporada *temporada = raiz->arvTemp;
    int i = 1;
    while (temporada != NULL) {
        Participantes *participante = temporada->p;
        printf("\nTEMPORADA %d\n\n", i);
        while (participante != NULL)
        {
            if (strcmp(participante->nome_personagem, personagem) == 0) {
                printf("Nome do artista: %s\n\n", participante->nome_artista);
            }
            participante = participante->prox;
```

```
        }
        i++;
        temporada = temporada->dir;
    }
}




/*MENU...*/
void menu()
{
    printf("----------------------MENU----------------------\n");
    printf("| 1 - CADASTRAR SERIE                          |\n");
    printf("| 2 - CADASTRAR TEMPORADA                      |\n");
    printf("| 3 - CADASTRAR PARTICIPANTE                   |\n");
    printf("| 4 - IMPRIMIR SERIES                          |\n");
    printf("| 5 - IMPRIMIR TODAS TEMPORADAS                |\n");
    printf("| 6 - IMPRIMIR PERSONAGENS                     |\n");
    printf("| 7 - IMPRIMIR ARTISTAS                        |\n");
    printf("| 0 - SAIR                                     |\n");
    printf("------------------------------------------------\n");

}

void libera_lista(Participantes *lista)
{
    Participantes *aux = lista;
    while (aux != NULL)
    {
        free(aux);
        aux = aux->prox;
    }

}

void liberarArvTemp(ArvTemporada *raiz) {
    if (raiz) {
        liberarArvTemp(raiz->esq);
        liberarArvTemp(raiz->dir);
        Participantes *p = raiz->p;
        while (p) {
            Participantes *temp = p;
            p = p->prox;
            free(temp);
        }
        free(raiz);
    }
}
```

```c
void liberarArvSerie(ArvSerie *raiz) {
    if (raiz) {
        liberarArvTemp(raiz->arvTemp);
        liberarArvSerie(raiz->esq);
        liberarArvSerie(raiz->dir);
        free(raiz);
    }
}

void liberarTodasSeries(ArvSerie *raiz) {
    liberarArvSerie(raiz);
}

int main()
{
    ArvSerie *raiz_s = criaArvSerie();

    int op = -1;
    int codigo;
    int temp;

    while (op != 0) {
        menu();
        scanf("%d", &op);
        switch(op){
            case 1:
                inserirArvSerie(&raiz_s, ler_serie());
                break;
            case 2:{
                printf("Informe o codigo da serie: ");
                scanf("%d", &codigo);
                ArvSerie *b = buscaArvSerie(raiz_s, codigo);
                if(b != NULL){
                    printf("\nSerie encontrada =)\n\n");
                    printf("Informe o numero da temporada: ");
                    scanf("%d", &temp);
                    ArvSerie* encontrou = buscaArvTempSerie(b ,temp);
                    if (encontrou != NULL){
                        inserirArvTemp(&(b->arvTemp), ler_temporada(temp));
                    }else
                    printf("Temporada nao existe.\n");
                }else{
                    printf("Esta serie nao existe.\n");
                }
                break;
            }
            case 3: {
                printf("Informe o codigo da serie: ");
                scanf("%d", &codigo);
                ArvSerie *b = buscaArvSerie(raiz_s, codigo);
                if (b != NULL) {
```

```
                    printf("\nSerie encontrada =)\n\n");
                    printf("Informe a temporada: ");
                    scanf("%d", &temp);
                    ArvTemporada *temp_serie = buscaArvTemp(b->arvTemp, temp);
                    if (temp_serie != NULL) {
                        inserirLista(temp_serie, temp);
                        printf("Personagem inserido na temporada.\n");
                    } else {
                        printf("Essa temporada nao existe.\n");
                    }
                } else {
                    printf("Esta serie nao existe.\n");
                }
                break;
            }
            case 4:
                imprimirArvSerie(raiz_s);
                break;
            case 5: {
                printf("Informe o codigo da serie: ");
                scanf("%d", &codigo);
                ArvSerie *b = buscaArvSerie(raiz_s, codigo);
                if (b != NULL) {
                    imprimirArvTemp(b->arvTemp);
                } else {
                    printf("Esta serie nao existe.\n");
                }
                break;
            }
            case 6: {
                printf("Informe o codigo da serie: ");
                scanf("%d", &codigo);
                ArvSerie *busca = buscaArvSerie(raiz_s, codigo);
                if (busca != NULL) {
                    printf("\nSerie encontrada =)\n");
                    printf("Informe a temporada: ");
                    scanf("%d", &temp);
                    ArvSerie *temp_serie_busca = buscaArvTempSerie(raiz_s, temp);
                    if (temp_serie_busca != NULL) {
                        imprimirTodosParticipantes(temp_serie_busca->arvTemp,
temp);
                    } else {
                        printf("Essa temporada nao existe.\n");
                    }
                } else {
                    printf("Esta serie nao existe.\n");
                }
                break;
            }
            case 7:
                printf("Informe o codigo da serie: ");
```

