Estudo Comparativo de Estruturas de Dados Não Lineares: Árvores Rubro Negras, Árvores 23 e Árvores 45

Luis Eduardo Silva Brito[1]

[1]CSHNB – Universidade Federal do Piauí (UFPI)

Professora: Juliana Oliveira de Carvalho
Disciplina: Estrutura de Dados II

Resumo. *Este projeto tem como objetivo desenvolver um sistema de acordo com os critérios estabelecidos na disciplina de Estrutura de dados II. Nesse sentido, irão ser implementadas estruturas de dados não lineares, conhecidas como "Árvores". Desse modo, serão implementadas as seguintes árvores: árvore rubro negra, árvore 23 e árvore 45. Em relação a este contexto, serão realizados testes de desempenho com intuito de avaliar as estruturas não lineares em diferentes cenários.*

1. Introdução

No contexto de estruturas de dados, existem várias alternativas que vão além das estruturas lineares. As estruturas de dados lineares, embora sejam fundamentais em certos cenários, podem não ser a solução ideal em outros. Um dos grandes problemas das estruturas lineares é a quantidade de dados que precisam ser processados. Neste sentido, existem alternativas que se destacam, por exemplo as árvores balanceadas como a rubro-negra a árvore 2-3 e a árvore 4-5, na qual oferecem vantagens significativas quando o assunto é desempenho e manutenção da ordenação dos dados.

No trabalho, serão exploradas três estruturas de dados conhecidas como "Árvores". Dentre elas estão a árvore rubro negra, a árvore 23 e a árvore 45. Cada um das estruturas mencionadas desenpenham um papel de suma importância na eficiência de dados em memória. Neste sentido, o estudo irá realizar uma análise detalhada a respeito de suas características, e respectivas implementações nos diferentes cenários propostos. Com o intuito de entender as limitações e vantagens de cada uma das árvores.

O presente trabalho, foi desenvolvido em linguagem C e foi estruturado em três pastas principais, com intuito de ajudar na compreensão e organização dos arquivos. A Figura 1, apresenta visualmente como estão estruturados os arquivos do projeto para facilitar a compreensão. Além disso, este projeto está organizado em seções claras, onde irão facilitar o entendimento de todo o processo de desenvolvimento. As seções estão organizadas em seções específicas, Resultados e Conclusão.

2. Seções Específicas

Esta seção tem como objetivo apresentar de maneira mais intuitiva cada uma das questões foram feitas, desde as estruturas que foram utilizadas até a interação com o usuário. Os testes de desempenho foram realizados na funcionalidade busca da árvore de cursos. Esses testes foram realizados em um computador com as especificações descritas na Tabela 1.

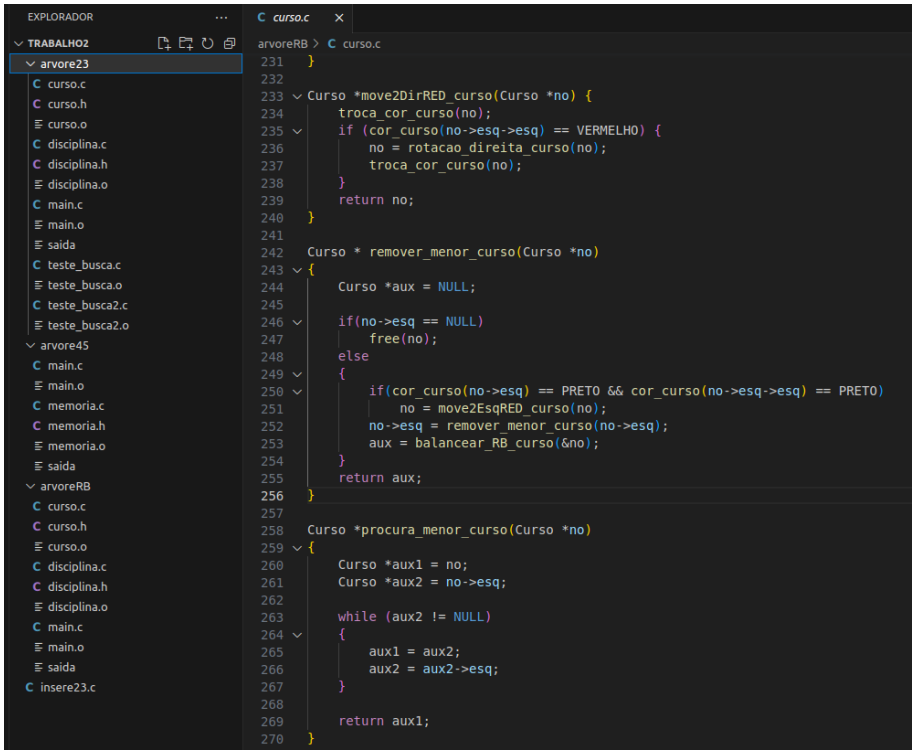

Figura 1. Organização dos arquivos do trabalho.

