



Ministério da Educação
Universidade Federal do Piauí – UFPI
Campus Senador Helvídio Nunes de Barros – Picos
Curso Bacharelado em Sistemas de Informação
Professora: Juliana Oliveira de Carvalho
Discente: Crisly Maria Silva dos Santos
Disciplina: Estrutura de Dados II – 2023.2



Relatório Trabalho I

Resumo do projeto

Esse relatório, teve como objetivo demonstrar a prática e implementação de Estrutura de Árvores de busca binária e árvores AVL para resolução de alguns problemas práticos propostos, utilizando a linguagem de programação C. O mesmo abordará as etapas de cada questão e apresentará suas respectivas implementações de maneira detalhada. Oferecendo uma visão abrangente das aplicações dessas estruturas de dados.

Introdução

Nesta seção será apresentado a Introdução do projeto. Estruturas de dados são fundamentais para a programação e muito se fala sobre a importância do estudo desses temas. Elas representam 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. Basicamente, estruturas de dados são objetos que contêm dados, na forma de campos, muitas vezes conhecidos como atributos; e código, na forma de procedimentos, muitas vezes conhecidos como métodos.

Uma árvore binária é uma estrutura de dados que possui um conjunto finito de elementos, que pode ser vazio ou com um único elemento designado como raiz. Cada um dos restantes elementos está constituído por duas árvores binárias disjuntas. As Árvores Binárias são estruturas eficientes para busca, inserção e remoção de elementos, permitindo a recuperação rápida dos dados armazenados. Além disso, as árvores binárias são fundamentais em muitos algoritmos de Aprendizado de Máquina, como árvores de decisão e florestas aleatórias.

Uma árvore AVL é um tipo de árvore binária de busca. Ela foi nomeada em homenagem aos seus criadores, Adelson-Velsky e Landis. As árvores AVL possuem a propriedade de auto balanceamento dinâmico, além de todas as

outras propriedades mostradas pelas árvores binárias de busca. A definição formal de uma árvore AVL é que ela é uma árvore binária onde, para qualquer nó v , a diferença entre a altura das subárvores esquerda $he(v)$ e direita $hd(v)$ é no máximo em módulo igual a 1234. Isso significa que para todos os nós de uma árvore AVL, a diferença entre as alturas das subárvores esquerda e direita deve ser -1, 0 ou 1.

Na continuação do projeto tem como objetivo mostrar ao leitor experimentos e justificativas acerca dos problemas resolvidos nessa estrutura de dados. A seção Seções Específicas mostrará os aspectos funcionais presentes no programa, a seção Resultados da Execução do Programa mostrará resultados através da execução do programa, bem como se familiarizar-se com o propósito do mesmo. A seção Conclusão reforça a informação já apresentada no projeto, bem como apresenta resumidamente o experimento. E por fim, a seção Apêndice contém os códigos fontes que foram produzidos como parte do experimento.

Seções Específicas

Nesta seção será apresentada as Seções Específicas. Neste trabalho foram fornecidos três problemas. O primeiro problema tem como propósito criar um programa em C para cadastrar as seguintes árvores:

Series: código, título, número de temporadas, um endereço para uma árvore binária de busca contendo informações de cada temporada. A árvore deve ser organizada pelo código da série.

Temporadas: número da temporada, título, quantidade de episódios, ano, endereço para uma lista simples de participantes. A árvore de Temporadas deve ser organizada pelo número da temporada. Para a lista simples de participantes deve ter: nome artista, nome personagem e uma descrição sobre o personagem. A lista simples de participantes deve estar em ordem alfabética do nome do artista.

Além de ter um menu com as seguintes opções:

- (a) Imprimir em ordem pelo código da série: o título, o número de temporadas.
- (b) Imprimir os dados de todas as temporadas de uma série, cujo usuário informe o código da série.
- (c) Imprimir todos os personagens de uma determinada temporada de uma dada série, cujo usuário informe o código da série e o número da temporada.

(d)Imprimir o nome de todos os artistas que interpretaram um determinado personagem em todas as temporadas de uma dada série, cujo usuário informe o código da série.

O segundo problema tem a mesma finalidade do primeiro, apenas vai ter acréscimo de algumas funcionalidades como verificar o tempo, utilizando a função time do C para realizar a inserção para cada elemento na árvore de séries, ou seja, o tempo de código, título, número de temporadas, no momento da inserção. Além disso, verificar também o tempo de busca de uma determinada temporada na árvore de temporadas. Na mesma questão ainda é apresentado algumas observações que devem ser feitas.

Obs. 1: para os experimentos cada execução deve inserir os mesmos códigos de séries em ordem diferente(pode utilizar comando para embaralhar os códigos e assim as árvores ficarem bem aleatórias e poder verificar a diferença entre inserções).

Obs. 2: Lembre-se que não pode haver impressão entre o tempo inicial e o tempo final, pois impressão consome muito tempo.

Obs. 3: Para validar o tempo de busca, o mesmo deve ser repetido 30 vezes. Faça uma média para obter o resultado final

E por último o terceiro problema que tem como finalidade as mesmas funcionalidades do problema 01, e problema 02. Ou seja, na questão 03 pede que repita todo o processo utilizado nas questões anteriores, mas agora implementando o código usando uma árvore AVL. Será analisada a diferença de inserção e busca de uma árvore binária de busca e uma AVL. É importante destacar que as árvores binárias AVL são balanceadas no momento da inserção, já na binária a inserção ocorre normalmente. Com isso, o objetivo central deste último problema é comparar o tempo de execução das operações de inserção e busca em uma árvore binária de busca e uma árvore AVL.

O ambiente para programação na resolução das questões propostas foi escolhido VScode por ser uma ferramenta ampla de funcionalidades e por apresentar uma interface agradável e versátil.

Resultados da Execução do Programa

Nesta seção será apresentado os resultados da execução dos programas realizados neste trabalho. Primeiramente será evidenciado o problema 01. A figura 01 mostra o menu desta questão.

```
1 - Cadastrar Serie.
2 - Cadastrar Temporada.
3 - Imprimir series.
4 - Imprimir os dados de todas as temporadas de uma serie.
5 - Imprimir todos os personagens de uma determinada temporada.
6 - Imprimir o nome de todos os artistas que interpretaram um determinado personagem em todas as temporadas.
5.
7 - Sair

Escolha uma opcao:
```

Figura 01 - Menu Questão 01.

Serão apresentados agora os testes das opções 02, 03, 04, 05, 06. Essas opções serão disponibilizadas ao usuário apenas no momento em que o mesmo cadastrar uma série, pois não tem como prosseguir com as funcionalidades seguintes se o mesmo não tiver uma série cadastrada. Na figura 02 será mostrado os testes, na mesma foram feitos os testes da opção 02, 03, 04.

```
1 - Cadastrar Serie.
2 - Cadastrar Temporada.
3 - Imprimir series.
4 - Imprimir os dados de todas as temporadas de uma serie.
5 - Imprimir todos os personagens de uma determinada temporada.
6 - Imprimir o nome de todos os artistas que interpretaram um determinado personagem em todas as temporadas.
7 - Sair

Escolha uma opcao:
2
Nenhuma Serie Cadastrada No Momento!

1 - Cadastrar Serie.
2 - Cadastrar Temporada.
3 - Imprimir series.
4 - Imprimir os dados de todas as temporadas de uma serie.
5 - Imprimir todos os personagens de uma determinada temporada.
6 - Imprimir o nome de todos os artistas que interpretaram um determinado personagem em todas as temporadas.
7 - Sair

Escolha uma opcao:
3
Nenhuma Serie Cadastrada No Momento!

1 - Cadastrar Serie.
2 - Cadastrar Temporada.
3 - Imprimir series.
4 - Imprimir os dados de todas as temporadas de uma serie.
5 - Imprimir todos os personagens de uma determinada temporada.
6 - Imprimir o nome de todos os artistas que interpretaram um determinado personagem em todas as temporadas.
7 - Sair

Escolha uma opcao:
4
Nenhuma Serie Cadastrada No Momento!
```

Figura 02 - Teste sem Séries Cadastradas

Na figura 03 será mostrado o restante dos testes, ou seja, opção 05, 06.

```
1 - Cadastrar Serie.
2 - Cadastrar Temporada.
3 - Imprimir series.
4 - Imprimir os dados de todas as temporadas de uma serie.
5 - Imprimir todos os personagens de uma determinada temporada.
6 - Imprimir o nome de todos os artistas que interpretaram um determinado personagem em todas as temporadas.
7 - Sair

Escolha uma opcao:
5
Nenhuma Serie Cadastrada No Momento!

1 - Cadastrar Serie.
2 - Cadastrar Temporada.
3 - Imprimir series.
4 - Imprimir os dados de todas as temporadas de uma serie.
5 - Imprimir todos os personagens de uma determinada temporada.
6 - Imprimir o nome de todos os artistas que interpretaram um determinado personagem em todas as temporadas.
7 - Sair

Escolha uma opcao:
6
Nenhuma Serie Cadastrada no Momento!
```

