



UNIVERSIDADE FEDERAL DO PIAUÍ – UFPI
CAMPUS SENADOR HELVÍDIO NUNES DE BARROS – PICOS
BACHARELADO EM SISTEMAS DE INFORMAÇÃO
PROFESSOR: Juliana Oliveira de Carvalho

Trabalho de Estruturas de dados 2

Autor:

David Marcos Santos Moura

Picos, Setembro de 2021

- **Resumo do projeto:**

Inicialmente vamos falar um pouco sobre Estruturas de dados, **Estrutura de dados** é o ramo da computação que estuda os diversos mecanismos de organização de dados para atender aos diferentes requisitos de processamento. Posto isso, este relatório tem como objetivo demonstrar e analisar como é utilizados na prática conceitos de manipulação de dados para solução de alguns problemas práticos.

- **Introdução:**

Neste relatório vamos analisar como foi resolvido alguns problemas, como: gerar 1000 valores aleatórios e inseri-los em uma árvore e analisar o seu desempenho para gerar 30 árvores diferentes, ler dados de um arquivo com palavras Inglês-Português e inserir numa árvore 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:**

Neste tópico iremos abordar de maneira resumida como foi solucionado os problemas propostos:

Problema 1 e 2:

Resumo do problema:

Gerar 1000 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 árvore 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 1000 números onde cada nó da árvore recebe um número inteiros entre 0 e 20.000, após a cada árvore gerada é calculado e mostrado o tempo gasto para de inserção dos valores na árvore e também é calculado o tempo gasto

para busca de um número fixo para todas as árvores geradas da questão 1 e 2, também a 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:

aloca(): Essa função irá receber como entrada um ponteiro para uma estrutura do tipo `arvorebin`, que armazenará as informações de um Nó da árvore é um valor inteiro, esse ponteiro irá apontar para um espaço alocado na memória para um No, e esse ponteiro será devolvido pela função por referência.

questão 1 Insere(): Na questão 1, nossa árvore será binária, então a inserção funcionará da seguinte forma, ela irá receber a Raiz da árvore e um No 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, 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 e quando valor for menor fazemos uma chamada recursiva passando a **Raiz.esq** e assim por diante até que a Raiz seja Nula e faremos com que a Raiz receba o No.

altura_no(): Recebe um Nó e devolve o comprimento do caminho do Nó recebido até folha a maior profundidade do Nó.

verifica_balanceamento(): 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()**.

questão 2 Insere(): Na questão 2, nossa árvore será AVL, então a inserção funcionará da seguinte forma, ela seguirá os mesmos passos da inserção da questão 1, 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.

Tempo_gasto(): recebe o tempo inicial e o final da execução e calcula e retorna o tempo gasto em milissegundos.

inserir_valores(): Essa função irá receber como entrada um ponteiro para uma estrutura do tipo `arvorebin`, que armazenará as informações de um Nó da árvore e esse ponteiro iniciará como nulo e será a raiz da árvore, esta função será responsável por gerar 1000 valores aleatórios e inseri-los na árvore, para isso teremos um laço de repetição que rodará 1000 vezes a cada laço é gerado um número pela função **num_aleatorio()** que irá devolver um valor aleatório entre 0 e 20.000 após isso é alocado um espaço na memória para um Nó da árvore pela função **aloca()**, após isso é inserido o No com o valor gerado, na árvore pela função **insere()**, por fim calculado e mostrado o tempo gasto para de inserção dos 1000 valores na árvore pela função **Tempo_gasto()**.

calcular_tempo_busca(): Recebe Raiz da árvore, essa função irá buscar um número na árvore utilizando a função **busca()** e irá calcular o tempo gasto para a busca do número utilizando a função **Tempo_gasto()**.

obs: Para todas as árvores geradas na questão 1 e 2 o tempo de busca foi calculado, buscando o número 19674 que pode ou não ter sido inserido nas árvores.

profundidade_no(): Recebe a Raiz da árvore e um valor de um Nó e devolve comprimento do caminho da raiz até ao Nó que armazena o valor recebido.

menor_profundidade(): Recebe Raiz da árvore e percorre toda a árvore e quando encontra um Nó folha calcula sua profundidade utilizando a função **profundidade_no()** e verifica se a folha possui a menor profundidade, após percorrer todas as folhas devolve o valor da distância da folha de menor profundidade.

calcular_profundidade(): A cada árvore gerada, esta função irá receber a Raiz da árvore e calcular o nível da folha de maior profundidade utilizando a função **altura_no()** passando a Raiz da árvore e como altura da árvore é comprimento do caminho do Nó recebido até a folha a maior profundidade do Nó, recebemos assim o nível da folha de maior profundidade, após isso calculamos o nível da folha de menor profundidade utilizando a função **menor_profundidade()** e depois é retornado pela função a diferença entre os níveis da folha de maior profundidade e a de menor profundidade.

mostrar_diferenca_profundidades(): Após cada árvore gerada é armazenado a diferença entre os níveis da folha de maior profundidade e a de menor profundidade em um vetor de profundidades, então esta função com ajuda de outras auxiliares irá mostrar a frequência em que diferença entre os níveis da folha ocorreram.