```
                scanf("%d", &codigo);
                ArvSerie *busca = buscaArvSerie(raiz_s, codigo);
                if (busca != NULL) {
                    char personagem[50];
                    printf("Informe o nome do personagem: ");
                    fflush(stdin);
                    scanf("%[^\n]s", personagem);
                    imprimirTodosArtistasPersonagem(busca, personagem);
                } else {
                    printf("Esta serie nao existe.\n");
                }
                break;
            default:
                if (op != 0)
                    printf("Opcao invalida!\n");
                break;
        }
    }
    liberarTodasSeries(raiz_s);
    libera_lista(raiz_s->arvTemp->p);

    return 0;
}
```

Questão 2:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>


typedef struct arvtemporada ArvTemporada;
typedef struct arvserie ArvSerie;
typedef struct serie Serie;
typedef struct temporada Temporada;




ArvSerie* criaArvSerie();
ArvTemporada* criaArvTemp();

void inserirArvSerie(ArvSerie **raiz, Serie s);
ArvTemporada* buscaArvTemp(ArvTemporada *raiz, int temporada);


Serie ler_serie();
Temporada ler_temporada(int temp);
```

```c
typedef struct serie
{
    int codigo;
    int num_temporadas;
    char titulo[50];
}Serie;

typedef struct temporada
{
    int num_temporada;
    int ano;
    int quantidade_ep;
    char titulo_temp[50];
}Temporada;



typedef struct arvserie
{
    Serie serie;
    ArvTemporada *arvTemp;
    struct arvserie *esq, *dir;
}ArvSerie;

typedef struct arvtemporada
{
    Temporada temp;
    struct arvtemporada *esq, *dir;
}ArvTemporada;


Serie ler_serie()
{
    Serie s;

    s.codigo = rand()%100 + 1;

    s.num_temporadas = rand()%4+1;

    strcpy(s.titulo, "oi");

    return s;
}


ArvSerie* criaArvSerie()
{
    return NULL;
}
```

```c
void inserirArvSerie(ArvSerie **raiz, Serie s)
{
    if (*raiz == NULL)
    {
        *raiz = (ArvSerie*)malloc(sizeof(ArvSerie));
        (*raiz)->esq = NULL;
        (*raiz)->dir = NULL;
        (*raiz)->arvTemp = criaArvTemp();
        (*raiz)->serie = s;
    }else{
        if ((*raiz)->serie.codigo > s.codigo)
            inserirArvSerie(&(*raiz)->esq, s);
        else if ((*raiz)->serie.codigo < s.codigo)
            inserirArvSerie(&(*raiz)->dir, s);
    }
}



ArvTemporada* criaArvTemp()
{
    return NULL;
}



ArvTemporada* buscaArvTemp(ArvTemporada *raiz, int temporada)
{
    while(raiz)
    {
        if (raiz->temp.num_temporada > temporada)
            raiz = raiz->esq;
        else if (raiz->temp.num_temporada < temporada)
            raiz = raiz->dir;
        else
            return raiz;
    }

    return NULL;
}


void liberarArvTemp(ArvTemporada *raiz) {
    if (raiz == NULL) {
        return;
    }
    liberarArvTemp(raiz->esq);
    liberarArvTemp(raiz->dir);
    free(raiz);
}
```

```c
void liberarArvSerie(ArvSerie *raiz) {
    if (raiz == NULL) {
        return;
    }
    liberarArvSerie(raiz->esq);
    liberarArvSerie(raiz->dir);
    liberarArvTemp(raiz->arvTemp);
    free(raiz);
}

int main()
{
    ArvSerie *raiz_s;
    for (int i = 0; i < 30; i++)
    {
        raiz_s = criaArvSerie();
        clock_t inicio, fim;
        double tempo;
        double media_busca = 0, media_insercao = 0;
        for (int j = 1; j <= 30000; j++)
        {
            inicio = clock();
            Serie nova_serie = ler_serie();
            inserirArvSerie(&raiz_s, nova_serie);
            fim = clock();
            tempo = (((double)(fim - inicio)) / CLOCKS_PER_SEC) * 1e9;
            media_insercao += tempo;
        }
        for (int j = 1; j <= 30000; j++)
        {
            inicio = clock();
            buscaArvTemp(raiz_s->arvTemp, 5);
            fim = clock();
            tempo = (((double)(fim - inicio)) / CLOCKS_PER_SEC) * 1e9;
            media_busca += tempo;
        }
        printf("Insercao: %.2lf nanossegundos\n", media_insercao/30000);
        printf("Busca: %.2lf nanossegundos\n",  media_busca/30000);
        printf("--------------------------------\n");
        liberarArvSerie(raiz_s);
    }

    return 0;
}
```