Figura 3 - Teste sem Séries Cadastradas

Assim como estabelecido para o caso de não haver séries cadastradas, o mesmo se aplica quando o usuário seleciona uma opção que envolve a impressão de dados de uma temporada que, no momento, não existe em uma série cadastrada. Nesse caso, será informado ao usuário que não há temporadas registradas para a série em questão. Isso é feito para evitar que o usuário procure por informações inexistentes e para prevenir a leitura de dados que não correspondem a nenhuma temporada. A Figura 3.1 ilustra essa funcionalidade. Vale lembrar que, embora uma série tenha sido cadastrada, sua árvore de temporadas está vazia no momento, ou seja, a raiz é NULL.

```
Escolha uma opcao:
1
Informe oCodigo da Serie:
10
Informe o Titulo da Serie:
sam e cat
Tempo de execucao: 700.00 nanossegundos
Serie Cadastrada Com Sucesso!

1 - Cadastrar Serie.
2 - Cadastrar Temporada.
3 - Imprimir series.
4 - Imprimir os dados de todas as temporadas de uma serie.
5 - Imprimir todos os personagens de uma determinada temporada.
6 - Imprimir o nome de todos os artistas que interpretaram um determinado personagem em todas as temporadas.
7 - Sair

Escolha uma opcao:
4
Informe o Código da Serie:
10
Nenhuma Temporada Cadastrada Para Essa Serie!
```

Figura 03.1

Ao examinar a figura 3 em detalhes, podemos confirmar que o programa está em conformidade com o que foi discutido anteriormente. É importante ressaltar que essa condição não se limita à opção 4, mas se estende a todas as opções que envolvem a manipulação de impressões e a busca de temporadas. O objetivo do terceiro teste foi registrar algumas séries para demonstrar a funcionalidade da opção 3, que é exibir todas as séries cadastradas no momento. Para isso, foram cadastradas 5 séries para testar essa opção. A figura 4 ilustra esse teste.

```
1 - Cadastrar Serie.
2 - Cadastrar Temporada.
3 - Imprimir series.
4 - Imprimir os dados de todas as temporadas de uma serie.
5 - Imprimir todos os personagens de uma determinada temporada.
6 - Imprimir o nome de todos os artistas que interpretaram um determinado personagem em todas as temporadas.
7 - Sair

Escolha uma opcao:
3
Codigo da Serie: 6
Titulo da Serie: Cobra Kai
Numero de Temporadas: 0
Codigo da Serie: 18
Titulo da Serie: La Casa de Papel
Numero de Temporadas: 0
Codigo da Serie: 11
Titulo da Serie: Riverdale
Numero de Temporadas: 0
Codigo da Serie: 12
Titulo da Serie: The Witcher
Numero de Temporadas: 0
Codigo da Serie: 13
Titulo da Serie: Suits
Numero de Temporadas: 0
```

Figura 04 - Opção 03 do menu

Ao analisar o resultado do terceiro teste, podemos ver todas as séries cadastradas até o momento. Vale ressaltar que nenhuma temporada foi registrada para essas séries, razão pela qual o número de temporadas para todas elas é zero. Assim que o usuário registrar uma temporada para uma dessas séries, o número de temporadas será incrementado. No quarto teste, foram registradas duas temporadas para a série com o código 11 para demonstrar a funcionalidade da opção 2, que é o registro de temporadas, a opção 4, que se refere à impressão dos dados de todas as temporadas de uma série, e a opção 5, que mostra todos os personagens de uma determinada temporada. As figuras 5 e 6 mostram o registro dessas duas temporadas. A figura 7 apresenta as informações de todas as temporadas existentes na série de código 11, que no momento são as duas que foram registradas.

```
Escolha uma opcao:  
2  
Informe o Código da Serie:  
11  
Buscando Serie...  
  
Serie Encontrada Com Sucesso!  
Informe o Numero da Temporada:  
23  
Informe o Titulo da Temporada 23:  
Ano Novo  
Informe a Quantidade de Episodios:  
56  
Informe o Ano da Temporada:  
2022  
Digite as Informacoes do Participante 1:  
Informe o Nome do Artista:  
Sandra Marques  
Informe o Nome do Personagem:  
Filomena  
Informe a Descricao do Personagem:  
Filomena nao e feliz na serie, pois perdeu a familia  
Deseja Inserir Mais um Participante (1 - SIM 2 - NAO)  
1  
Digite as Informacoes do Participante 2:  
Informe o Nome do Artista:  
Joao Levitch  
Informe o Nome do Personagem:  
Treviel  
Informe a Descricao do Personagem:  
Um homem raivoso e apaixonado  
Deseja Inserir Mais um Participante (1 - SIM 2 - NAO)  
2  
Temporada Cadastrada Com Sucesso!
```

figura 05 - Opção 02 Menu

```
Escolha uma opcao:  
2  
Informe o Código da Serie:  
11  
Buscando Serie...  
  
Serie Encontrada Com Sucesso!  
Informe o Numero da Temporada:  
19  
Informe o Titulo da Temporada 19:  
amanhecer  
Informe a Quantidade de Episodios:  
13  
Informe o Ano da Temporada:  
2022  
Digite as Informacoes do Participante 1:  
Informe o Nome do Artista:  
Fernando Costa  
Informe o Nome do Personagem:  
Garibaldo  
Informe a Descricao do Personagem:  
Garibaldo é um brasileiro famoso e rico  
Deseja Inserir Mais um Participante (1 - SIM 2 - NAO)  
2  
Temporada Cadastrada Com Sucesso!
```

Figura 06 - Opção 02 Menu

```
Escolha uma opcao:  
4  
Informe o Codigo da Serie:  
11  
Numero da Temporada: 19  
Nome da Temporada: amanhecer  
Quantidade de Episodios: 13  
  
Participantes da 19 Temporada:  
Informacoes do 1 Participante:  
Nome do Artista: Fernando Costa  
Nome do Personagem: Garibaldo  
Descricao do Personagem: Garibaldo é um brasileiro famoso e rico  
  
Numero da Temporada: 23  
Nome da Temporada: Ano Novo  
Quantidade de Episodios: 56  
  
Participantes da 23 Temporada:  
Informacoes do 1 Participante:  
Nome do Artista: Joao Levitch  
Nome do Personagem: Treviel  
Descricao do Personagem: Um homem raivoso e apaixonado  
  
Informacoes do 2 Participante:  
Nome do Artista: Sandra Marques  
Nome do Personagem: Filomena  
Descricao do Personagem: Filomena não é feliz na serie, pois perdeu a familia  
  
Tempo de execucao: 6116800.00 nanossegundos
```

Figura 07 - Opção 04 Menu

Ao examinar o teste apresentado na figura 7, fica evidente que a recuperação das informações de todas as temporadas da série 11 foi bem-sucedida, sem omitir nenhuma informação relevante para o usuário.

```
Escolha uma opcao:  
5  
Informe o Codigo da Serie:  
11  
Informe o Codigo da Temporada:  
23  
Nome do Personagem: Treviel  
Nome do Personagem: Filomena
```

Figura 08 - Opção 05 Menu

Ao analisar o teste apresentado na figura 8, fica claro que o programa recuperou todos os personagens relacionados à temporada de número 23, que atualmente inclui apenas Treviel e Filomena. Também será destacada a opção 6 do menu, que se refere à exibição dos nomes de todos os artistas que interpretaram um determinado personagem em todas as temporadas. No momento, não temos nenhum artista que tenha participado em todas as temporadas, mas veremos isso em ação.

```

Escolha uma opcao:
6
Informe o Código da Serie:
11
Personagem Nao Encontrado Em Todas As Temporadas da Serie.

```

Figura 09 - Opção 06 Menu.

Ao examinar o teste realizado na figura 9, fica claro que não há nenhum personagem que tenha participado de todas as séries. No entanto, a figura 10 demonstra essa funcionalidade para uma série em que um artista participou de todas as temporadas existentes.

A Tabela 1 apresenta um caso de teste abrangente que envolve uma sequência de 30 números: 10000, 4000, 200, 12, 45, 789, 5000, 4322, 7654, 23, 63523, 5673, 987, 432, 9876, 43, 675, 90, 3421, 98, 321, 75, 21, 999, 46, 1, 16, 8, 24, 4686. Foram realizados 30 testes, com cada teste alterando a sequência para variar a ordem de inserção na árvore de série. A partir das sequências inseridas individualmente na árvore de série, foi registrado o tempo de inserção de cada uma em nanossegundos. A tabela soma o tempo em nanossegundos das trinta inserções de cada sequência. No final da coluna de soma do tempo, é apresentada a soma de todos os casos de teste e a média obtida a partir das inserções dessas sequências.

