Universidade Federal do Piauí – Campus Senador Helvídio Nunes de Barros – CSHNB Curso de Sistemas de Informação Disciplina: Programação Funcional

Professora: Juliana Oliveira de Carvalho

Relatório de Estruturas de Dados II

Autor:

Bryan Victor da Costa Martins

Picos PI, janeiro de 2023

* **Resumo:**

Esse relatório tem como objetivo demostrar a pratica e implementação Estrutura de Arvores de busca binaria e arvores AVL para resolução de alguns problemas práticos propostos, utilizando a Linguagem de programação C.

* **Introdução:**

Neste relatório vamos analisar como foi resolvido alguns problemas, inicialmente foi proposto gerar 1000 valores aleatórios e inseri-los em uma árvore e analisar o seu desempenho ao inserir e fazer busca em 30 árvores diferentes, contudo foi gerado uma amostra de 20.000 elementos para melhor resolução dos dados.

Ler dados de um arquivo com palavras Inglês-Português e inserir numa arvore as palavras em português e em uma lista sua tradução em inglês. Para solução destes problemas foi utilizada a linguagem de programação C e foram empregados alguns conceitos importantes para resolução de tais problemas, dentre eles estão: ponteiros, árvores binárias e árvores AVL.

* **Seções Específicas**

**Informações técnicas:**

Para o desenvolvimento e testes deste projeto foi utilizado um notebook com um processador AMD Ryzen™ 7 5700U 4.3GHz, oito gigas de memória RAM ddr4, sistema operacional com a arquitetura de 64 bits, Windows 11 Pro.

Os códigos foram feitos na IDE e editor de texto Visual Studio Code e compilados pelo compilador gcc (MinGW.org GCC-6.3.0-1) 6.3.0.

**Problema 1 e 2:**

Resumo do problema: Gerar 1000, (Foram gerados 20.000) valores aleatórios e inseri-los em uma árvore, mostrar o nível da folha de maior profundidade e o nível da folha de menor profundidade e calcular a frequência da diferença entre elas, gerar 30 árvores diferentes e analisar o seu desempenho da inserção e da busca em cada árvore.

**Resumo da solução:**

Em ambas questões 1 e 2, seguem a mesma lógica para solução do problema diferem apenas na lógica de inserção das árvores na questão 1 teremos a árvore binária e na questão 2 teremos a árvore AVL, resumidamente para a solução deste problema teremos um laço de repetição onde irá rodar 30 vezes e a cada repetição será gerada uma árvore com 20.000 elementos onde cada nó da árvore recebe um número inteiros entre 1 e dobro da quantidade de elementos (20.000 \* 2).

Após cada árvore gerada é calculado e mostrado o tempo gasto para inserção dos valores na árvore, é calculado o tempo gasto para busca de uma sequência de elementos (2.000 buscas) fixo para todas as árvores geradas da questão 1 e 2, também para cada árvore é calculado e mostrado o nível da folha de maior profundidade e o nível da folha de menor profundidade e por fim é calculado e mostrado frequência da diferença entre os níveis.

**Funções principais:**

**Inicializar ():** Essa função recebe um ponteiro com o endereço de uma arvore e inicializar sua raiz com NULL;

**Inserir (Questão 1):** Essa função recebe um ponteiro com o endereço da arvore e um elemento inteiro para ser adicionado, caso a raiz seja NULL será adicionado, quando a raiz não for nula, vai ocorrer uma verificar para saber se o elemento a ser adicionado é maior fazendo uma chamada recursiva da função inserirDireita ou menor do que o elemento fazendo uma chamada recursiva da função inserirEsquerda do Nó atual até que o Nó seguinte a chamada seja nulo.

**InserirDireita (Questão 1):** Essa função vai receber um Nó e um elemento para ser adicionado a arvore, caso o elemento seja maior do que o elemento do Nó atual vai ocorrer uma chamada recursiva para o Nó a direita do Nó atual.

**InserirEsquerda (Questão 1):** Essa função vai receber um Nó e um elemento para ser adicionado a arvore, caso o elemento seja menor do que o elemento do Nó atual vai ocorrer uma chamada recursiva para o Nó a esquerda do Nó atual.

**Busca (Questão 1 e 2):** Essa função vai receber um o Nó raiz e um elemento para ser buscado na arvore, caso o elemento de busca seja menor do que o elemento do Nó, vai ocorrer uma chamada recursiva para o nó da esquerda, caso seja maior para o nó da direita, até que o elemento seja igual e retornando esse elemento ou que encontre raiz nula retornando assim 0.