Questão 3.1:

```c
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <string.h>


/*Prototipos*/
typedef struct arvtemporada ArvTemporada; //Arvore
typedef struct arvserie ArvSerie; //Arvore
typedef struct serie Serie;
typedef struct temporada Temporada;
typedef struct participantes Participantes;



ArvSerie* criaArvSerie();
ArvTemporada* criaArvTemp();
ArvSerie* buscaArvSerie(ArvSerie *raiz, int cod);
ArvSerie* buscaArvTempSerie(ArvSerie *raiz, int temp);

ArvTemporada* buscaArvTemp(ArvTemporada *raiz, int temporada);
void imprimirTodosArtistasPersonagem(ArvSerie *raiz, char *personagem);

Serie ler_serie();
Temporada ler_temporada(int temp);

void imprimirSerie(Serie s);
void imprimirTemp(Temporada t);
void inserirArvSerie(ArvSerie **raiz, Serie s);
void imprimirArvTemp(ArvTemporada *raiz);

int alturaNOArvSerie(ArvSerie* raiz);
int fbArvSerie(ArvSerie* raiz);
int maiorArvSerie(int a, int b);
void rotacaoDEDArvSerie(ArvSerie **raiz);
void rotacaoDDEArvSerie(ArvSerie **raiz);
void rotacaoSDArvSerie(ArvSerie **raiz);
void rotacaoSEArvSerie(ArvSerie **raiz);

int alturaNOArvTemp(ArvTemporada* raiz);
int fbArvTemp(ArvTemporada* raiz);
int maiorArvTemp(int a, int b);
void rotacaoSDArvTemp(ArvTemporada **raiz);
void rotacaoSEArvTemp(ArvTemporada **raiz);
void rotacaoDEDArvTemp(ArvTemporada **raiz);
void rotacaoDDEArvTemp(ArvTemporada **raiz);


/*Structs*/

typedef struct serie
{
    int codigo;
    int num_temporadas;
```

```
    char titulo[50];
}Serie;



typedef struct temporada
{
    int num_temporada;
    int quantidade_ep;
    int ano;
    char titulo_temp[50];
}Temporada;

typedef struct participantes
{
    char nome_artista[50];
    char nome_personagem[50];
    char descricao_personagem[100];
    struct participantes *prox;
}Participantes;



/*Arvores*/

typedef struct arvserie
{
    Serie serie;
    ArvTemporada *arvTemp;
    int alt;
    struct arvserie *esq, *dir;
}ArvSerie;

typedef struct arvtemporada
{
    Temporada temp;
    Participantes *p; //lista
    int alt;
    struct arvtemporada *esq, *dir;
}ArvTemporada;



/*Leituras*/

Serie ler_serie()
{
    Serie s;
    printf("Informe o codigo da serie: ");
    scanf("%d", &s.codigo);
    fflush(stdin);
```

```
    printf("Informe o numero de temporadas: ");
    scanf("%d", &s.num_temporadas);
    fflush(stdin);
    printf("Informe o nome da serie: ");
    scanf("%[^\n]s", s.titulo);
    fflush(stdin);
    return s;
}

Temporada ler_temporada(int temp)
{

    Temporada t;
    t.num_temporada = temp;
    printf("Informe o ano da temporada: ");
    scanf("%d", &t.ano);
    fflush(stdin);
    printf("Informe a quantidade de episodios da temporada: ");
    scanf("%d", &t.quantidade_ep);
    fflush(stdin);
    printf("Informe o nome da temporada: ");
    scanf("%[^\n]s", t.titulo_temp);
    fflush(stdin);

    return t;
}



/*Impressões*/
void imprimirSerie(Serie s)
{
    printf("Codigo da serie: %d\nQuantidade de temporadas: %d\nTitulo da serie:
%s\n\n", s.codigo, s.num_temporadas, s.titulo);
}

void imprimirTemp(Temporada t)
{
    printf("Ano: %d\nTemporada: %d\nQuantidade de episodios: %d\nNome da temporada:
%s\n\n",t.ano ,t.num_temporada, t.quantidade_ep, t.titulo_temp);
}

void imprimirParticipantes(Participantes* aux)
{
    printf("Nome do artista: %s\nNome do personagem: %s\nDescricao do personagem:
%s\n\n", aux->nome_artista, aux->nome_personagem, aux->descricao_personagem);
}

/*Implementação das Arvores*/

/*Series*/
```

```
ArvSerie* criaArvSerie()
{
    return NULL;
}

void inserirArvSerie(ArvSerie **raiz, Serie s)
{
    if (*raiz == NULL)
    {
        *raiz = (ArvSerie*)malloc(sizeof(ArvSerie));
        (*raiz)->serie = s;
        (*raiz)->alt = 0;
        (*raiz)->esq = NULL;
        (*raiz)->dir = NULL;
        (*raiz)->arvTemp = NULL;
    }
    else if (s.codigo < (*raiz)->serie.codigo)
    {
        inserirArvSerie(&(*raiz)->esq, s);
    }
    else if (s.codigo > (*raiz)->serie.codigo)
    {
        inserirArvSerie(&(*raiz)->dir, s);
    }


    (*raiz)->alt = maiorArvSerie(alturaNOArvSerie((*raiz)->esq),
alturaNOArvSerie((*raiz)->dir)) + 1;


    int fb = fbArvSerie(*raiz);
    if (fb == 2)
    {
        if (fbArvSerie((*raiz)->esq) == 1)
            rotacaoSDArvSerie(raiz);
        else
            rotacaoDEDArvSerie(raiz);