Sequência de 30 Números Embaralhadas Para a Árvore de Busca	Tempo de Cada Inserção em Nanossegundos
432, 16, 24, 987, 5673, 4686, 4322, 999, 4000, 90, 7654, 63523, 9876, 23, 5000, 43, 675, 321, 75, 12, 8, 46, 3421, 45, 21, 1, 10000, 200, 98, 789.	9000.00
90, 4686, 16, 675, 4000, 432, 75, 23, 21, 5673, 9876, 321, 7654, 98, 63523, 43, 789, 987, 999, 4322, 12, 3421, 200, 8, 45, 46, 5000, 10000, 1, 24.	35100.00
90, 8, 321, 789, 24, 4322, 200, 5000, 3421, 7654, 4000, 21, 45, 675, 75, 1, 432, 5673, 46, 43, 23, 987, 10000, 12, 16, 63523, 4686, 999, 98, 9876.	15800.00

3421, 5000, 98, 321, 7654, 987, 8, 999, 432, 4686, 12, 46, 24, 9876, 5673, 75, 675, 16, 4000, 789, 4322, 21, 45, 10000, 200, 63523, 23, 43, 1, 90.	15300.00
3421, 8, 75, 16, 43, 999, 98, 789, 4322, 4686, 63523, 4000, 21, 46, 200, 1, 45, 12, 10000, 9876, 7654, 5000, 5673, 987, 321, 90, 24, 23, 432, 675.	16700.00
3421, 10000, 432, 24, 987, 12, 5673, 43, 789, 46, 1, 7654, 9876, 4686, 4322, 75, 8, 999, 21, 4000, 98, 23, 90, 16, 321, 675, 5000, 200, 63523, 45.	6800.00
3421, 5673, 24, 4686, 675, 987, 10000, 9876, 75, 4322, 7654, 12, 63523, 46, 999, 5000, 432, 16, 4000, 1, 43, 8, 200, 23, 90, 21, 321, 98, 45, 789.	16800.00
3421, 999, 90, 675, 5673, 24, 4686, 63523, 9876, 10000, 200, 987, 4322, 321, 432, 12, 45, 98, 46, 4000, 21, 789, 23, 7654, 16, 75, 1, 8, 5000, 43.	15200.00
3421, 1, 23, 675, 21, 4000, 75, 46, 4686, 9876, 987, 432, 98, 43, 999, 5673, 10000, 16, 5000, 4322, 200, 12, 8, 45, 789, 63523, 321, 7654, 90, 24.	11200.00
98, 675, 5673, 987, 321, 75, 4322, 21, 999, 45, 8, 9876, 24, 4686, 90, 200, 63523, 4000, 7654, 46, 5000, 432, 12, 789, 43, 10000, 1, 23, 16, 3421.	17600.00
98, 1, 432, 63523, 789, 4000, 3421, 321, 4686, 16, 75, 675, 8, 90, 5000, 200, 999, 43, 12, 21, 7654, 9876, 10000, 23, 4322, 5673, 24, 46, 45,	17200.00

987.	
98, 3421, 4000, 90, 7654, 200, 4322, 432, 5673, 1, 12, 63523, 45, 75, 16, 999, 8, 987, 4686, 675, 9876, 10000, 21, 24, 23, 5000, 789, 321, 46, 43.	7400.00
46, 12, 5673, 3421, 4686, 1, 8, 321, 63523, 43, 23, 789, 90, 98, 200, 10000, 987, 21, 24, 999, 9876, 4322, 7654, 16, 675, 45, 4000, 75, 5000, 432.	15500.00
321, 4322, 21, 46, 5673, 675, 45, 987, 789, 4686, 98, 200, 90, 23, 7654, 12, 4000, 5000, 1, 24, 75, 432, 16, 8, 9876, 3421, 43, 10000, 999, 63523.	18800.00
999, 4322, 7654, 4686, 432, 675, 98, 987, 5673, 90, 45, 46, 21, 16, 4000, 789, 12, 3421, 9876, 43, 1, 63523, 321, 5000, 10000, 24, 200, 75, 8, 23.	18800.00
675, 321, 90, 21, 4686, 24, 1, 999, 5673, 432, 12, 8, 5000, 789, 7654, 10000, 4322, 987, 46, 200, 3421, 16, 43, 9876, 23, 98, 63523, 75, 45, 4000.	7600.00
789, 43, 21, 5000, 45, 987, 90, 12, 4322, 63523, 4000, 4686, 321, 7654, 3421, 10000, 8, 5673, 999, 24, 98, 675, 1, 16, 432, 23, 9876, 200, 75, 46.	18300.00
75, 1, 675, 321, 90, 999, 9876, 24, 21, 45, 63523, 987, 4322, 789, 3421, 23, 432, 5673, 4686, 7654, 46, 5000, 43, 98, 8, 200, 12, 10000, 16, 4000.	15500.00
5000, 24, 5673, 12, 4322, 3421, 63523, 43, 16, 4000, 7654, 432, 4686, 75, 98, 321, 45, 10000, 675, 9876, 999, 23, 789, 1, 200, 46, 8, 987, 90, 21.	15500.00

432, 3421, 5673, 675, 8, 12, 789, 90, 23, 321, 7654, 9876, 10000, 200, 75, 4000, 63523, 21, 24, 999, 98, 987, 1, 16, 4686, 45, 4322, 5000, 43, 46.	18500.00
75, 46, 90, 43, 45, 9876, 432, 7654, 8, 4686, 5000, 675, 999, 63523, 3421, 200, 98, 789, 4000, 5673, 21, 987, 1, 321, 24, 4322, 10000, 23, 16,	20500.00
46, 5000, 45, 3421, 321, 4322, 10000, 7654, 63523, 12, 24, 23, 200, 5673, 90, 8, 675, 75, 1, 4000, 43, 987, 21, 9876, 789, 16, 999, 4686, 98, 432.	7000.00
45, 75, 5000, 3421, 90, 98, 23, 4686, 24, 21, 321, 63523, 8, 675, 1, 46, 16, 4000, 12, 4322, 999, 10000, 43, 200, 789, 9876, 432, 5673, 987, 7654.	7400.00
21, 789, 675, 46, 8, 1, 7654, 43, 63523, 45, 200, 75, 4686, 321, 24, 5000, 98, 432, 987, 10000, 12, 16, 4322, 4000, 90, 9876, 3421, 5673, 23, 999.	20000.00
21, 7654, 43, 3421, 90, 4686, 12, 98, 321, 9876, 16, 46, 45, 789, 4000, 4322, 23, 5000, 200, 75, 8, 63523, 987, 432, 1, 24, 675, 10000, 5673, 999.	10000.00
12, 4686, 321, 46, 200, 999, 7654, 987, 21, 432, 10000, 1, 5000, 90, 24, 5673, 4322, 3421, 8, 16, 675, 789, 98, 75, 23, 4000, 9876, 45, 43, 63523.	8700.00
21, 4686, 8, 98, 1, 5000, 200, 45, 3421, 4322, 675, 789, 24, 432, 90, 43, 999, 10000, 321, 46, 5673, 75, 23, 12, 987, 7654, 9876, 4000, 63523, 16.	17900.0
45, 9876, 98, 675, 46, 321, 90, 43, 75, 10000, 432, 24, 23, 12, 4686, 63523, 21, 8, 7654, 5000, 4000, 3421, 789, 1,	18700.00

5673, 16, 999, 200, 987, 4322.	
999, 16, 23, 12, 98, 63523, 4686, 5673, 9876, 4000, 24, 5000, 3421, 4322, 200, 432, 90, 45, 43, 46, 10000, 21, 1, 75, 8, 7654, 789, 987, 321, 675.	17700.00
999, 675, 3421, 63523, 9876, 46, 24, 98, 23, 7654, 10000, 5673, 5000, 16, 75, 4322, 789, 1, 12, 8, 21, 4686, 4000, 45, 987, 43, 321, 432, 200, 90.	18700.00
	Soma: 459.20,00 Média: 15.306,7

Tab.1 – Casos de Testes de Inserções de 30 números na Árvore Binária de Busca de Série.

A Tabela 02 exibe as temporadas pesquisadas no nó dos 200 da Árvore de Série, utilizando o caso de testes de Busca. Além disso, o tempo de execução de cada busca é mostrado. No final da tabela, a soma dos tempos de busca e a média correspondente de todos os tempos de busca em nanossegundos são apresentados.

Temporadas Buscadas no nó 200 da Árvore Série	Tempo de Cada Busca em Nanossegundos
16	300,00
4322	2300,00
63523	300,00
43	400,00
789	600,00
12	200,00
3421	500,00
9876	1200,00
98	500,00

46	500,00
8	200,00
200	400,00
7654	800,00
24	300,00
5673	600,00
4686	700,00
45	300,00
987	600,00
1	200,00
1000	700,00
23	300,00
5000	800,00
675	300,00
432	400,00
999	600,00
321	2400,00
75	500,00
21	300,00
4000	700,00
90	400,00
	Soma:18300,00 Média: 610,00

Tab2. – Casos de Testes de Buscas de 30 Elementos Na Árvore de Temporadas.

Nesta tabela 03, apresentamos uma análise detalhada dos números embaralhados para uma árvore AVL. A tabela exibe a sequência de números utilizados no processo, juntamente com o tempo necessário para cada inserção. Cada número é inserido na árvore AVL em uma ordem específica, resultando em uma estrutura de árvore equilibrada. O tempo necessário para inserir cada número é medido em nanossegundos, proporcionando uma visão precisa do desempenho do algoritmo de inserção. Além disso, a tabela também fornece a média de tempo de inserção, que é calculada somando todos os tempos de inserção e dividindo pelo número total de inserções. Esta média oferece uma visão geral do tempo que o algoritmo leva para inserir um número na árvore AVL.