Tabela 1. Especificações do Sistema

Componente	**Detalhes**
Memória RAM	8 GB
Processador	Intel Core i5-1135G7
Sistema Operacional	Ubuntu

2.1. Gerenciamento Acadêmico com Árvore Vermelha-Preta

A questão proposta no trabalho, envolve o desenvolvimento de um sistema em linguagem C para gerenciar cursos e disciplinas. Nesse sentido, o sistema deve armazenar as informações de cada uma das estruturas mencionadas na proposta, ou seja, de cursos e disciplinas. Além disso, o sistema deve fornecer ao usuário opções de cadastro, consulta, e remoções de elementos e informações. As informações referentes aos cursos e disciplias foram estruturadas em formato de árvore vermelha-preta.

No projeto, foram utilizadas 2 estruturas de dados para gerenciar os cursos e disciplinas. As estruturas do sistemas estão organizadas dessa forma:

Árvore de Cursos: Esta árvore armazena os diferentes cursos. Cada curso é representado por o código do cuso, um nome único, pela quantidade de blocos, pelo número de semanas para cada disciplina, por uma árvore de disciplinas, e pela cor. A árvore é organizada pelo código do curso.

```
typedef struct Curso {
    int codigo;
    char nome[TAM_NOME_CURSO];
    int qtd_blocos;
```

```
    int semanas_por_disciplina;
    Disciplina *disciplinas;
    struct Curso *esq;
    struct Curso *dir;
    int cor;
} Curso;
```

Árvore de Disciplinas: Esta árvore armazena as diferentes Disciplinas de cada curso. Cada disciplina é representada por um código, um nome único, pelo bloco, por a carga horária, e por a cor. A árvore é organizada pelo código da disciplina.

```
typedef struct Disciplina {
    int codigo;
    char nome[TAM_NOME_DISCIPLINA];
    int bloco;
    int carga_horaria;
    struct Disciplina *esq;
    struct Disciplina *dir;
    int cor;
} Disciplina;
```

2.1.1. Funções Utilizadas na Árvore de Cursos

Esta estrutura desempenha um papel de extrema importância para o gerenciamento dos cursos. Com base nisso, serão apresentadas as funções que facilitam a organização e recuperação de informações dos cursos, onde serão discutidos os parâmetros de entrada, qual seu objetivo principal e o que a função retorna.

```
Curso *cria_no_curso(int codigo, char nome[], int qtd_blocos,
int semanas_por_disciplina);
```

Entrada: Esta função recebe como entrada o código do curso, o nome do curso, a quantidade de blocos e o número de semanas por disciplina.

Função: Cria um novo nó para um curso.

Retorno: Retorna um ponteiro para o novo nó do curso ou NULL em caso de falha.

```
int insere_no_RB_curso(Curso **raiz, Curso *novo);
```

Entrada: Esta função recebe como entrada um ponteiro para a raiz da árvore de cursos e um ponteiro para o novo nó do curso a ser inserido.

Função: Insere um novo nó na árvore vermelha-preta de cursos.

Retorno: Retorna 1 se o nó foi inserido com sucesso, -1 se o nó já está cadastrado, e 0 em caso de falha.

```
Curso *balancear_RB_curso(Curso **raiz);
```

Entrada: Esta função recebe como entrada um ponteiro para a raiz da árvore de cursos.

Função: Realiza o balanceamento da árvore vermelha-preta de cursos.

Retorno: Retorna um ponteiro para a raiz balanceada da árvore de cursos.

```
int cor_curso(Curso *raiz);
```

Entrada: Esta função recebe como entrada um ponteiro para o nó raiz de um curso.

Função: Verifica a cor do nó do curso.

Retorno: Retorna a cor do nó (VERMELHO ou PRETO). Se o nó for NULL, retorna PRETO.

```
void troca_cor_curso(Curso *raiz);
```

Entrada: Esta função recebe como entrada um ponteiro para o nó raiz de um curso.

Função: Troca a cor do nó do curso e de seus filhos.

Retorno: Não retorna nada.

```
Curso *rotacao_direita_curso(Curso *raiz);
```

Entrada: Esta função recebe como entrada um ponteiro para o nó raiz de um curso.

Função: Realiza uma rotação à direita na árvore de cursos.

Retorno: Retorna um ponteiro para a nova raiz após a rotação.

```
Curso *rotacao_esquerda_curso(Curso *raiz);
```

Entrada: Esta função recebe como entrada um ponteiro para o nó raiz de um curso.

Função: Realiza uma rotação à esquerda na árvore de cursos.

Retorno: Retorna um ponteiro para a nova raiz após a rotação.

```
void imprime_em_ordem(Curso *raiz);
```

Entrada: Esta função recebe como entrada um ponteiro para o nó raiz de um curso.