    }
    else if (fb == -2)
    {
        if (fbArvSerie((*raiz)->dir) == -1)
            rotacaoSEArvSerie(raiz);
        else
            rotacaoDDEArvSerie(raiz);
    }
}

ArvSerie* buscaArvSerie(ArvSerie *raiz, int cod)
{
    while (raiz)
```

```
    {
        if (raiz->serie.codigo > cod)
            raiz = raiz->esq;
        else if(raiz->serie.codigo < cod)
            raiz = raiz->dir;
        else
            return raiz;
    }

    return NULL;
}

ArvSerie* buscaArvTempSerie(ArvSerie *raiz, int temp)
{
    for (int i = 1; i <= raiz->serie.num_temporadas; i++)
    {
        if (i == temp)
        {
            return raiz;
        }

    }

    return NULL;
}

int alturaNOArvSerie(ArvSerie* raiz)
{
    if (raiz == NULL)
        return -1;
    else
        return raiz->alt;
}

int fbArvSerie(ArvSerie* raiz)
{
    return alturaNOArvSerie(raiz->esq) - alturaNOArvSerie(raiz->dir);
}

int maiorArvSerie(int a, int b)
{
    return (a > b) ? a : b;
}

void rotacaoSDArvSerie(ArvSerie **raiz)
{
    ArvSerie *aux = (*raiz)->esq;
    (*raiz)->esq = aux->dir;
    aux->dir = *raiz;
    (*raiz)->alt = maiorArvSerie(alturaNOArvSerie((*raiz)->esq),
alturaNOArvSerie((*raiz)->dir)) + 1;
```

```
    aux->alt = maiorArvSerie(alturaNOArvSerie(aux->esq), (*raiz)->alt) + 1;
    *raiz = aux;
}

void rotacaoSEArvSerie(ArvSerie **raiz)
{
    ArvSerie *aux = (*raiz)->dir;
    (*raiz)->dir = aux->esq;
    aux->esq = *raiz;
    (*raiz)->alt = maiorArvSerie(alturaNOArvSerie((*raiz)->esq),
alturaNOArvSerie((*raiz)->dir)) + 1;
    aux->alt = maiorArvSerie(alturaNOArvSerie(aux->dir), (*raiz)->alt) + 1;
    *raiz = aux;
}

void rotacaoDEDArvSerie(ArvSerie **raiz)
{
    rotacaoSEArvSerie(&(*raiz)->esq);
    rotacaoSDArvSerie(raiz);
}

void rotacaoDDEArvSerie(ArvSerie **raiz)
{
    rotacaoSDArvSerie(&(*raiz)->dir);
    rotacaoSEArvSerie(raiz);
}


void imprimirArvSerie(ArvSerie *raiz)
{
    if (raiz)
    {
        imprimirArvSerie(raiz->esq);
        imprimirSerie(raiz->serie);
        imprimirArvSerie(raiz->dir);
    }

}

/*Temporada*/

ArvTemporada* criaArvTemp()
{
    return NULL;
}

void inserirArvTemp(ArvTemporada **raiz, Temporada t)
{
    if (*raiz == NULL)
    {
        *raiz = (ArvTemporada*)malloc(sizeof(ArvTemporada));
```

```
        (*raiz)->temp = t;
        (*raiz)->alt = 0;
        (*raiz)->esq = NULL;
        (*raiz)->dir = NULL;
        (*raiz)->p = NULL;
    }
    else if (t.num_temporada < (*raiz)->temp.num_temporada)
    {
        inserirArvTemp(&(*raiz)->esq, t);
    }
    else if (t.num_temporada > (*raiz)->temp.num_temporada)
    {
        inserirArvTemp(&(*raiz)->dir, t);
    }


    (*raiz)->alt = maiorArvTemp(alturaNOArvTemp((*raiz)->esq),
alturaNOArvTemp((*raiz)->dir)) + 1;


    int fb = fbArvTemp(*raiz);
    if (fb == 2)
    {
        if (fbArvTemp((*raiz)->esq) == 1)
            rotacaoSDArvTemp(raiz);
        else
            rotacaoDEDArvTemp(raiz);
    }
    else if (fb == -2)
    {
        if (fbArvTemp((*raiz)->dir) == -1)
            rotacaoSEArvTemp(raiz);
        else
            rotacaoDDEArvTemp(raiz);
    }
}

ArvTemporada* buscaArvTemp(ArvTemporada *raiz, int temporada)
{
    while(raiz)
    {
        if (raiz->temp.num_temporada > temporada)
            raiz = raiz->esq;
        else if (raiz->temp.num_temporada < temporada)
            raiz = raiz->dir;
        else
            return raiz;
    }

    return NULL;
}
```

```c
void imprimirArvTemp(ArvTemporada *raiz)
{
    if (raiz)
    {
        imprimirArvTemp(raiz->esq);
        imprimirTemp(raiz->temp);
        imprimirArvTemp(raiz->dir);
    }

}

int alturaNOArvTemp(ArvTemporada* raiz)
{
    if (raiz == NULL)
        return -1;
    else
        return raiz->alt;
}

int fbArvTemp(ArvTemporada* raiz)
{
    return alturaNOArvTemp(raiz->esq) - alturaNOArvTemp(raiz->dir);
}

int maiorArvTemp(int a, int b)
{
    return (a > b) ? a : b;
}

void rotacaoSDArvTemp(ArvTemporada **raiz)
{
    ArvTemporada *aux = (*raiz)->esq;
    (*raiz)->esq = aux->dir;
    aux->dir = *raiz;
    (*raiz)->alt = maiorArvTemp(alturaNOArvTemp((*raiz)->esq),
alturaNOArvTemp((*raiz)->dir)) + 1;
    aux->alt = maiorArvTemp(alturaNOArvTemp(aux->esq), (*raiz)->alt) + 1;
    *raiz = aux;
}

void rotacaoSEArvTemp(ArvTemporada **raiz)
{