Sequência de 30 Números Embaralhadas Para a Árvore AVL	Soma do Tempo de cada Inserção em Nanossegundos
3421, 75, 1, 90, 16, 5000, 432, 10000, 4000, 21, 7654, 4686, 43, 46, 12, 9876, 5673, 200, 8, 63523, 23, 4322, 24, 321, 98, 987, 999, 45, 789, 675	97800.00
3421, 432, 200, 999, 4686, 1, 8, 987, 789, 21, 90, 4322, 12, 24, 7654, 5673, 98, 9876, 43, 16, 45, 46, 4000, 321, 10000, 675, 75, 63523, 5000, 23	14700.00
3421, 4686, 90, 321, 23, 98, 4322, 63523, 675, 43, 45, 987, 9876, 21, 789, 16, 200, 46, 8, 12, 4000, 999, 7654, 5673, 1, 432, 75, 10000, 5000, 24.	15200.00
3421, 675, 45, 90, 43, 432, 987, 23, 4686, 4322, 789, 63523, 46, 200, 24, 9876, 5673, 999, 10000, 16, 7654, 21, 321, 4000, 98, 75, 1, 5000, 8, 12.	14900.00
98, 63523, 5673, 7654, 21, 789, 4686,	69200.00

75, 4322, 999, 5000, 9876, 987, 8, 3421, 23, 432, 45, 321, 16, 1, 10000, 24, 46, 90, 4000, 12, 43, 200, 675.	
98, 200, 432, 9876, 4686, 43, 321, 75, 23, 999, 10000, 90, 46, 63523, 675, 8, 7654, 21, 987, 4000, 1, 5000, 5673, 3421, 24, 12, 45, 16, 789, 4322.	19200.00
98, 7654, 75, 12, 46, 45, 987, 23, 5673, 3421, 675, 90, 999, 21, 63523, 789, 200, 4322, 4000, 321, 5000, 8, 4686, 43, 1, 16, 10000, 432, 24, 9876.	22900.00
98, 432, 24, 3421, 10000, 7654, 200, 75, 4000, 987, 9876, 12, 4322, 46, 43, 23, 321, 5000, 1, 21, 675, 4686, 90, 789, 45, 999, 8, 63523, 16, 5673.	16900.00
98, 10000, 3421, 8, 675, 7654, 432, 43, 16, 4000, 987, 23, 321, 90, 9876, 1, 12, 4686, 200, 75, 45, 63523, 999, 789, 24, 4322, 46, 21, 5673, 5000.	22300.00
98, 5000, 43, 10000, 46, 3421, 63523, 321, 24, 45, 4000, 12, 4686, 5673, 999, 789, 200, 432, 90, 1, 8, 21, 675, 7654, 9876, 987, 23, 16, 75, 4322.	33900.00
98, 5673, 987, 432, 90, 4000, 23, 16, 321, 24, 8, 7654, 5000, 12, 21, 46, 999, 1, 200, 3421, 45, 10000, 4322, 789, 63523, 9876, 4686, 43, 675, 75.	29100.00
98, 4322, 9876, 4686, 5000, 7654, 675, 21, 4000, 90, 1, 75, 200, 43, 8, 321, 3421, 45, 432, 10000, 12, 789, 16, 5673, 63523, 987, 46, 999, 24, 23.	21600.00
321, 90, 98, 5673, 75, 63523, 5000, 675, 4322, 7654, 10000, 21, 4686, 987, 16, 999, 12, 789, 46, 43, 3421, 45, 9876, 4000, 8, 23, 432, 1, 24, 200.	27500.00
321, 675, 432, 24, 10000, 12, 4686, 8,	18000.00

5673, 21, 200, 46, 90, 16, 987, 9876, 98, 75, 1, 789, 4000, 3421, 23, 45, 7654, 5000, 4322, 999, 43, 63523.	
321, 200, 3421, 12, 1, 5000, 675, 10000, 432, 45, 5673, 23, 8, 4686, 987, 7654, 21, 43, 4000, 46, 90, 24, 4322, 63523, 9876, 75, 16, 999, 98, 789.	27400.00
321, 3421, 675, 21, 5000, 200, 432, 1, 43, 23, 4686, 4322, 4000, 45, 98, 999, 24, 9876, 90, 789, 63523, 7654, 75, 8, 12, 46, 16, 10000, 5673, 987.	20400.00
321, 3421, 675, 21, 5000, 200, 432, 1, 43, 23, 4686, 4322, 4000, 45, 98, 999, 24, 9876, 90, 789, 63523, 7654, 75, 8, 12, 46, 16, 10000, 5673, 987.	20400.00
321, 200, 98, 1, 987, 21, 43, 789, 4000, 5000, 9876, 4686, 4322, 7654, 8, 5673, 3421, 10000, 12, 75, 63523, 45, 16, 24, 23, 90, 432, 46, 999, 675.	21200.00
321, 8, 21, 9876, 16, 98, 75, 789, 5000, 1, 5673, 63523, 23, 7654, 43, 46, 4000, 45, 4686, 999, 432, 987, 90, 3421, 675, 24, 10000, 4322, 200, 12.	10800.00
75, 9876, 23, 5673, 5000, 432, 16, 3421, 987, 98, 4686, 7654, 90, 4000, 46, 8, 789, 1, 321, 63523, 10000, 999, 21, 200, 24, 43, 675, 12, 45, 4322.	104600.00
75, 5673, 45, 90, 46, 12, 98, 8, 432, 4000, 24, 987, 321, 675, 4322, 23, 10000, 9876, 43, 63523, 200, 5000, 3421, 7654, 4686, 789, 1, 999, 21, 16.	16100.00
75, 90, 24, 21, 1, 45, 999, 5673, 10000, 4686, 321, 9876, 12, 5000, 43, 8, 16, 3421, 4000, 789, 46, 98, 200, 63523, 7654, 4322, 432, 987, 675, 23.	6900.00
75, 3421, 4322, 675, 789, 200, 4000,	7500.00

46, 43, 321, 9876, 12, 16, 23, 432, 5673, 1, 21, 999, 5000, 987, 98, 8, 45, 63523, 7654, 4686, 90, 10000, 24	
75, 45, 999, 432, 5673, 16, 21, 200, 24, 12, 987, 675, 90, 23, 4686, 98, 4322, 10000, 789, 7654, 3421, 5000, 46, 63523, 8, 9876, 1, 321, 4000, 43.	14200.00
75, 321, 63523, 200, 90, 4000, 8, 10000, 98, 5000, 12, 45, 23, 46, 21, 9876, 789, 4322, 3421, 999, 16, 5673, 24, 7654, 4686, 43, 432, 1, 675, 987.	15000.00
75, 16, 90, 1, 987, 4686, 98, 63523, 10000, 21, 24, 3421, 4000, 45, 23, 5000, 43, 200, 12, 46, 675, 999, 8, 7654, 432, 5673, 321, 4322, 9876, 789.	10900.00
75, 12, 90, 4686, 63523, 321, 10000, 5673, 999, 98, 16, 3421, 7654, 24, 789, 432, 23, 21, 8, 200, 9876, 45, 4322, 43, 46, 675, 1, 987, 5000, 4000.	11900.00
75, 3421, 21, 789, 43, 999, 7654, 16, 1, 200, 12, 98, 45, 432, 9876, 5000, 63523, 90, 10000, 4686, 4000, 5673, 8, 987, 24, 4322, 23, 46, 321, 675.	9400.00
21, 98, 75, 675, 999, 432, 789, 24, 4000, 43, 3421, 321, 63523, 16, 987, 5000, 45, 4686, 10000, 46, 12, 90, 4322, 5673, 1, 200, 23, 8, 7654, 9876.	6100.00
	Soma: 733.200,00 Média: 24.440,00

Tab3 - Casos de Testes de Inserções de 30 números na Árvore Binária AVL de Série.

Os dados apresentados na Tabela 3 revelam uma média de cadastro de 24.440,00, valor significativamente superior à média encontrada para a árvore binária de busca. Isso se deve ao fato de que, na árvore AVL, além da inserção, ocorre também o balanceamento. Este processo envolve o cálculo da altura, do fator de balanceamento e a realização de rotações simples ou duplas.

Portanto, é natural que o tempo de inserção na árvore AVL seja maior do que na árvore binária de busca. Em um teste adicional, cadastramos 30 temporadas com números aleatórios na série de código 7654. Utilizamos a mesma sequência observada no cadastro da série, porém, embaralhada. Assim, nas buscas realizadas, aplicamos um novo embaralhamento à sequência utilizada para o cadastro. Mas porque fizemos isso? O objetivo era buscar os dados de maneira aleatória, a fim de avaliar a eficiência do tempo de busca coletado. A Tabela 4 apresenta esses resultados de forma mais detalhada.