Função: Imprime os cursos em ordem crescente pelo código.

Retorno: Não retorna nada.

```
void imprime_curso_por_codigo(Curso *raiz, int codigo);
```

Entrada: Esta função recebe como entrada um ponteiro para o nó raiz de um curso e o código do curso a ser buscado.

Função: Imprime os dados de um curso dado o código do mesmo.

Retorno: Não retorna nada.

```
void imprime_cursos_por_qtd_blocos(Curso *raiz, int qtd_blocos);
```

Entrada: Esta função recebe como entrada um ponteiro para a raiz da árvore de cursos e a quantidade de blocos desejada.

Função: Imprime todos os cursos que possuem a quantidade de blocos especificada.

Retorno: Não retorna nada.

```
Curso *buscar_codigo(Curso *raiz, int codigo);
```

Entrada: Esta função recebe como entrada um ponteiro para a raiz da árvore de cursos e o código do curso a ser buscado.

Função: Busca um curso pelo código.

Retorno: Retorna um ponteiro para o nó do curso encontrado ou NULL se o curso não for encontrado.

```
void troca_cor_raiz_RB_curso(Curso *raiz);
```

Entrada: Esta função recebe como entrada um ponteiro para a raiz da árvore de cursos.

Função: Troca a cor da raiz da árvore para PRETO.

Retorno: Não retorna nada.

```
int remove_no_curso_ARVRB(Curso **raiz, int codigo_curso);
```

Entrada: Esta função recebe como entrada um ponteiro para a raiz da árvore de cursos e o código do curso a ser removido.

Função: Remove um curso da árvore vermelha-preta, se ele não tiver disciplinas cadastradas.

Retorno: Retorna 1 se o curso foi removido com sucesso, ou 0 em caso de falha.

```
Curso *remove_no_curso(Curso *raiz, int codigo_curso);
```

Entrada: Esta função recebe como entrada um ponteiro para a raiz da árvore de cursos e o código do curso a ser removido.

Função: Remove um nó da árvore de cursos.

Retorno: Retorna um ponteiro para a nova raiz da árvore após a remoção.

```
Curso *move2EsqRED_curso(Curso *no);
```

Entrada: Esta função recebe como entrada um ponteiro para um nó da árvore de cursos.

Função: Move um nó vermelho para a esquerda.

Retorno: Retorna um ponteiro para o nó após a movimentação.

```
Curso *move2DirRED_curso(Curso *no);
```

Entrada: Esta função recebe como entrada um ponteiro para um nó da árvore de cursos.

Função: Move um nó vermelho para a direita.

Retorno: Retorna um ponteiro para o nó após a movimentação.

```
Curso *remover_menor_curso(Curso *no);
```

Entrada: Esta função recebe como entrada um ponteiro para um nó da árvore de cursos.

Função: Remove o menor nó da árvore de cursos.

Retorno: Retorna um ponteiro para o nó após a remoção.

```
Curso *procura_menor_curso(Curso *no);
```

Entrada: Esta função recebe como entrada um ponteiro para um nó da árvore de cursos.

Função: Procura o menor nó da árvore de cursos.

Retorno: Retorna um ponteiro para o menor nó encontrado.

2.2. Gerenciamento Acadêmico com Árvore 2-3

A proposta do trabalho continua a mesma, só que agora ao invés de utilizar a árvore vermelha-preta, iremos utiliza a árvore 2-3. Esta árvore é uma árvore de busca balanceada onde cada um dos nós da árvore, exceto a raiz, tem dois ou três filhos.

As estruturas e as funções utlizadas para gerenciar os cursos e disciplinas são organizadas da mesma forma que a árvore vermelha-preta. Porém são organizadas em formato de árvore 2-3.

```
typedef struct Curso {
    int codigo;
    char nome[100];
    int blocos;
    int semanasPorDisciplina;
    Arv23Disciplina* disciplinas;
} Curso;

typedef struct Arv23Curso {
    Curso Info1;
    Curso Info2;
    struct Arv23Curso *Esq, *Cen, *Dir;
    int nInfos;
} Arv23Curso;
```

2.2.1. Funções Utilizadas na Árvore de Cursos

```
int ehFolhaCurso(Arv23Curso *No);
```

Entrada: Esta função recebe como entrada um ponteiro para um nó da árvore 2-3 de cursos.

Função: Verifica se o nó é uma folha.

Retorno: Retorna 1 se o nó é uma folha, ou 0 caso contrário.

```
Arv23Curso* criaNoCurso(Curso valor, Arv23Curso* FEsq,
Arv23Curso *FCen);
```

Entrada: Esta função recebe como entrada um curso e dois ponteiros para os filhos esquerdo e central.

Função: Cria um novo nó da árvore 2-3 de cursos com o curso fornecido e os filhos especificados.