    ArvTemporada *aux = (*raiz)->dir;
    (*raiz)->dir = aux->esq;
    aux->esq = *raiz;
    (*raiz)->alt = maiorArvTemp(alturaNOArvTemp((*raiz)->esq),
alturaNOArvTemp((*raiz)->dir)) + 1;
    aux->alt = maiorArvTemp(alturaNOArvTemp(aux->dir), (*raiz)->alt) + 1;
    *raiz = aux;
}
```

```c
void rotacaoDEDArvTemp(ArvTemporada **raiz)
{
    rotacaoSEArvTemp(&(*raiz)->esq);
    rotacaoSDArvTemp(raiz);
}

void rotacaoDDEArvTemp(ArvTemporada **raiz)
{
    rotacaoSDArvTemp(&(*raiz)->dir);
    rotacaoSEArvTemp(raiz);
}

/*Implementação da lista*/

/*Participantes*/

ArvTemporada* criarLista()
{
    return NULL;
}

void inserirLista(ArvTemporada* lista, int temp)
{
    ArvTemporada* temp_serie = buscaArvTemp(lista, temp);
    if (temp_serie != NULL)
    {
        Participantes* aux = (Participantes*)malloc(sizeof(Participantes));
        printf("Informe o nome do artista: ");
        fflush(stdin);
        scanf("%[^\n]s", aux->nome_artista);
        printf("Informe o nome do personagem: ");
        fflush(stdin);
        scanf("%[^\n]s", aux->nome_personagem);
        printf("Informe uma descricao sobre o personagem: ");
        fflush(stdin);
        scanf("%[^\n]s", aux->descricao_personagem);
        if (temp_serie->p == NULL || strcmp(aux->nome_artista, temp_serie->p-
>nome_artista) < 0)
        {
            aux->prox = temp_serie->p;
            temp_serie->p = aux;
        }else
        {
            while (temp_serie->p->prox != NULL && strcmp(aux->nome_artista,
temp_serie->p->prox->nome_artista) > 0)
            {
                temp_serie->p = temp_serie->p->prox;
            }
            aux->prox = temp_serie->p->prox;
            temp_serie->p->prox = aux;
```

```
        }

    } else {
        printf("Temporada nao encontrada.\n");
    }
}


void imprimirTodosParticipantes(ArvTemporada *temp_serie_busca, int temporada)
{
    ArvTemporada *aux = buscaArvTemp(temp_serie_busca, temporada);
    if (aux != NULL) {
        Participantes *participante = aux->p;
        while (participante != NULL)
        {
            imprimirParticipantes(participante);
            participante = participante->prox;
        }
    } else
    {
        printf("Essa temporada nao existe.\n");
    }
}

void imprimirTodosArtistasPersonagem(ArvSerie *raiz, char *personagem)
{
    ArvTemporada *temporada = raiz->arvTemp;
    int i = 1;
    while (temporada != NULL) {
        Participantes *participante = temporada->p;
        printf("\nTEMPORADA %d\n\n", i);
        while (participante != NULL)
        {
            if (strcmp(participante->nome_personagem, personagem) == 0) {
                printf("Nome do artista: %s\n\n", participante->nome_artista);
            }
            participante = participante->prox;
        }
        i++;
        temporada = temporada->dir;
    }
}


void liberarArvTemp(ArvTemporada *raiz) {
    if (raiz) {
        liberarArvTemp(raiz->esq);
        liberarArvTemp(raiz->dir);
        Participantes *p = raiz->p;
        while (p) {
            Participantes *temp = p;
```

```
            p = p->prox;
            free(temp);
        }
        free(raiz);
    }
}

/* Libera a memória de uma árvore de séries */
void liberarArvSerie(ArvSerie *raiz) {
    if (raiz) {
        liberarArvTemp(raiz->arvTemp);  // Libera a árvore de temporadas associada
à série
        liberarArvSerie(raiz->esq);
        liberarArvSerie(raiz->dir);
        free(raiz);
    }
}

/* Função para liberar a memória de todas as séries */
void liberarTodasSeries(ArvSerie *raiz) {
    liberarArvSerie(raiz);
}

void libera_lista(Participantes *lista)
{
    Participantes *aux = lista;
    while (aux != NULL)
    {
        free(aux);
        aux = aux->prox;
    }

}



/*MENU...*/
void menu()
{
    printf("----------------------MENU----------------------\n");
    printf("| 1 - CADASTRAR SERIE                         |\n");
    printf("| 2 - CADASTRAR TEMPORADA                     |\n");
    printf("| 3 - CADASTRAR PARTICIPANTE                  |\n");
    printf("| 4 - IMPRIMIR SERIES                         |\n");
    printf("| 5 - IMPRIMIR TODAS TEMPORADAS               |\n");
    printf("| 6 - IMPRIMIR PERSONAGENS                    |\n");
    printf("| 7 - IMPRIMIR ARTISTAS                       |\n");