Temporadas Buscadas no Nô 7654 da Árvore AVL	Tempo de Cada Busca em Nanossegundos
8	700.000
45	1100.00
9876	1700.00
789	1400.00
24	400.00
63523	1200.00
4000	500.00
3421	500.00
1	200.00
4686	1100.00
16	500.00
321	700.00
200	700.00
46	200.00
98	100.00

12	600.00
675	600.00
4322	600.00
5673	100.00
1000	600.00
43	400.00
432	600.00
75	400.00
987	400.00
90	800.00
23	300.00
5000	500.00
21	500.00
999	600.00
7654	400.00
	Soma: 19.200,00 Média: 640,00

Tab4. – Casos de Testes de Buscas de 30 Elementos Na Árvore de Temporadas

Conclusão

Este trabalho destacou uma implementação eficaz de estruturas de dados, como árvores binárias de busca e árvores AVL, na resolução de problemas complexos relacionados ao cadastramento e manipulação de árvores de séries e temporadas. A utilização dessas estruturas de dados não só proporcionou uma organização sistemática e acessível dos dados, mas também otimizou a execução de operações de inserção e busca, reduzindo significativamente a complexidade computacional dessas operações.

Os resultados obtidos através dos experimentos realizados foram bastante promissores. Foi possível desenvolver programas que abordam os problemas propostos de maneira eficiente, lidando com a criação, manipulação e análise de árvores de séries e temporadas de forma precisa e eficaz. Estes programas não só resolveram os problemas propostos, mas também demonstraram a aplicabilidade prática das estruturas de dados estudadas. Além disso, este trabalho permitiu um aprofundamento no entendimento das estruturas de dados utilizadas.

Através da implementação prática, foi possível explorar as nuances e os detalhes de como as árvores binárias de busca e as árvores AVL funcionam, bem como as vantagens e desvantagens de cada uma. Este conhecimento prático complementa a teoria estudada, proporcionando uma compreensão mais completa e integrada das estruturas de dados.

Em resumo, a resolução dos problemas propostos neste trabalho não só permitiu o desenvolvimento de habilidades de programação relacionadas ao cadastramento e manipulação de árvores de séries e temporadas, mas também proporcionou uma compreensão aprofundada e prática das estruturas de dados estudadas. Este trabalho, portanto, representa um passo significativo no estudo e na aplicação de estruturas de dados na resolução de problemas reais.

Apêndice

Problema 1:

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

```
typedef struct Participantes {
    char nomeArtista[100];
    char nomePerso[100];
    char descricaoPerso[250];
    struct Participantes *Prox;
} Participantes;
```

```
typedef struct DadosTemporadas {
    int NumdaTemp;
    char tituTemp[100];
    int quantEp;
    int ano;
} DadosTemporadas;
```

```
typedef struct Temporadas {
    struct DadosTemporadas Info;
    struct Temporadas *Esq;
    struct Temporadas *Dir;
    struct Participantes *Parti;
} Temporadas;
```

```
typedef struct Series {
    int cod;
    char titulo[100];
    struct Series *Esq;
    struct Series *Dir;
    struct Temporadas *Temp;
} Series;
```

```
int inserirArvbinSerie(Series **raiz, Series *no) {
    int inseriu = 1;
```

```

if (*raiz == NULL) {
    *raiz = no;
    (*raiz)->Esq = NULL;
    (*raiz)->Dir = NULL;

}
else if (no->cod < (*raiz)->cod) {
    inseriu = inserirArvbinSerie(&(*raiz)->Esq), no);
}
else if (no->cod > (*raiz)->cod) {
    inseriu = inserirArvbinSerie(&(*raiz)->Dir), no);
}
else {
    inseriu = 0;
}

return inseriu;
}

```

```

void inserirArvbinTemp(Temporadas **raiz, Temporadas *no){

if(*raiz == NULL){
    *raiz = no;
    (*raiz)->Esq = NULL;
    (*raiz)->Dir = NULL;

}
else if(no->Info.NumdaTemp < (*raiz)->Info.NumdaTemp){
    inserirArvbinTemp(&(*raiz)->Esq), no);
}
else{
    inserirArvbinTemp(&(*raiz)->Dir), no);
}
}

```

```

void inserirParticipantes(Participantes **lista) {
    Participantes *novoParti = (Participantes*)malloc(sizeof(Participantes));

    printf("Informe o Nome do Artista:\n");
    fflush(stdin);
    scanf("%[^\\n]", novoParti->nomeArtista);
    printf("Informe o Nome do Personagem:\n");
    fflush(stdin);
    scanf("%[^\\n]", novoParti->nomePerso);
    printf("Informe a Descricao do Personagem:\n");
    fflush(stdin);
    scanf("%[^\\n]", novoParti->descricaoPerso);

    novoParti->Prox = NULL;

    if (*lista == NULL) {
        *lista = novoParti;
    }
    else {
        Participantes *aux = *lista;
        Participantes *ant = NULL;

        while (aux != NULL && strcmp(novoParti->nomeArtista,
aux->nomeArtista) > 0) {
            ant = aux;
            aux = aux->Prox;
        }

        if (ant == NULL) {
            novoParti->Prox = *lista;
            *lista = novoParti;
        }
        else {
            ant->Prox = novoParti;
            novoParti->Prox = aux;
        }
    }
}

```

```
}
```

```
}
```

```
int contaTemp(Temporadas *raiz){  
    int cont=0;  
    if(raiz != NULL){  
        cont += 1;  
        cont += contaTemp(raiz->Esq);  
        cont += contaTemp(raiz->Dir);  
    }  
  
    return cont;  
}
```

```
void imprimirSeries(Series *raiz){  
    int numTemp;  
    if(raiz != NULL){  
        imprimirSeries(raiz->Esq);  
        printf("Codigo da Serie: %d\n", raiz->cod);  
        printf("Titulo da Serie: %s\n", raiz->titulo);  
        numTemp = contaTemp(raiz->Temp);  
        printf("Numero de Temporadas: %d\n", numTemp);  
        imprimirSeries(raiz->Dir);  
    }  
}
```

```
void chamaParticipantes(Participantes *lista, int codTemp){  
    Participantes *aux;  
    int cont = 1;  
    aux = lista;  
    printf("Participantes da %d Temporada:\n", codTemp);  
    while(aux != NULL){
```

```

printf("Informacoes do %d Participante:\n", cont);
printf("Nome do Artista: %s\n", aux->nomeArtista);
printf("Nome do Personagem: %s\n", aux->nomePerso);
printf("Descricao do Personagem: %s\n", aux->descricaoPerso);
printf("\n");
aux = aux->Prox;
cont++;
}
}

```

```

void chamaTemporadas(Temporadas *raiz){

if(raiz!=NULL){
    printf("Numero da Temporada: %d\n", raiz->Info.NumdaTemp);
    printf("Nome da Temporada: %s\n", raiz->Info.tituTemp);
    printf("Quantidade de Episodios: %d\n", raiz->Info.quantEp);
    printf("\n");
    chamaParticipantes(raiz->Parti, raiz->Info.NumdaTemp);
    chamaTemporadas(raiz->Esq);
    chamaTemporadas(raiz->Dir);

}
}

```

```

void mostraDadosTemps(Series *raiz, int codigo){

if(raiz != NULL){
    if(raiz->cod == codigo){
        chamaTemporadas(raiz->Temp);

    }
    mostraDadosTemps(raiz->Esq, codigo);
    mostraDadosTemps(raiz->Dir, codigo);

}
}

```

```
}
```

```
void ImprimirPartiTemp(Participantes *lista){
```

```
    Participantes *aux = lista;
```

```
    do {
```

```
        printf("Nome do Personagem: %s\n", aux->nomePerso);
```

```
        aux = aux->Prox;
```

```
    }while(aux != NULL);
```

```
}
```

```
void ProcuraTemp(Temporadas *raiz, int numTemp){
```

```
    if(raiz != NULL){
```

```
        if(raiz->Info.NumdaTemp == numTemp){
```

```
            ImprimirPartiTemp(raiz->Parti);
```

```
}
```

```
    ProcuraTemp(raiz->Esq, numTemp);
```

```
    ProcuraTemp(raiz->Dir, numTemp);
```

```
}
```

```
}
```

```
void ImprimirPersonagemSerie(Series *raiz, int codSerie, int numTemp){
```

```
    if(raiz != NULL){
```

```
        if(raiz->cod == codSerie){
```

```
            ProcuraTemp(raiz->Temp, numTemp);
```

```
}
```

```
    ImprimirPersonagemSerie(raiz->Esq, codSerie, numTemp);
```

```
    ImprimirPersonagemSerie(raiz->Dir, codSerie, numTemp);
```

```
}
```

```
}
```

```

Series *BuscarSerie(Series **raiz, int cod) {
    Series *aux = NULL;
    if (*raiz != NULL) {
        if ((*raiz)->cod == cod) {
            aux = *raiz;
        } else {
            aux = BuscarSerie(&(*raiz)->Esq), cod);
            if (aux == NULL) {
                aux = BuscarSerie(&(*raiz)->Dir), cod);
            }
        }
    }
    return aux;
}