Retorno: Retorna um ponteiro para o novo nó criado.

```
void adicionaInfoCurso(Arv23Curso** No, Curso valor,
Arv23Curso *Filho);
```

Entrada: Esta função recebe como entrada um ponteiro para um ponteiro de nó da árvore 2-3 de cursos, um curso e um ponteiro para um filho.

Função: Adiciona um curso a um nó existente que já contém um curso.

Retorno: Não retorna nada.

```
Arv23Curso* quebraNoCurso(Arv23Curso **No, Curso valor,
Curso *sobe, Arv23Curso *Filho);
```

Entrada: Esta função recebe como entrada um ponteiro para um ponteiro de nó da árvore 2-3 de cursos, um curso, um ponteiro para um curso a ser promovido, e um ponteiro para um filho.

Função: Divide um nó da árvore 2-3 de cursos que já contém dois cursos.

Retorno: Retorna um ponteiro para o novo nó criado após a divisão.

```
Arv23Curso* insereArv23Curso(Arv23Curso* Pai, Arv23Curso** R,
Curso Info, Curso* sobe);
```

Entrada: Esta função recebe como entrada um ponteiro para o pai de um nó, um ponteiro para um ponteiro da raiz da árvore 2-3 de cursos, um curso a ser inserido e um ponteiro para um curso a ser promovido.

Função: Insere um curso na árvore 2-3 de cursos.

Retorno: Retorna um ponteiro para o novo nó criado se houver divisão, ou NULL se não houver divisão.

```
Arv23Curso* buscaCurso(Arv23Curso *R, int codigo);
```

Entrada: Esta função recebe como entrada um ponteiro para a raiz da árvore 2-3 de cursos e o código do curso a ser buscado.

Função: Busca um curso pelo código na árvore 2-3 de cursos.

Retorno: Retorna um ponteiro para o nó do curso encontrado ou NULL se o curso não for encontrado.

```
void imprimeArvoreCurso(Arv23Curso *raiz);
```

Entrada: Esta função recebe como entrada um ponteiro para a raiz da árvore 2-3 de cursos.

Função: Imprime todos os cursos na árvore 2-3 de cursos em ordem crescente.

Retorno: Não retorna nada.

```
void imprimirDadosCurso(Arv23Curso *raizCurso, int codigo);
```

Entrada: Esta função recebe como entrada um ponteiro para a raiz da árvore 2-3 de cursos e o código do curso a ser impresso.

Função: Imprime os dados de um curso específico.

Retorno: Não retorna nada.

```
void imprimirCursosPorBlocos(Arv23Curso *raizCurso, int blocos);
```

Entrada: Esta função recebe como entrada um ponteiro para a raiz da árvore 2-3 de cursos e a quantidade de blocos.

Função: Imprime todos os cursos que possuem a quantidade de blocos especificada.

Retorno: Não retorna nada.

```
void imprimirDadosDisciplina(Arv23Curso *raizCurso,
int codigoCurso, int codigoDisciplina);
```

Entrada: Esta função recebe como entrada um ponteiro para a raiz da árvore 2-3 de cursos, o código do curso e o código da disciplina.

Função: Imprime os dados de uma disciplina específica de um curso específico.

Retorno: Não retorna nada.

```
void imprimirDisciplinasPorCargaHoraria(Arv23Curso* raizCurso,
int codigoCurso, int cargaHoraria);
```

Entrada: Esta função recebe como entrada um ponteiro para a raiz da árvore 2-3 de cursos, o código do curso e a carga horária.

Função: Imprime todas as disciplinas de um curso específico que possuem a carga horária especificada.

Retorno: Não retorna nada.

```
void menorInfoDireita(Arv23Curso *raiz, Arv23Curso **no,
Arv23Curso **paiAtual);
```

Entrada: Esta função recebe como entrada um ponteiro para a raiz da árvore 2-3 e dois ponteiros duplos para nós, um para armazenar o menor nó à direita e outro para armazenar o pai atual.

Função: Encontra o menor nó à direita na árvore 2-3 e armazena o resultado no ponteiro duplo fornecido.

Retorno: Não há retorno, o resultado é armazenado nos ponteiros duplos fornecidos.

```
void maiorInfoEsquerda(Arv23Curso *raiz, Arv23Curso **no,
Arv23Curso **paiAtual);
```

Entrada: Esta função recebe como entrada um ponteiro para a raiz da árvore 2-3 e dois ponteiros duplos para nós, um para armazenar o maior nó à esquerda e outro para armazenar o pai atual.

Função: Encontra o maior nó à esquerda na árvore 2-3 e armazena o resultado no ponteiro duplo fornecido.

Retorno: Não há retorno, o resultado é armazenado nos ponteiros duplos fornecidos.

```
void resetCurso(Curso *curso);
```

Entrada: Recebe como entrada um ponteiro para uma estrutura do tipo `Curso`.