    printf("| 0 - SAIR                                    |\n");
    printf("-----------------------------------------------\n");

}
```

```c
int main()
{
    ArvSerie *raiz_s = criaArvSerie();

    int op = -1;
    int codigo;
    int temp;

    while (op != 0) {
        menu();
        scanf("%d", &op);
        switch (op) {
            case 1:
                inserirArvSerie(&raiz_s,ler_serie());
                break;
            case 2: {
                printf("Informe o codigo da serie: ");
                scanf("%d", &codigo);
                ArvSerie *b = buscaArvSerie(raiz_s, codigo);

                if (b != NULL) {
                    printf("\nSerie encontrada =)\n\n");
                    printf("Informe o numero da temporada: ");
                    scanf("%d", &temp);
                    ArvSerie* encontrou = buscaArvTempSerie(b ,temp);
                    if (encontrou != NULL)
                    {

                        inserirArvTemp(&(b->arvTemp), ler_temporada(temp));
                    }else
                        printf("Temporada nao
existe.\n");
                } else {
                    printf("Esta serie nao existe.\n");
                }
                break;
            }
            case 3: {
                printf("Informe o codigo da serie: ");
                scanf("%d", &codigo);
                ArvSerie *b = buscaArvSerie(raiz_s, codigo);
                if (b != NULL) {
                    printf("\nSerie encontrada =)\n\n");
                    printf("Informe a temporada: ");
                    scanf("%d", &temp);
                    ArvTemporada *temp_serie = buscaArvTemp(b->arvTemp, temp);
                    if (temp_serie != NULL) {
                        inserirLista(temp_serie, temp);
                        printf("Personagem inserido na temporada.\n");
```

```
                    } else {
                        printf("Essa temporada nao existe.\n");
                    }
                } else {
                    printf("Esta serie nao existe.\n");
                }
                break;
            }
            case 4:
                imprimirArvSerie(raiz_s);
                break;
            case 5: {
                printf("Informe o codigo da serie: ");
                scanf("%d", &codigo);
                ArvSerie *b = buscaArvSerie(raiz_s, codigo);
                if (b != NULL) {
                    imprimirArvTemp(b->arvTemp);
                } else {
                    printf("Esta serie nao existe.\n");
                }
                break;
            }
            case 6: {
                printf("Informe o codigo da serie: ");
                scanf("%d", &codigo);
                ArvSerie *busca = buscaArvSerie(raiz_s, codigo);
                if (busca != NULL) {
                    printf("\nSerie encontrada =)\n");
                    printf("Informe a temporada: ");
                    scanf("%d", &temp);
                    ArvSerie *temp_serie_busca = buscaArvTempSerie(raiz_s, temp);
                    if (temp_serie_busca != NULL) {
                        imprimirTodosParticipantes(temp_serie_busca->arvTemp,
temp);
                    } else {
                        printf("Essa temporada nao existe.\n");
                    }
                } else {
                    printf("Esta serie nao existe.\n");
                }
                break;
            }
            case 7:
                printf("Informe o codigo da serie: ");
                scanf("%d", &codigo);
                fflush(stdin);
                ArvSerie *busca = buscaArvSerie(raiz_s, codigo);
                if (busca != NULL) {
                    char personagem[50];
                    printf("Informe o nome do personagem: ");
                    scanf("%[^\n]s", personagem);
```

```
                    fflush(stdin);
                    imprimirTodosArtistasPersonagem(busca, personagem);
                } else {
                    printf("Esta serie nao existe.\n");
                }
                break;
            default:
                if (op != 0)
                    printf("Opcao invalida!\n");

                break;
        }
    }

    liberarTodasSeries(raiz_s);
    libera_lista(raiz_s->arvTemp->p);

    return 0;
}
```