```

```
void imprimirArtistasPorPersonagem(Series *raiz){
```

```

Temporadas *temporada = raiz->Temp;
char personagemEncontrado[100];

```

```
personagemEncontrado[0] = '\0';
```

```
int todasTemporadas = 1;
```

```

while (temporada != NULL){
    Participantes *participante = temporada->Parti;

    while (participante != NULL){
        if (personagemEncontrado[0] == '\0'){
            strcpy(personagemEncontrado, participante->nomePerso);
        }
        else if (strcmp(personagemEncontrado, participante->nomePerso) != 0){

```

```

        todasTemporadas = 0;
        break;
    }

    participante = participante->Prox;
}

if (todasTemporadas == 0){
    break;
}

temporada = temporada->Dir;
}

if (todasTemporadas == 1){
    temporada = raiz->Temp;
    printf("Artistas que interpretaram o personagem %s na serie %s em todas
as temporadas:\n", personagemEncontrado, raiz->titulo);

    while (temporada != NULL){
        Participantes *participante = temporada->Parti;

        while (participante != NULL){
            if (strcmp(personagemEncontrado, participante->nomePerso) == 0){
                printf("- %s\n", participante->nomeArtista);
            }
            participante = participante->Prox;
        }

        temporada = temporada->Dir;
    }
}
else{
    printf("Personagem Nao Encontrado Em Todas As Temporadas da
Serie.\n");
}

```

```

}

int ExisteTemporada(Temporadas *raiz, int codigo){
    int existe = 0;
    if(raiz != NULL){
        if(raiz->Info.NumdaTemp == codigo){
            existe = 1;
        }
        if(existe != 1){
            existe = ExisteTemporada(raiz->Esq, codigo);
        }
        if(existe != 1){
            existe = ExisteTemporada(raiz->Dir, codigo);
        }
    }
    return existe;
}

int main(){
    Series *raizSeries;
    raizSeries = NULL;

    int escolha;
    do{
        printf("\n");
        printf("1 - Cadastrar Serie.\n");
        printf("2 - Cadastrar Temporada.\n");
        printf("3 - Imprimir series.\n");
        printf("4 - Imprimir os dados de todas as temporadas de uma serie.\n");
        printf("5 - Imprimir todos os personagens de uma determinada
temporada.\n");
        printf("6 - Imprimir o nome de todos os artistas que interpretaram um
determinado personagem em todas as temporadas.\n");
        printf("7 - Sair\n");
        printf("\n");
        printf("Escolha uma opcao:\n");
    }

```

```
scanf("%d", &escolha);

switch (escolha)
{
case 1:{

    int recebeu;
    Series *novaSerie;
    novaSerie = (Series*)malloc(sizeof(Series));
    printf("Informe o Código da Serie:\n");
    scanf("%d", &(novaSerie->cod));
    printf("Informe o Título da Serie:\n");
    fflush(stdin);
    scanf("%[^\\n]", novaSerie->título);

    novaSerie->Temp = NULL;

    recebeu = inserirArvbinSerie(&raizSeries, novaSerie);

    if(recebeu == 1){
        printf("Série Cadastrada Com Sucesso!\n");
    }
    else{
        printf("O Código da Série Informado Ja Existe!\n");
    }

    break;
}

case 2:{

    if(raizSeries == NULL){
        printf("Nenhuma Série Cadastrada No Momento!\n");
    }
    else{
        int CodSerie, numTemp;
```

```

printf("Informe o Codigo da Serie:\n");
scanf("%d", &CodSerie);
printf("Buscando Serie...\n");
printf("\n");
Series *recebeSerie;
recebeSerie = BuscarSerie(&raizSeries, CodSerie);
if(recebeSerie != NULL){
    printf("Serie Encontrada Com Sucesso!\n");
    printf("Informe o Numero da Temporada:\n");
    scanf("%d", &numTemp);
    int sabeTemp;
    sabeTemp = ExisteTemporada(recebeSerie->Temp, numTemp);
    if(sabeTemp == 1){
        printf("O Numero da Temporada Informado Ja Existe!\n");
    }
    else{
        Participantes *participantes;
        participantes = NULL;
        Temporadas *novaTemp;
        novaTemp = (Temporadas*)malloc(sizeof(Temporadas));
        novaTemp->Info.NumdaTemp = numTemp;
        printf("Informe o Titulo da Temporada %d:\n", numTemp);
        fflush(stdin);
        scanf("%[^\\n]", novaTemp->Info.tituTemp);
        printf("Informe a Quantidade de Episodios:\n");
        scanf("%d", &(novaTemp->Info.quantEp));
        printf("Informe o Ano da Temporada:\n");
        scanf("%d", &(novaTemp->Info.ano));
        int inserirperso =0;
        int contParti = 1;
        do{
            printf("Digite as Informacoes do Participante %d:\n", contParti);
            inserirParticipantes(&participantes);
            printf("Deseja Inserir Mais um Participante (1 - SIM 2 - NAO)\n");
            scanf("%d", &inserirperso);
            contParti++;
        }while(inserirperso !=2);
    }
}

```

```

    novaTemp->Parti = participantes;
    inserirArvbinTemp(&recebeSerie->Temp, novaTemp);
    printf("Temporada Cadastrada Com Sucesso!\n");

}

else{
    printf("Serie Nao Encontrada!\n");
}
}

break;

}

case 3:{

if(raizSeries == NULL){
    printf("Nenhuma Serie Cadastrada No Momento!\n");
}
else{
    imprimirSeries(raizSeries);
}
break;
}

case 4:{

if(raizSeries == NULL){
    printf("Nenhuma Serie Cadastrada No Momento!\n");
}
else{

```

```

int codSerie;

printf("Informe o Código da Serie:\n");
scanf("%d", &codSerie);

Series *verifica;
verifica = BuscarSerie(&raizSeries, codSerie);

if(verifica == NULL){
    printf("Série Nao Encotrada!\n");
}
else if(verifica->Temp == NULL){
    printf("Nenhuma Temporada Cadastrada Para Essa Serie!\n");
}
else{

    mostraDadosTemps(raizSeries, codSerie);
}

}

break;
}

case 5:{

if(raizSeries == NULL){
    printf("Nenhuma Serie Cadastrada No Momento!\n");
}
else{
    int codSerie, codTemp;
    printf("Informe o Código da Serie:\n");
    scanf("%d", &codSerie);
    Series *verificaSerie;
    verificaSerie = BuscarSerie(&raizSeries, codSerie);

    if(verificaSerie == NULL){
        printf("Série Nao Encotrada!\n");
    }
    else if(verificaSerie->Temp == NULL){

}
}
}

```

```

        printf("Nenhuma Temporada Cadastrada Para Essa Serie!\n");
    }
    else{
        int tem;
        printf("Informe oCodigo da Temporada:\n");
        scanf("%d", &codTemp);
        tem = ExisteTemporada(verificaSerie->Temp, codTemp);
        if(tem == 0){
            printf("Temporada Nao Encotrada!\n");
        }
        else{
            ImprimirPersonagemSerie(raizSeries, codSerie, codTemp);
        }
    }
}
break;
}

case 6:{

int codSerie;
if(raizSeries == NULL){
    printf("Nenhuma Serie Cadastrada no Momento!\n");
}
else{
    printf("Informe o Código da Serie:\n");
    scanf("%d", &codSerie);
    Series *Procura;
    Procura = BuscarSerie(&raizSeries, codSerie);
    if(Procura == NULL){
        printf("Serie Nao Encotrada!\n");
    }
    else if(Procura->Temp == NULL){
        printf("Nenhuma Temporada Cadastrada Para Essa Serie!\n");
    }
    else{
        imprimirArtistasPorPersonagem(Procura);
    }
}
}
```

```

    }
    break;

}

default:
    break;
}
}while(escolha != 7);
printf("Programa Finalizado!\n");

free(raizSeries);
return 0;
}

```

Problema 3:

```

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

typedef struct Participantes{
    char nomeArtista[100];
    char nomePerso[100];
    char descricaoPerso[250];
    struct Participantes *Prox;
}Participantes;

```

```

typedef struct DadosTemporadas{
    int NumdaTemp;
    char tituTemp[100];
    int quantEp;
    int ano;
    int altura;
}

```

```
}DadosTemporadas;

typedef struct Temporadas{
    struct DadosTemporadas Info;
    struct Temporadas *Esq;
    struct Temporadas *Dir;
    struct Participantes *Parti;
```

```
}Temporadas;
```

```
typedef struct Series{
    int cod;
    char titulo[100];
    struct Series *Esq;
    struct Series *Dir;
    struct Temporadas *Temp;
    int altura;
}Series;
```

```
int AlturadaArvoredeSerie(Series *raiz)
{
    int altura = 0;
    if (raiz != NULL)
    {
        int altura_esquerda = AlturadaArvoredeSerie(raiz->Esq);
        int altura_direita = AlturadaArvoredeSerie(raiz->Dir);
        altura = (altura_esquerda > altura_direita ? altura_esquerda : altura_direita)
+ 1;
    }
    return altura;
}
```

```
void rotacaoDireitaSerie(Series **raiz)
{
    Series *aux;
    aux = (*raiz)->Esq;
    (*raiz)->Esq = aux->Dir;
```

```

aux->Dir = (*raiz);
(*raiz)->altura = AlturadaArvoredeSerie(*raiz);
aux->altura = AlturadaArvoredeSerie(aux);
*raiz = aux;
}