Função: Reseta o campo `codigo` da estrutura `Curso`, definindo-o como 0. Esta função é útil para marcar um curso como ”livre”ou ”não utilizado”.

Retorno: Não há retorno explícito.

```
void removerCursoNoFolha(Arv23Curso **raiz, Arv23Curso *pai,
int codigo);
```

Entrada: Esta função recebe como entrada um ponteiro duplo para a raiz da árvore 2-3, um ponteiro para o pai do nó atual, e um inteiro representando o código do curso a ser removido.

Função: Remove um curso de um nó folha na árvore 2-3, ajustando os nós e redistribuindo ou fundindo elementos se necessário.

Retorno: Não há retorno, a árvore é modificada diretamente pelos ponteiros fornecidos.

```
void redistribuirOuFundirCurso(Arv23Curso **raiz, Arv23Curso
*pai, Arv23Curso *no, Arv23Curso *paiAtual);
```

Entrada: Esta função recebe como entrada um ponteiro duplo para a raiz da árvore 2-3, um ponteiro para o pai do nó atual, um ponteiro para o nó atual, e um ponteiro para o pai atual.

Função: Redistribui ou funde elementos nos nós da árvore 2-3 após a remoção de um curso.

Retorno: Não há retorno, a árvore é modificada diretamente pelos ponteiros fornecidos.

```
void removerCursoArvore23(Arv23Curso **raiz, int codigo);
```

Entrada: Esta função recebe como entrada um ponteiro para um ponteiro da raiz da árvore 2-3 de cursos e o código do curso a ser removido.

Função: Remove um curso da árvore 2-3 de cursos.

Retorno: Não retorna nada.

```
void removerDisciplinaCurso(Arv23Curso **raizCurso,
int codigoCurso, int codigoDisciplina);
```

Entrada: Recebe como entrada um ponteiro para a raiz da árvore 2-3 de cursos, o código do curso e o código da disciplina a ser removida.

Função: Remove uma disciplina específica de um curso na árvore 2-3 de cursos, mantendo a integridade da estrutura.

Retorno: Não retorna valor, apenas imprime mensagens de sucesso ou erro.

```
void liberaArvoreCurso(Arv23Curso *raiz);
```

Entrada: Recebe como entrada um ponteiro para a raiz da árvore 2-3 de cursos.

Função: Libera toda a memória alocada para os nós e disciplinas da árvore 2-3 de cursos, recursivamente.

Retorno: Não retorna valor.

2.2.2. Funções Utilizadas na Árvore de Disciplinas

Esta estrutura desempenha um papel de extrema importância para o gerenciamento das disciplinas de cada curso. Com base nisso, serão apresentadas as funções que facilitam a organização e recuperação de informações das disciplinas, onde serão discutidos os parâmetros de entrada, qual seu objetivo principal e o que a função retorna.

```
Arv23Disciplina* criaNoDisciplina(Disciplina valor,
Arv23Disciplina* FEsq, Arv23Disciplina* FCen);
```

Entrada: Esta função recebe como entrada o valor de uma disciplina, um ponteiro para o filho à esquerda e um ponteiro para o filho central.

Função: Cria um novo nó para a árvore 2-3 de disciplinas.

Retorno: Retorna um ponteiro para o novo nó criado.

```
void adicionaInfoDisciplina(Arv23Disciplina** No,
Disciplina valor, Arv23Disciplina* Filho);
```

Entrada: Esta função recebe como entrada um ponteiro para um ponteiro de nó da árvore 2-3 de disciplinas, um valor de disciplina a ser adicionado e um ponteiro para um filho.

Função: Adiciona uma nova informação a um nó da árvore 2-3 de disciplinas.

Retorno: Não há retorno explícito.

```
Arv23Disciplina* quebraNoDisciplina(Arv23Disciplina** No,
Disciplina valor, Disciplina* sobe, Arv23Disciplina* Filho);
```

Entrada: Esta função recebe como entrada um ponteiro para um ponteiro de nó da árvore 2-3 de disciplinas, um valor de disciplina a ser inserido, um ponteiro para uma disciplina a ser promovida e um ponteiro para um filho.

Função: Divide um nó da árvore 2-3 de disciplinas que já contém duas disciplinas.

Retorno: Retorna um ponteiro para o novo nó criado após a divisão.

```
int ehFolhaDisciplina(Arv23Disciplina* No);
```

Entrada: Esta função recebe como entrada um ponteiro para um nó da árvore 2-3 de disciplinas.

Função: Verifica se o nó é uma folha da árvore 2-3 de disciplinas.