**GerarNumeros (Questão 1 e 2):** Essa função vai receber um ponteiro de um vetor vazio e vai preencher suas 20.000 posições com números aleatórios entre 1 e (2\*20.000).

**EmbaralharVetor (Questão 1 e 2):** Essa função vai receber um ponteiro de um vetor já preenchido e vai trocar os elementos de posições de forma aleatório.

**ProfundidadeMax (Questão 1 e 2):** Essa função vai receber um ponteiro para o nó raiz e vai devolver a maior profundidade do último nó folha (esquerda e direita igual a NULL) dessa arvore.

**ProfundidadeMin (Questão 1 e 2):** Essa função vai receber um ponteiro para o nó raiz e vai devolver a menor profundidade do primeiro nó folha (esquerda e direita igual a NULL) dessa arvore.

**VerificarRepetições (Questão 1 e 2):** Essa função vai receber um ponteiro para o vetor com os valores das distâncias entre a profundidade máxima e mínima das 30 arvores e informar suas repetições.

**LiberarArvore (Questão 1 e 2):** Essa função vai receber um ponteiro para o nó raiz e vai liberar a memória alocada em todos os elementos a esquerda do nó raiz e em seguida para todos da direita e por último da própria raiz.

**Main (Questão 1 e 2):** Funcionalidade do tempo foi implementada no main, pois é contabilizada variáveis do tipo clock para armazenar tempo inicial e final de inserção e busca.

**Observação (Questão 1 e 2):** Ambas as funções ProfundidadeMax e ProfundidadeMin retornam os valores para o main onde é feito a diferença e armazenada em um vetor.

**CriarNo (Questão 2):** Função vai receber o elemento para ser adicionado a arvore e alocar memoria em um novo nó para esse elemento e retornar o ponteiro desse nó.

**Inserir (Questão 2):** Função vai receber um ponteiro para o nó raiz e o elemento para ser adicionado, caso essa raiz seja vazia, a função criarNo será chamada, caso a raiz não seja vazia será verificado (elemento menor que o elemento do nó) caso seja valido a função inserir fará uma chamava recursiva passando o nó a esquerda, caso (elemento maior que o elemento do nó) será chamada à função inserir só que dessa vez passando o nó a direita, até que seja encontrado um nó nulo para adicionar o elemento.

**Balancear (Questão 2):** Função vai receber um ponteiro para a raiz (nó), e vai verificar o fator de balanceamento do nó (chamando a função fatorBalanceamento), caso esse valor seja menor que –1 ou maior que 1 a nossa arvore (ou sub-arvore) precisa ser balanceada, depois vai ser verificado o fator de tanto do nó a esquerda quanto à direita do nosso nó raiz para saber qual rotação precisará ser executada simples (Esquerda, Direita), duplas (Esquerda – Direita ou Direita – Esquerda).

**RotacaoEsq (Questão 2):** Função vai receber um ponteiro para o nó raiz (nó), e vai sentar as novas alturas (chamado a função maior) e verificando a altura tanto do nó quanto do nó que vai se tornar o novo nó raiz da arvore (ou sub-arvore) executando uma rotação a esquerda e retornando o ponteiro para a nova raiz.

**RotacaoDir (Questão 2):** Função vai receber um ponteiro para o nó raiz (nó), e vai sentar as novas alturas (chamado a função maior) e verificando a altura tanto do nó quanto do nó que vai se tornar o novo nó raiz da arvore (ou sub-arvore) executando uma rotação a direita e retornando o ponteiro para a nova raiz.

**RotacaoEsqDir (Questão 2):** Função vai receber um ponteiro para o nó raiz (nó), e vai chamar a função RotacaoEsq e em seguida a função RotacaoDir e retornar um ponteiro para a nova raiz.

**RotacaoDirEsq (Questão 2):** Função vai receber um ponteiro para o nó raiz (nó), e vai chamar a função RotacaoDir e em seguida a função RotacaoEsqe retornar um ponteiro para a nova raiz.

**Maior (Questão 2):** Função vai receber dois inteiros A e B (alturas de nó), e com um operador ternário vai devolver qual foi maior.

**AlturaNo (Questão 2):** Função vai receber um ponteiro para um nó e vai devolver a altura desse nó, caso o nó seja nulo o retorno e –1.