```

```

void rotacaoEsquerdaSerie(Series **raiz)
{
    Series *aux;
    aux = (*raiz)->Dir;
    (*raiz)->Dir = aux->Esq;
    aux->Esq = (*raiz);
    (*raiz)->altura = AlturadaArvoredeSerie(*raiz);
    aux->altura = AlturadaArvoredeSerie(aux);
    *raiz = aux;
}

```

```

int inserirArvAVLSerie(Series **raiz, Series *no)
{
    int inseriu = 1;

    if (*raiz == NULL)
    {
        *raiz = no;
        (*raiz)->Esq = NULL;
        (*raiz)->Dir = NULL;
        (*raiz)->altura = 1;
    }
    else if (no->cod < (*raiz)->cod)
    {
        inseriu = inserirArvAVLSerie(&((*raiz)->Esq), no);
    }
    else if (no->cod > (*raiz)->cod)
    {
        inseriu = inserirArvAVLSerie(&((*raiz)->Dir), no);
    }
    else

```

```

{
    inseriu = 0;
}

if (inseriu){
    (*raiz)->altura = 1 + max(AlturadaArvoredeSerie((*raiz)->Esq),
AlturadaArvoredeSerie((*raiz)->Dir));

    int FB = AlturadaArvoredeSerie((*raiz)->Esq) -
AlturadaArvoredeSerie((*raiz)->Dir);

    if (FB > 1 && no->cod < (*raiz)->Esq->cod){
        rotacaoDireitaSerie(raiz);
    }
    else if (FB < -1 && no->cod > (*raiz)->Dir->cod){
        rotacaoEsquerdaSerie(raiz);
    }
    else if (FB > 1 && no->cod > (*raiz)->Esq->cod){
        rotacaoEsquerdaSerie(&((*raiz)->Esq));
        rotacaoDireitaSerie(raiz);
    }
    else if (FB < -1 && no->cod < (*raiz)->Dir->cod)
    {
        rotacaoDireitaSerie(&((*raiz)->Dir));
        rotacaoEsquerdaSerie(raiz);
    }
}

return inseriu;
}

int AlturadaArvoredeTemp(Temporadas *raiz){
    int altura = 0;
    if (raiz != NULL)
    {
        int altura_esquerda = AlturadaArvoredeTemp(raiz->Esq);
        int altura_direita = AlturadaArvoredeTemp(raiz->Dir);
    }
}

```

```

    altura = (altura_esquerda > altura_direita ? altura_esquerda : altura_direita)
+ 1;
}
return altura;
}

```

```

void rotacaoDireitaTemp(Temporadas **raiz){
    Temporadas *aux;
    aux = (*raiz)->Esq;
    (*raiz)->Esq = aux->Dir;
    aux->Dir = (*raiz);
    (*raiz)->Info.altura = AlturadaArvoredeTemp(*raiz);
    aux->Info.altura = AlturadaArvoredeTemp(aux);
    *raiz = aux;
}

```

```

void rotacaoEsquerdaTemp(Temporadas **raiz){
    Temporadas *aux;
    aux = (*raiz)->Dir;
    (*raiz)->Dir = aux->Esq;
    aux->Esq = (*raiz);
    (*raiz)->Info.altura = AlturadaArvoredeTemp(*raiz);
    aux->Info.altura = AlturadaArvoredeTemp(aux);
    *raiz = aux;
}

```

```

void inserirArvAVLTemp(Temporadas **raiz, Temporadas *no)
{

```

```

if (*raiz == NULL)
{
    *raiz = no;
    (*raiz)->Esq = NULL;
    (*raiz)->Dir = NULL;
}

```

```

(*raiz)->Info.altura= 1;
}
else if (no->Info.NumdaTemp < (*raiz)->Info.NumdaTemp){
    inserirArvAVLTemp(&((*raiz)->Esq), no);
}
else if (no->Info.NumdaTemp > (*raiz)->Info.NumdaTemp){
    inserirArvAVLTemp(&((*raiz)->Dir), no);
}

(*raiz)->Info.altura = 1 + max(AlturadaArvoredeTemp((*raiz)->Esq),
AlturadaArvoredeTemp((*raiz)->Dir));

int FB = AlturadaArvoredeTemp((*raiz)->Esq) -
AlturadaArvoredeTemp((*raiz)->Dir);

if (FB > 1 && no->Info.NumdaTemp < (*raiz)->Esq->Info.NumdaTemp){
    rotacaoDireitaTemp(raiz);
}
else if (FB < -1 && no->Info.NumdaTemp >
(*raiz)->Dir->Info.NumdaTemp)
{
    rotacaoEsquerdaTemp(raiz);
}
else if (FB > 1 && no->Info.NumdaTemp >
(*raiz)->Esq->Info.NumdaTemp)
{
    rotacaoEsquerdaTemp(&((*raiz)->Esq));
    rotacaoDireitaTemp(raiz);
}
else if (FB < -1 && no->Info.NumdaTemp <
(*raiz)->Dir->Info.NumdaTemp)
{
    rotacaoDireitaTemp(&((*raiz)->Dir));
    rotacaoEsquerdaTemp(raiz);
}

```

```
}
```

```
void inserirParticipantes(Participantes **lista) {
    Participantes *novoParti = (Participantes*)malloc(sizeof(Participantes));

    printf("Informe o Nome do Artista:\n");
    fflush(stdin);
    scanf("%[^\\n]", novoParti->nomeArtista);
    printf("Informe o Nome do Personagem:\n");
    fflush(stdin);
    scanf("%[^\\n]", novoParti->nomePerso);
    printf("Informe a Descricao do Personagem:\n");
    fflush(stdin);
    scanf("%[^\\n]", novoParti->descricaoPerso);

    novoParti->Prox = NULL;

    if (*lista == NULL) {
        *lista = novoParti;
    }
    else {
        Participantes *aux = *lista;
        Participantes *ant = NULL;

        while (aux != NULL && strcmp(novoParti->nomeArtista,
aux->nomeArtista) > 0) {
            ant = aux;
            aux = aux->Prox;
        }

        if (ant == NULL) {
            novoParti->Prox = *lista;
            *lista = novoParti;
        }
        else {
```

```

    ant->Prox = novoParti;
    novoParti->Prox = aux;
}
}
}

int contaTemp(Temporadas *raiz){
    int cont=0;
    if(raiz != NULL){
        cont += 1;
        cont += contaTemp(raiz->Esq);
        cont += contaTemp(raiz->Dir);
    }

    return cont;
}

```

```

void imprimirSeries(Series *raiz){
    int numTemp;
    if(raiz != NULL){
        printf("Codigo da Serie: %d\n", raiz->cod);
        printf("Titulo da Serie: %s\n", raiz->titulo);
        numTemp = contaTemp(raiz->Temp);
        printf("Numero de Temporadas: %d\n", numTemp);
        imprimirSeries(raiz->Esq);
        imprimirSeries(raiz->Dir);
    }
}

```

```

void chamaParticipantes(Participantes *lista, int codTemp){
    Participantes *aux;
    int cont = 1;
    aux = lista;
    printf("Participantes da %d Temporada:\n", codTemp);
    while(aux != NULL){

```

```

printf("Informacoes do %d Participante:\n", cont);
printf("Nome do Artista: %s\n", aux->nomeArtista);
printf("Nome do Personagem: %s\n", aux->nomePerso);
printf("Descricao do Personagem: %s\n", aux->descricaoPerso);
printf("\n");
aux = aux->Prox;
cont++;
}
}

```

```

void chamaTemporadas(Temporadas *raiz){

if(raiz!=NULL){
    printf("Numero da Temporada: %d\n", raiz->Info.NumdaTemp);
    printf("Nome da Temporada: %s\n", raiz->Info.tituTemp);
    printf("Quantidade de Episodios: %d\n", raiz->Info.quantEp);
    printf("\n");
    chamaParticipantes(raiz->Parti, raiz->Info.NumdaTemp);
    chamaTemporadas(raiz->Esq);
    chamaTemporadas(raiz->Dir);

}
}

```

```

void mostraDadosTemps(Series *raiz, int codigo){
    if(raiz != NULL){
        if(raiz->cod == codigo){
            chamaTemporadas(raiz->Temp);

        }
        mostraDadosTemps(raiz->Esq, codigo);
        mostraDadosTemps(raiz->Dir, codigo);

    }
}