Retorno: Retorna 1 se o nó for uma folha, caso contrário retorna 0.

```
Arv23Disciplina* insereArv23Disciplina(Arv23Disciplina* Pai,
Arv23Disciplina** R, Disciplina Info, Disciplina* sobe);
```

Entrada: Esta função recebe como entrada um ponteiro para o nó pai, um ponteiro para um ponteiro de nó raiz da árvore 2-3 de disciplinas, um valor de disciplina a ser inserido e um ponteiro para uma disciplina a ser promovida.

Função: Insere uma nova disciplina na árvore 2-3 de disciplinas.

Retorno: Retorna um ponteiro para um novo nó criado, se houver divisão.

```
Arv23Disciplina* buscaDisciplina(Arv23Disciplina* R,
int codigo);
```

Entrada: Esta função recebe como entrada um ponteiro para o nó raiz da árvore 2-3 de disciplinas e um código de disciplina a ser buscado.

Função: Busca uma disciplina na árvore 2-3 de disciplinas com base no código.

Retorno: Retorna um ponteiro para o nó que contém a disciplina buscada, ou NULL se não encontrada.

```
void imprimeArvoreDisciplina(Arv23Disciplina* raiz);
```

Entrada: Esta função recebe como entrada um ponteiro para o nó raiz da árvore 2-3 de disciplinas.

Função: Imprime todas as disciplinas da árvore 2-3 de disciplinas em ordem.

Retorno: Não há retorno explícito.

```
void imprimirDisciplinasPorBloco(Arv23Disciplina* raiz,
int bloco);
```

Entrada: Esta função recebe como entrada um ponteiro para o nó raiz da árvore 2-3 de disciplinas e um número de bloco.

Função: Imprime todas as disciplinas da árvore 2-3 de disciplinas que pertencem ao bloco especificado.

Retorno: Não há retorno explícito.

```
void imprimirDisciplinasPorCargaHorariaAux(Arv23Disciplina* raiz,
int cargaHoraria);
```

Entrada: Esta função recebe como entrada um ponteiro para o nó raiz da árvore 2-3 de disciplinas e uma carga horária.

Função: Imprime todas as disciplinas da árvore 2-3 de disciplinas que possuem a carga horária especificada.

Retorno: Não há retorno explícito.

```
void liberaArvoreDisciplina(Arv23Disciplina* raiz);
```

Entrada: Esta função recebe como entrada um ponteiro para o nó raiz da árvore 2-3 de disciplinas.

Função: Libera toda a memória alocada para a árvore 2-3 de disciplinas.

Retorno: Não há retorno explícito.

```
int removerDisciplina(Arv23Disciplina** raiz, int codigo);
```

Entrada: Esta função recebe como entrada um ponteiro para um ponteiro de nó raiz da árvore 2-3 de disciplinas e um código de disciplina a ser removida.

Função: Remove uma disciplina da árvore 2-3 de disciplinas com base no código.

Retorno: Retorna 1 se a disciplina foi removida com sucesso, caso contrário retorna 0.

```
void removerDisciplinaNoFolha(Arv23Disciplina **raiz,
Arv23Disciplina *pai, int codigo);
```

Entrada: Recebe como entrada a raiz da árvore `Arv23Disciplina` (um ponteiro para ponteiro), um ponteiro para o nó pai e o código da disciplina a ser removida.

Função: Remove uma disciplina de um nó folha na árvore 2-3. Se o nó possui dois valores, apenas um valor é removido e o nó se torna um nó com um único valor. Se o nó possui apenas um valor, o nó é removido e a árvore é ajustada através de redistribuição ou fusão de nós, se necessário.

Retorno: Não há retorno explícito.

—

```
void menorInfoDireitaDisciplina(Arv23Disciplina *raiz,
Arv23Disciplina **no, Arv23Disciplina **paiAtual);
```

Entrada: Recebe a raiz da subárvore (`Arv23Disciplina`), um ponteiro para armazenar o nó encontrado (`no`), e um ponteiro para armazenar o pai atual (`paiAtual`).

Função: Encontra o nó com a menor informação na subárvore direita, atualizando os ponteiros fornecidos.

Retorno: Não há retorno explícito.

—

```
void maiorInfoEsquerdaDisciplina(Arv23Disciplina *raiz,
Arv23Disciplina **no, Arv23Disciplina **paiAtual);
```

Entrada: Recebe a raiz da subárvore (`Arv23Disciplina`), um ponteiro para armazenar o nó encontrado (`no`), e um ponteiro para armazenar o pai atual (`paiAtual`).

Função: Encontra o nó com a maior informação na subárvore esquerda, atualizando os ponteiros fornecidos.

Retorno: Não há retorno explícito.

—

```
void resetDisciplina(Disciplina *disciplina);
```

Entrada: Recebe um ponteiro para uma estrutura do tipo `Disciplina`.

Função: Reseta todos os campos da estrutura `Disciplina`, definindo o código como 0, o nome como uma string vazia, o bloco como 0, e a carga horária como 0. Esta função é útil para marcar uma disciplina como ”livre”ou ”não utilizada”.