Questão 3.2:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>



typedef struct arvtemporada ArvTemporada;
typedef struct arvserie ArvSerie;
typedef struct serie Serie;
typedef struct temporada Temporada;

ArvSerie* criaArvSerie();
ArvTemporada* criaArvTemp();
ArvTemporada* buscaArvTemp(ArvTemporada *raiz, int temporada);


Serie ler_serie();

void inserirArvSerie(ArvSerie **raiz, Serie s);

int alturaNOArvSerie(ArvSerie* raiz);
int fbArvSerie(ArvSerie* raiz);
int maiorArvSerie(int a, int b);
void rotacaoDEDArvSerie(ArvSerie **raiz);
void rotacaoDDEArvSerie(ArvSerie **raiz);
void rotacaoSDArvSerie(ArvSerie **raiz);
```

```c
void rotacaoSEArvSerie(ArvSerie **raiz);



typedef struct serie
{
    int codigo;
    int num_temporadas;
    char titulo[50];
}Serie;



typedef struct temporada
{
    int num_temporada;
    int quantidade_ep;
    int ano;
    char titulo_temp[50];
}Temporada;



typedef struct arvserie
{
    Serie serie;
    ArvTemporada *arvTemp;
    int alt;
    struct arvserie *esq, *dir;
}ArvSerie;

typedef struct arvtemporada
{
    Temporada temp;
    int alt;
    struct arvtemporada *esq, *dir;
}ArvTemporada;




Serie ler_serie()
{
    Serie s;

    s.codigo = rand()%100 + 1;

    s.num_temporadas = rand()%5 + 1;

    strcpy(s.titulo, "oi");
```

```
    return s;
}



ArvSerie* criaArvSerie()
{
    return NULL;
}

void inserirArvSerie(ArvSerie **raiz, Serie s)
{
    if (*raiz == NULL)
    {
        *raiz = (ArvSerie*)malloc(sizeof(ArvSerie));
        (*raiz)->serie = s;
        (*raiz)->alt = 0;
        (*raiz)->esq = NULL;
        (*raiz)->dir = NULL;
        (*raiz)->arvTemp = criaArvTemp();
    }
    else if (s.codigo < (*raiz)->serie.codigo)
    {
        inserirArvSerie(&(*raiz)->esq, s);
    }
    else if (s.codigo > (*raiz)->serie.codigo)
    {
        inserirArvSerie(&(*raiz)->dir, s);
    }