**FatorBalanceamento (Questão 2):** Função vai receber um ponteiro para um nó e caso esse nó não seja nulo vai ser retorno à subtração entre a altura do nó a esquerda e nó a direita.

**Problema 3 e 4:**

Faça um programa que converta um conjunto de vocabulários armazenados em um arquivo Inglês em um conjunto de vocabulários Português-Inglês. Além disso, faça as seguintes funcionalidades: Imprimir palavras de uma unidade, imprimir palavras em inglês equivalentes a uma em português, remover palavra em português e suas equivalentes em inglês.

**Resumo da solução:**

Em ambas questões 3 e 4, seguem a mesma lógica para solução do problema diferem apenas na lógica de inserção e remoção das árvore na questão 3 teremos a árvore binária e na questão 4 teremos a árvore AVL, Para solução deste problema inicialmente ler os dados do arquivo já definido no código, o arquivo será lido linha por linha e cada vez que for encontrado uma string com o início com % representando uma unidade é lido todas as palavras que vem nas linhas a seguir e são inseridas em uma árvore, onde a palavra em português é armazenada no Nó e as palavras equivalentes em inglês são armazenados em uma lista dentro do Nó, e cada unidade lida é criada uma nova árvore .

**Funções principais:**

**Calcula\_tempo (Questão 3 e 4):** recebe o tempo inicial e o final da execução e calcula e retorna o tempo gasto em milissegundos.

**Open\_file (Questão 3 e 4):** vai carregar as palavras do arquivo.

**Trata\_string (Questão 3 e 4):** recebe uma string de palavras recebidas do arquivo e depois armazena no vocabulário tratando os caracteres especiais.

**Aloca\_lista\_ingles (Questão 3 e 4):** função para alocar uma variável do tipo typedef struct lista para receber as palavras em ingles.

**Aloca\_arvore( Questão 3 e 4):** Essa função irá receber como entrada um ponteiro para uma estrutura do tipo arvorebin, que armazenará as informações de um Nó da árvore é uma string, esse ponteiro irá apontar para um espaço alocado na memória para um Nó, e esse ponteiro será devolvido pela função por referência.

**Insere\_plv\_ingles(Questão 3 e 4):** Recebe um ponteiro para uma estrutura do tipo Lista e uma string e coloca um Nó para lista utilizando a função aloca\_lista\_ingles() e depois insere o Nó na lista utilizando a função insere\_lista().

**Insere\_arvore (Questão 3):** Na questão 3, nossa árvore será binária, então a inserção funcionará da seguinte forma, ela irá receber a Raiz da árvore , um Nó com a palavra em portugues e uma string com a palavra em inglês equivalente, então percorremos a árvore até que a Raiz seja NULL, no primeiro caso como a Raiz será NULL bastará fazer com que a Raiz receba o No e iremos chamar a função insere\_plv\_ingles() passando a lista do Nó e a palavra em inglês, nos demais casos quando o valor que o Nó que armazena for maior que o valor da raiz iremos fazer uma chamada recursiva passando a Raiz.dir para ser a nova raiz , quando valor for menor fazemos uma chamada recursiva passando a Raiz.esq quando a palavra da raiz for igual a do No, ou seja, já tiver sido inserido então faremos apenas a inserção da palavra em inglês na lista do No utilizando a função insere\_plv\_ingles() e se

a palavra já não tiver sido inserida iremos percorrer até que a Raiz seja Nula e faremos com que a Raiz receba o No e iremos inserir a palavra em inglês na lista do No utilizando unção insere\_plv\_ingles().

**Insere\_arvore (Questão 4):** Na questão 4, nossa árvore será AVL, então a inserção funcionará da seguinte forma, ela seguirá os mesmos passos da inserção da questão 3, porém teremos alguns passos a mais, a cada chamada recursiva ficará pendente calcular a altura do No pela função altura\_no() e verificar o balanceamento da árvore pela função verifica\_balanceamento(), então após a inserção do Nó o código irá voltar pelo caminho que foi percorrido para a inserção resolvendo as pendências.

**altura\_no(Questão 3 e 4):** Recebe um Nó e devolve o comprimento do caminho do Nó recebido até folha a maior profundidade do Nó.