Retorno: Não há retorno explícito.

—

```
void redistribuirOuFundirDisciplina(Arv23Disciplina **pai,
Arv23Disciplina **noDeficiente, Arv23Disciplina **irmao, int posPai);
```

Entrada: Recebe como entrada um ponteiro para o nó pai, um ponteiro para o nó deficiente, um ponteiro para o nó irmão e a posição do pai.

Função: Redistribui ou funde os nós da árvore 2-3 quando ocorre uma remoção de um nó que resulta em um nó deficiente (um nó com menos de uma informação).

Retorno: Não há retorno explícito.

2.3. Gerenciamento de uma Memória com Árvore 4-5

A questão proposta no trabalho, envolve o desenvolvimento de um sistema em linguagem C para gerenciar a memória de um computador. Nesse sentido, o sistema deve armazenar as informações, com o objetivo de manter o controle dos blocos livres e ocupados da memória. A questão foi estruturada em formato de árvore 4-5.

No projeto, foram utilizadas 2 estruturas de dados para gerenciar a memória. As estruturas do sistemas estão organizadas dessa forma:

```
typedef struct BlocoMemoria {
    char status;
    int blocoInicial;
    int blocoFinal;
    void *enderecoInicial;
    void *enderecoFinal;
} BlocoMemoria;

typedef struct Arv45 {
    int nInfos;
    BlocoMemoria infos[4];
    struct Arv45 *Esq, *Centro1, *Centro2, *Centro3, *Dir;
} Arv45;
```

2.3.1. Funções Utilizadas na Árvore da Memória

Esta estrutura desempenha um papel de extrema importância para o gerenciamento da memória de um computador. Com base nisso, serão apresentadas as funções que facilitam a organização e recuperação de informações dos blocos livres e ocupados, onde serão discutidos os parâmetros de entrada, qual seu objetivo principal e o que a função retorna.

```
Arv45* criaNoBloco(BlocoMemoria bloco);
```

Entrada: Recebe como entrada um objeto ‘BlocoMemoria‘ para criar um novo nó na árvore 4-5.

Função: Aloca memória e cria um novo nó na árvore 4-5 com as informações do bloco fornecido.

Retorno: Retorna um ponteiro para o novo nó criado, ou NULL se a alocação falhar.

```
int ehFolha(Arv45 *no);
```

Entrada: Recebe como entrada um nó da árvore 4-5.

Função: Verifica se o nó fornecido é uma folha.

Retorno: Retorna 1 se o nó é uma folha, ou 0 caso contrário.

```
void adicionaInfoBloco(Arv45 *no, BlocoMemoria bloco,
Arv45 *filhoDir);
```

Entrada: Recebe como entrada um nó da árvore 4-5, um bloco de memória para adicionar e um ponteiro para um filho direito.

Função: Adiciona um novo bloco de memória ao nó da árvore 4-5, mantendo a ordenação.

Retorno: Não há retorno explícito.

```
Arv45* divideNoBloco(Arv45 *no, BlocoMemoria bloco,
BlocoMemoria *sobe, Arv45 *filhoDir);
```

Entrada: Recebe como entrada um nó da árvore 4-5, um bloco de memória para dividir, um ponteiro para um bloco a ser elevado e um ponteiro para um filho direito.

Função: Divide um nó da árvore 4-5 em dois nós, mantendo a ordenação e retornando o novo nó criado.

Retorno: Retorna um ponteiro para o novo nó criado durante a divisão.

```
Arv45* insereArv45(Arv45* pai, Arv45** raiz, BlocoMemoria bloco,
BlocoMemoria *sobe);
```

Entrada: Recebe como entrada um nó pai, um ponteiro para a raiz da árvore 4-5, um bloco de memória para inserir e um ponteiro para um bloco a ser elevado.

Função: Insere um bloco de memória na árvore 4-5, mantendo a estrutura balanceada.

Retorno: Retorna um ponteiro para um novo nó criado durante a operação de divisão, ou NULL se nenhum novo nó foi criado.

```
void imprimeNo(Arv45 *no);
```

Entrada: Recebe como entrada um nó da árvore 4-5.

Função: Imprime no console as informações contidas no nó da árvore 4-5.

Retorno: Não há retorno explícito.

```
void imprimeArvore(Arv45 *raiz);
```

Entrada: Recebe como entrada a raiz da árvore 4-5.

Função: Imprime no console a árvore 4-5 completa, percorrendo os nós em pré-ordem.

Retorno: Não há retorno explícito.

```
BlocoMemoria* buscaBloco(Arv45 *raiz, void *enderecoInicial);
```

Entrada: Recebe como entrada a raiz da árvore 4-5 e um ponteiro para um endereço inicial.

Função: Busca e retorna um ponteiro para o bloco de memória correspondente ao endereço inicial na árvore 4-5.