    (*raiz)->alt = maiorArvSerie(alturaNOArvSerie((*raiz)->esq),
alturaNOArvSerie((*raiz)->dir)) + 1;

    int fb = fbArvSerie(*raiz);
    if (fb == 2)
    {
        if (fbArvSerie((*raiz)->esq) == 1)
            rotacaoSDArvSerie(raiz);
        else
            rotacaoDEDArvSerie(raiz);
    }
    else if (fb == -2)
    {
        if (fbArvSerie((*raiz)->dir) == -1)
            rotacaoSEArvSerie(raiz);
        else
            rotacaoDDEArvSerie(raiz);
    }
}
```

```c
int alturaNOArvSerie(ArvSerie* raiz)
{
    if (raiz == NULL)
        return -1;
    else
        return raiz->alt;
}

int fbArvSerie(ArvSerie* raiz)
{
    return alturaNOArvSerie(raiz->esq) - alturaNOArvSerie(raiz->dir);
}

int maiorArvSerie(int a, int b)
{
    return (a > b) ? a : b;
}

void rotacaoSDArvSerie(ArvSerie **raiz)
{
    ArvSerie *aux = (*raiz)->esq;
    (*raiz)->esq = aux->dir;
    aux->dir = *raiz;
    (*raiz)->alt = maiorArvSerie(alturaNOArvSerie((*raiz)->esq),
alturaNOArvSerie((*raiz)->dir)) + 1;
    aux->alt = maiorArvSerie(alturaNOArvSerie(aux->esq), (*raiz)->alt) + 1;
    *raiz = aux;
}

void rotacaoSEArvSerie(ArvSerie **raiz)
{
    ArvSerie *aux = (*raiz)->dir;
    (*raiz)->dir = aux->esq;
    aux->esq = *raiz;
    (*raiz)->alt = maiorArvSerie(alturaNOArvSerie((*raiz)->esq),
alturaNOArvSerie((*raiz)->dir)) + 1;
    aux->alt = maiorArvSerie(alturaNOArvSerie(aux->dir), (*raiz)->alt) + 1;
    *raiz = aux;
}

void rotacaoDEDArvSerie(ArvSerie **raiz)
{
    rotacaoSEArvSerie(&(*raiz)->esq);
    rotacaoSDArvSerie(raiz);
}

void rotacaoDDEArvSerie(ArvSerie **raiz)
{
    rotacaoSDArvSerie(&(*raiz)->dir);
    rotacaoSEArvSerie(raiz);
}
```

```c
ArvTemporada* criaArvTemp()
{
    return NULL;
}


ArvTemporada* buscaArvTemp(ArvTemporada *raiz, int temporada)
{
    while(raiz)
    {
        if (raiz->temp.num_temporada > temporada)
            raiz = raiz->esq;
        else if (raiz->temp.num_temporada < temporada)
            raiz = raiz->dir;
        else
            return raiz;
    }

    return NULL;
}

void liberarArvTemp(ArvTemporada *raiz) {
    if (raiz == NULL) {
        return;
    }
    liberarArvTemp(raiz->esq);
    liberarArvTemp(raiz->dir);
    free(raiz);
}


void liberarArvSerie(ArvSerie *raiz) {
    if (raiz == NULL) {
        return;
    }
    liberarArvSerie(raiz->esq);
    liberarArvSerie(raiz->dir);
    liberarArvTemp(raiz->arvTemp);
    free(raiz);
}

int main()
{
    ArvSerie *raiz_s;
    for (int i = 0; i < 30; i++)
    {
        raiz_s = criaArvSerie();
```

```
        clock_t inicio, fim;
        double tempo;
        double media_busca = 0, media_insercao = 0;
        for (int j = 1; j <= 30000; j++)
        {
            inicio = clock();
            Serie nova_serie = ler_serie();
            inserirArvSerie(&raiz_s, nova_serie);
            fim = clock();
            tempo = (((double)(fim - inicio)) / CLOCKS_PER_SEC) * 1e9;
            media_insercao += tempo;
        }
        for (int j = 1; j <= 30000; j++)
        {
            inicio = clock();
            int temp = rand()%5 + 1;
            buscaArvTemp(raiz_s->arvTemp, temp);
            fim = clock();
            tempo = (((double)(fim - inicio)) / CLOCKS_PER_SEC) * 1e9;
            media_busca += tempo;
        }
        printf("Insercao: %.2lf nanossegundos\n", media_insercao/30000);
        printf("Busca: %.2lf nanossegundos\n",  media_busca/30000);
        printf("--------------------------------\n");
        liberarArvSerie(raiz_s);
    }

    return 0;
}
```