**verifica\_balanceamento(Questão 3 e 4):** Recebe um Nó da árvore, esta função irá calcular o fator de balanceamento da árvore utilizando a função fator\_balanceamento(), após isso é verificado se é necessário balancear a árvore, se o fator de balanceamento for 2 temos que balancear a árvore para esquerda chamando a função balanceia\_esquerda() e se o fator de balanceamento for -2 temos que balancear a árvore para direita chamando a função balanceia\_direita().

**carregar\_dados(Questão 3 e 4)**: Essa função recebe um vetor de unidades que armazena árvores e um ponteiro para um arquivo essa função, ler os dados dos arquivos e a cada unidade encontrada inserem as palavras da unidade em uma árvore e isso será feito da seguinte forma, é feito um laço de repetição onde a cada laço é lido uma linha do arquivo, e quando for encontrado um string com início com % então nas linhas a seguir do arquivo iremos ler todas as palavra em portugues e sua equivalente inglês e inserir na árvore da unidade até que seja encontrada no arquivo uma nova string com início com % , então é incrementado um contador de unidades e é lido os dados e armazenados em outra árvore da unidade e assim por diante até o fim do arquivo.

**imprimir\_unidade(Questão 3 e 4):** Recebe vetor de árvores, e o nome de uma unidade, essa função irá buscar a unidade escolhida e irá imprimir todas as palavras em portugues e sua lista de equivalentes em inglês da árvore.

**imprimir\_equivalentes\_ingles(Questão 3 e 4):** Recebe vetor de árvores e uma palavra em portugues e busca em todas as árvores das unidades as palavras em inglês equivalentes a palavra em portugues e imprime.

**Remove (Questão 3):** Recebe a Raiz da árvore e a string com a palavra em portugues que será removida, busca palavra da forma parecido como já foi explicado na inserção e quando é encontrado o Nó que iremos chamar a função remover\_lista() passando a lista do No para remover todas palavras da lista do No, após isso é removido o Nó da árvore.

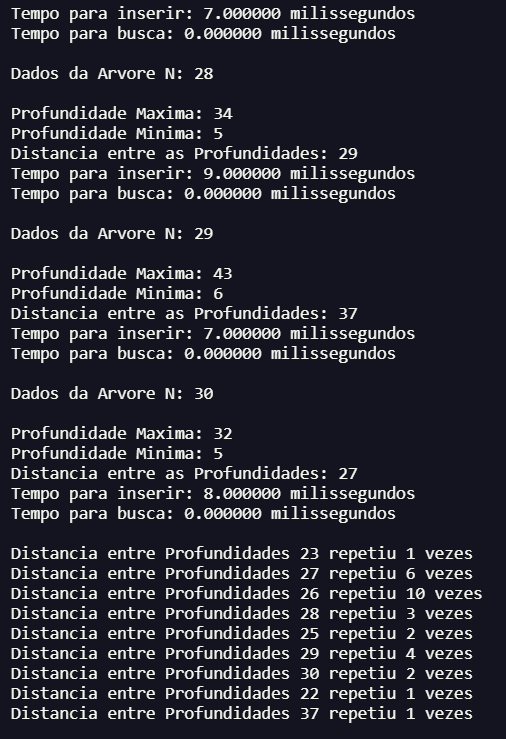
**remove (Questão 4):** Na questão 4, nossa árvore será AVL, então a remoção funcionará da seguinte forma, ela seguirá os mesmos passos da remoção da questão 3, porém teremos alguns passos a mais, a cada chamada recursiva ficará pendente calcular a altura do No pela função altura\_no() e verificar o balanceamento da árvore pela função verifica\_balanceamento(), então após a remoção do Nó o código irá voltar pelo caminho que foi percorrido para a inserção resolvendo as pendências.

**remover\_palavra(Questão 3 e 4):** Recebe vetor de árvores e o nome de uma unidade e uma palavra e busca a unidade escolhida e remove a palavra da árvore utilizando a **função remove ():** e sua lista.