Retorno: Retorna um ponteiro para o bloco de memória encontrado, ou NULL se não encontrado.

```
void liberaBlocos(Arv45 *raiz, void *enderecoInicial,
int numBlocos);
```

Entrada: Recebe como entrada a raiz da árvore 4-5, um ponteiro para um endereço inicial e o número de blocos a serem liberados.

Função: Libera um número específico de blocos de memória, marcando-os como livres na árvore 4-5.

Retorno: Não há retorno explícito.

3. Resultados

Nesta seção serão apresentados os resultados obtidos a partir dos testes que foram realizados nas funções da árvore vermelha-preta e na árvore 2-3. Essas estruturas foram avaliadas em relação aos tempos de busca. O objetivo principal destes testes era determinaer a eficiência de cada estrutura.

3.1. Resultados dos testes na árvore vermelha-preta e na árvore 2-3

Os resultados obtidos na árvore vermelha-preta e na árvore 2-3 do tempo de execução da função de busca, em ambas as árvores, foi registrado na Tabela 2. Esta tabela ilustra o comportamento da função sob diferentes condições de busca de dados, no caso 30 dados diferentes. Além disso, foram realizados testes para cada uma das situações e realizada a média de tempo.

Tabela 2. Tempo médio de busca de dados

Tipo de Árvore	Quantidade de Dados	Tempo Médio (ms)
ARB	30	0,001200
A2-3	30	0,003167

Para ilustrar os resultados obtidos a partir dos testes de busca na árvore vermelha-preta e na árvore 2-3, foram criados gráficos que demonstram o tempo de execução da

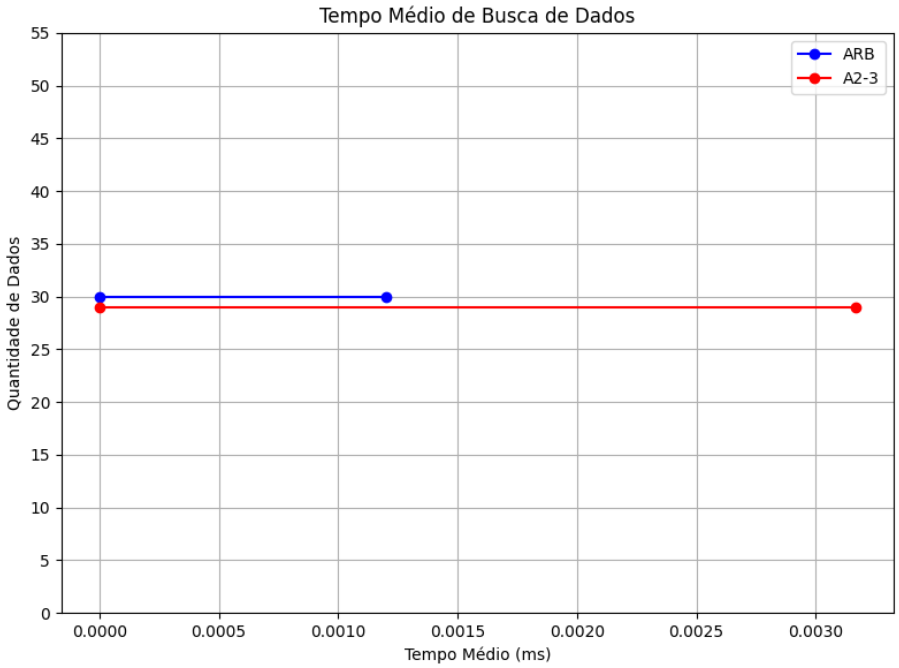

Figura 2. Tempo médio da função de busca nas árvores

função de busca em ambas as árvores. A Figura 2 representa os resultado da função de busca.

Com base nas análises dos gráficos e das tabela podemos concluir que em termos de tempo médio de busca para encontrar os elementos na árvore, podemos observar que a árvore vermelha-preta teve um tempo de busca inferior ao da árvore 2-3, devido a quantidade de dados que foram buscados.

4. Conclusão

Este trabalho destaca a importância das estruturas de dados não lineares. Ao realizarmos a comparação entre as árvores RB e 2-3 podemos observar diferenças entre elas. Na árvore RB, a busca pelos elementos especificados, vai depender de onde esse elemento vai estar na árvore, de acordo com a ordem inserida, da mesma forma é a 2-3. Realizando uma comparação com estes dados percebemos que a árvore 2-3 é mais lenta em comparação com a RB.

Analisando esses dados, observamos que a árvore RB demonstra maior eficiência na busca de dados específicos. Por outro lado, a árvore 2-3, embora mais lenta, pode oferecer desempenho superior em cenários específicos, dependendo das operações realizadas. Portanto, a escolha entre a árvore RB e a 2-3, vai depender das necessidades estabelecidas.