```

```
void ImprimirPartiTemp(Participantes *lista){  
  
    Participantes *aux = lista;  
    do{  
        printf("Nome do Personagem: %s\n", aux->nomePerso);  
        aux = aux->Prox;  
    }while(aux != NULL);  
}
```

```
void ProcuraTemp(Temporadas *raiz, int numTemp){  
    if(raiz != NULL){  
        if(raiz->Info.NumdaTemp == numTemp){  
            ImprimirPartiTemp(raiz->Parti);  
  
        }  
  
        ProcuraTemp(raiz->Esq, numTemp);  
        ProcuraTemp(raiz->Dir, numTemp);  
    }  
}
```

```
void ImprimirPersonagemSerie(Series *raiz, int codSerie, int numTemp){  
    if(raiz != NULL){  
        if(raiz->cod == codSerie){  
            ProcuraTemp(raiz->Temp, numTemp);  
        }  
  
        ImprimirPersonagemSerie(raiz->Esq, codSerie, numTemp);  
        ImprimirPersonagemSerie(raiz->Dir, codSerie, numTemp);  
    }  
}
```

```
Series *BuscarSerie(Series **raiz, int cod) {  
    Series *aux = NULL;  
    if (*raiz != NULL) {
```

```

if ((*raiz)->cod == cod) {
    aux = *raiz;
} else {
    aux = BuscarSerie(&(*raiz)->Esq), cod);
    if (aux == NULL) {
        aux = BuscarSerie(&(*raiz)->Dir), cod);
    }
}
return aux;
}

```

```
void imprimirArtistasPorPersonagem(Series *raiz){
```

```

Temporadas *temporada = raiz->Temp;
char personagemEncontrado[100];

```

```
personagemEncontrado[0] = '\0';
```

```
int todasTemporadas = 1;
```

```

while (temporada != NULL){
    Participantes *participante = temporada->Parti;

    while (participante != NULL){
        if (personagemEncontrado[0] == '\0'){
            strcpy(personagemEncontrado, participante->nomePerso);
        }
        else if (strcmp(personagemEncontrado, participante->nomePerso) != 0){

            todasTemporadas = 0;
            break;
        }
    }
}
```

```

        participante = participante->Prox;
    }

    if (todasTemporadas == 0){
        break;
    }

    temporada = temporada->Dir;
}

if (todasTemporadas == 1){
    temporada = raiz->Temp;
    printf("Artistas que interpretaram o personagem %s na serie %s em todas
as temporadas:\n", personagemEncontrado, raiz->titulo);

    while (temporada != NULL){
        Participantes *participante = temporada->Parti;

        while (participante != NULL){
            if (strcmp(personagemEncontrado, participante->nomePerso) == 0){
                printf("- %s\n", participante->nomeArtista);
            }
            participante = participante->Prox;
        }

        temporada = temporada->Dir;
    }
}
else{
    printf("Personagem Nao Encontrado Em Todas As Temporadas da
Serie.\n");
}
}

```

```
int ExisteTemporada(Temporadas *raiz, int codigo){  
    int existe = 0;  
    if(raiz != NULL){  
        if(raiz->Info.NumdaTemp == codigo){  
            existe = 1;  
        }  
        if(existe != 1){  
            existe = ExisteTemporada(raiz->Esq, codigo);  
        }  
        if(existe != 1){  
            existe = ExisteTemporada(raiz->Dir, codigo);  
        }  
    }  
    return existe;  
}
```

```
int main(){  
    Series *raizSeries = NULL;  
  
    int escolha;  
    do{  
        printf("\n");  
        printf("1 - Cadastrar Serie.\n");  
        printf("2 - Cadastrar Temporada.\n");  
        printf("3 - Imprimir series.\n");  
        printf("4 - Imprimir os dados de todas as temporadas de uma serie.\n");  
        printf("5 - Imprimir todos os personagens de uma determinada  
temporada.\n");  
        printf("6 - Imprimir o nome de todos os artistas que interpretaram um  
determinado personagem em todas as temporadas.\n");  
        printf("7 - Sair\n");  
        printf("\n");  
        printf("Escolha uma opcao:\n");  
        scanf("%d", &escolha);
```

```

switch (escolha)
{
case 1:{

    int recebeu;
    Series *novaSerie;
    novaSerie = (Series*)malloc(sizeof(Series));
    printf("Informe o Código da Serie:\n");
    scanf("%d", &(novaSerie->cod));
    printf("Informe o Título da Serie:\n");
    fflush(stdin);
    scanf("%[^\\n]", novaSerie->título);

    novaSerie->Temp = NULL;

    recebeu = inserirArvAVLSerie(&raizSeries, novaSerie);

    if(recebeu == 1){
        printf("Série Cadastrada Com Sucesso!\n");
    }
    else{
        printf("O Código da Série Informado Ja Existe!\n");
    }

    break;
}

case 2:{

    if(raizSeries == NULL){
        printf("Nenhuma Série Cadastrada No Momento!\n");
    }
    else{
        int CodSerie, numTemp;

```

```

printf("Informe o Codigo da Serie:\n");
scanf("%d", &CodSerie);
printf("Buscando Serie...\n");
printf("\n");
Series *recebeSerie;
recebeSerie = BuscarSerie(&raizSeries, CodSerie);
if(recebeSerie != NULL){
    printf("Serie Encontrada Com Sucesso!\n");
    printf("Informe o Numero da Temporada:\n");
    scanf("%d", &numTemp);
    int sabeTemp;
    sabeTemp = ExisteTemporada(recebeSerie->Temp, numTemp);
    if(sabeTemp == 1){
        printf("O Numero da Temporada Informado Ja Existe!\n");
    }
    else{
        Participantes *participantes;
        participantes = NULL;
        Temporadas *novaTemp;
        novaTemp = (Temporadas*)malloc(sizeof(Temporadas));
        novaTemp->Info.NumdaTemp = numTemp;
        printf("Informe o Titulo da Temporada %d:\n", numTemp);
        fflush(stdin);
        scanf("%[^\\n]", novaTemp->Info.tituTemp);
        printf("Informe a Quantidade de Episodios:\n");
        scanf("%d", &(novaTemp->Info.quantEp));
        printf("Informe o Ano da Temporada:\n");
        scanf("%d", &(novaTemp->Info.ano));
        int inserirperso =0;
        int contParti = 1;
        do{
            printf("Digite as Informacoes do Participante %d:\n", contParti);
            inserirParticipantes(&participantes);
            printf("Deseja Inserir Mais um Participante (1 - SIM 2 - NAO)\n");
            scanf("%d", &inserirperso);
            contParti++;
        }while(inserirperso !=2);
    }
}

```

```

    novaTemp->Parti = participantes;
    inserirArvAVLtemp(&recebeSerie->Temp, novaTemp);
    printf("Temporada Cadastrada Com Sucesso!\n");

}

else{
    printf("Serie Nao Encontrada!\n");
}
}

break;
}

case 3:{

if(raizSeries == NULL){
    printf("Nenhuma Serie Cadastrada No Momento!\n");
}
else{
    imprimirSeries(raizSeries);
}
break;
}

case 4:{

if(raizSeries == NULL){
    printf("Nenhuma Serie Cadastrada No Momento!\n");
}
else{
    int codSerie;
    printf("Informe oCodigo da Serie:\n");
    scanf("%d", &codSerie);

    Series *verifica;
    verifica = BuscarSerie(&raizSeries, codSerie);
}
}

```

```

if(verifica == NULL){
    printf("Serie Nao Encotrada!\n");
}
else if(verifica->Temp == NULL){
    printf("Nenhuma Temporada Cadastrada Para Essa Serie!\n");
}
else{

    mostraDadosTemps(raizSeries, codSerie);
}
}
break;
}
case 5:{

if(raizSeries == NULL){
    printf("Nenhuma Serie Cadastrada No Momento!\n");
}
else{
    int codSerie, codTemp;
    printf("Informe o Código da Serie:\n");
    scanf("%d", &codSerie);
    Series *verificaSerie;
    verificaSerie = BuscarSerie(&raizSeries, codSerie);

    if(verificaSerie == NULL){
        printf("Serie Nao Encotrada!\n");
    }
    else if(verificaSerie->Temp == NULL){
        printf("Nenhuma Temporada Cadastrada Para Essa Serie!\n");
    }
    else{
        int tem;
        printf("Informe o Código da Temporada:\n");
        scanf("%d", &codTemp);
        tem = ExisteTemporada(verificaSerie->Temp, codTemp);
        if(tem == 0){
            printf("Temporada Nao Encotrada!\n");
        }
    }
}
}

```

```
        }
    else{
        ImprimirPersonagemSerie(raizSeries, codSerie, codTemp);
    }
}
break;
}
```

case 6:{

```
if(raizSeries == NULL){
    printf("Nenhuma Serie Cadastrada no Momento!\n");
}
else{
    int codSerie;
    printf("Informe oCodigo da Serie:\n");
    scanf("%d", &codSerie);
    Series *Procura;
    Procura = BuscarSerie(&raizSeries, codSerie);
    if(Procura == NULL){
        printf("Serie Nao Encotrada!\n");
    }
    else if(Procura->Temp == NULL){
        printf("Nenhuma Temporada Cadastrada Para Essa Serie!\n");
    }
    else{
        imprimirArtistasPorPersonagem(Procura);
    }
}
break;

}

default:
    break;
```

```
}

}while(escolha != 7);
printf("Programa Finalizado!\n");

free(raizSeries);
return 0;

}
```