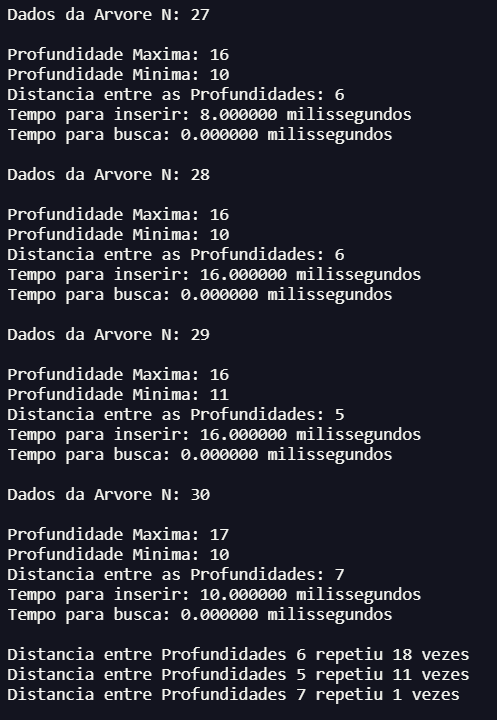
* **Resultados da Execução do Programa:**

Neste tópico iremos analisar dado algumas entradas como os códigos e suas saídas:

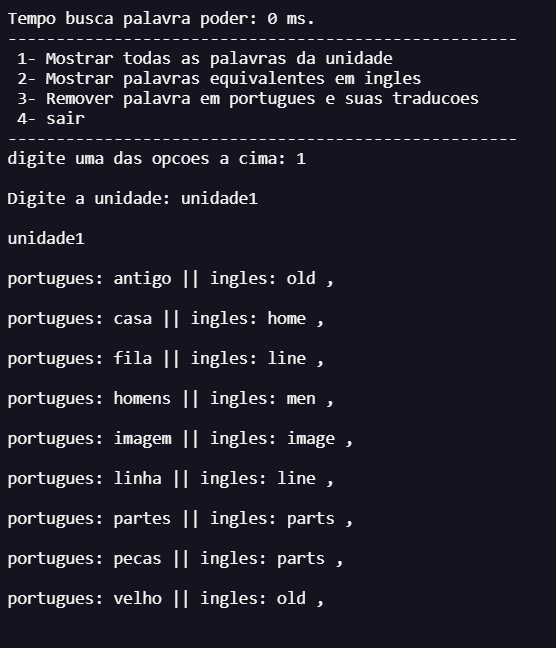
**Questão 1**



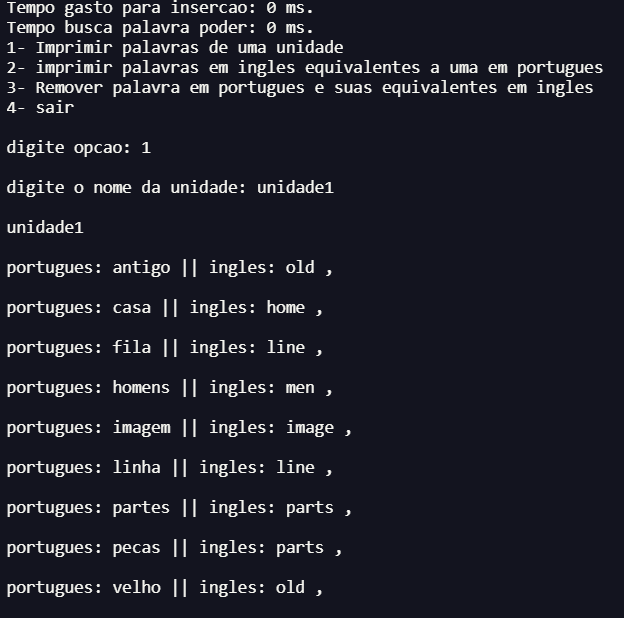
**Questão 2**



**Questão 3**



**Questão 4**



* **Analise de Desempenho**

**Arvore Binaria Questão 1**

Tempo de inserção máximo: 1.00000 milissegundo

Tempo de inserção mínimo: 9.00000 milissegundos

Tempo de Busca máximo: 8.00000 milissegundos

Tempo de Busca Mínimo: 0.00010 milissegundos

**Arvore AVL Questão 2**

Tempo de inserção máximo: 16.00000 milissegundos

Tempo de inserção mínimo: 7.00000 milissegundos

Tempo de Busca máximo: 8.00000 milissegundos

Tempo de Busca Mínimo: 0.06000 milissegundos

**questões 3 e 4**

Não foi possível retornar resultados de tempo referentes a essas questões, tanto na árvore binária quanto na árvore AVL.

* **Conclusão**

Por fim, podemos concluir que este trabalho teve como objetivo proporcionar para os estudantes um aprofundamento em conceitos importantes para a disciplina como manipulação de dados, processamento de dados, manipulação de ponteiros e estruturas de dados como árvores binárias e AVL.