Problema 3 e 4:

Resumo do problema:

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 portugues, Remover palavra em portugues e suas equivalentes em inglês.

Resumo solução do problema:

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 portugues 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:

aloca_arvore(): 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(): Recebe um ponteiro para uma estrutura do tipo Lista e uma string e coloca um Nó para lista utilizando a função **aloca_lista()** e depois insere o Nó na lista utilizando a função **insere_lista()**.

questão 3 Insere_arvore() : 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()**.

altura_no(): Recebe um Nó e devolve o comprimento do caminho do Nó recebido até folha a maior profundidade do Nó.

verifica_balanceamento(): 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()**.

questão 4 Insere_arvore(): 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.

Tempo_gasto(): recebe o tempo inicial e o final da execução e calcula e retorna o tempo gasto em milissegundos .

ler_dados(): 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 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(): 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(): 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.

questao3 remove(): 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.

questao 4 remove(): 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(): 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.

calcula_tempo_busca(): Recebe o vetor de árvore, e busca uma palavra nas árvores e calcula o tempo de busca.

- **Resultados da Execução do Programa:**

Neste tópico iremos analisar dado algumas entradas como os codigos dao suas saídas:

Todas as questões a seguir foram executadas pelo terminal linux.

Questão 1:

```
Atividades Terminal
daviddd@daviddd-300E5M-300E5L

Arvore: 27
Tempo gasto insercao: 0.285 ms.
Tempo busca numero 19674: 0.002 ms
maior profundidade: 22
menor profundidade: 5

Arvore: 28
Tempo gasto insercao: 0.269 ms.
Tempo busca numero 19674: 0.002 ms
maior profundidade: 21
menor profundidade: 4

Arvore: 29
Tempo gasto insercao: 0.268 ms.
Tempo busca numero 19674: 0.001 ms
maior profundidade: 20
menor profundidade: 6

Arvore: 30
Tempo gasto insercao: 0.28 ms.
Tempo busca numero 19674: 0.002 ms
maior profundidade: 24
menor profundidade: 5

Quantidade de vezes em que diferenca entre as profundiades das Arvores foi 16: 4
Quantidade de vezes em que diferenca entre as profundiades das Arvores foi 18: 5
Quantidade de vezes em que diferenca entre as profundiades das Arvores foi 15: 3
Quantidade de vezes em que diferenca entre as profundiades das Arvores foi 17: 5
Quantidade de vezes em que diferenca entre as profundiades das Arvores foi 19: 2
Quantidade de vezes em que diferenca entre as profundiades das Arvores foi 20: 4
Quantidade de vezes em que diferenca entre as profundiades das Arvores foi 14: 4
Quantidade de vezes em que diferenca entre as profundiades das Arvores foi 21: 2
Quantidade de vezes em que diferenca entre as profundiades das Arvores foi 13: 1

daviddd@daviddd-300E5M-300E5L:~/Documentos/ESTRUTURA DE DADOS 2/Trabalho-1$
```

Na imagem acima é possível ver o teste da questão 1 , e mostra como o código da sua saída de acordo com os valores recebidos de entrada.

questão 2:



```
Atividades Terminal ▼
daviddd@daviddd-300E5M-300E5L:~$

Arvore: 26
Tempo gasto insercao: 24.338 ms.
Tempo busca numero 19674: 0.002 ms
maior profundidade: 11
menor profundidade: 8

Arvore: 27
Tempo gasto insercao: 25.586 ms.
Tempo busca numero 19674: 0.001 ms
maior profundidade: 11
menor profundidade: 7

Arvore: 28
Tempo gasto insercao: 25.196 ms.
Tempo busca numero 19674: 0.001 ms
maior profundidade: 11
menor profundidade: 7

Arvore: 29
Tempo gasto insercao: 23.921 ms.
Tempo busca numero 19674: 0.001 ms
maior profundidade: 11
menor profundidade: 7

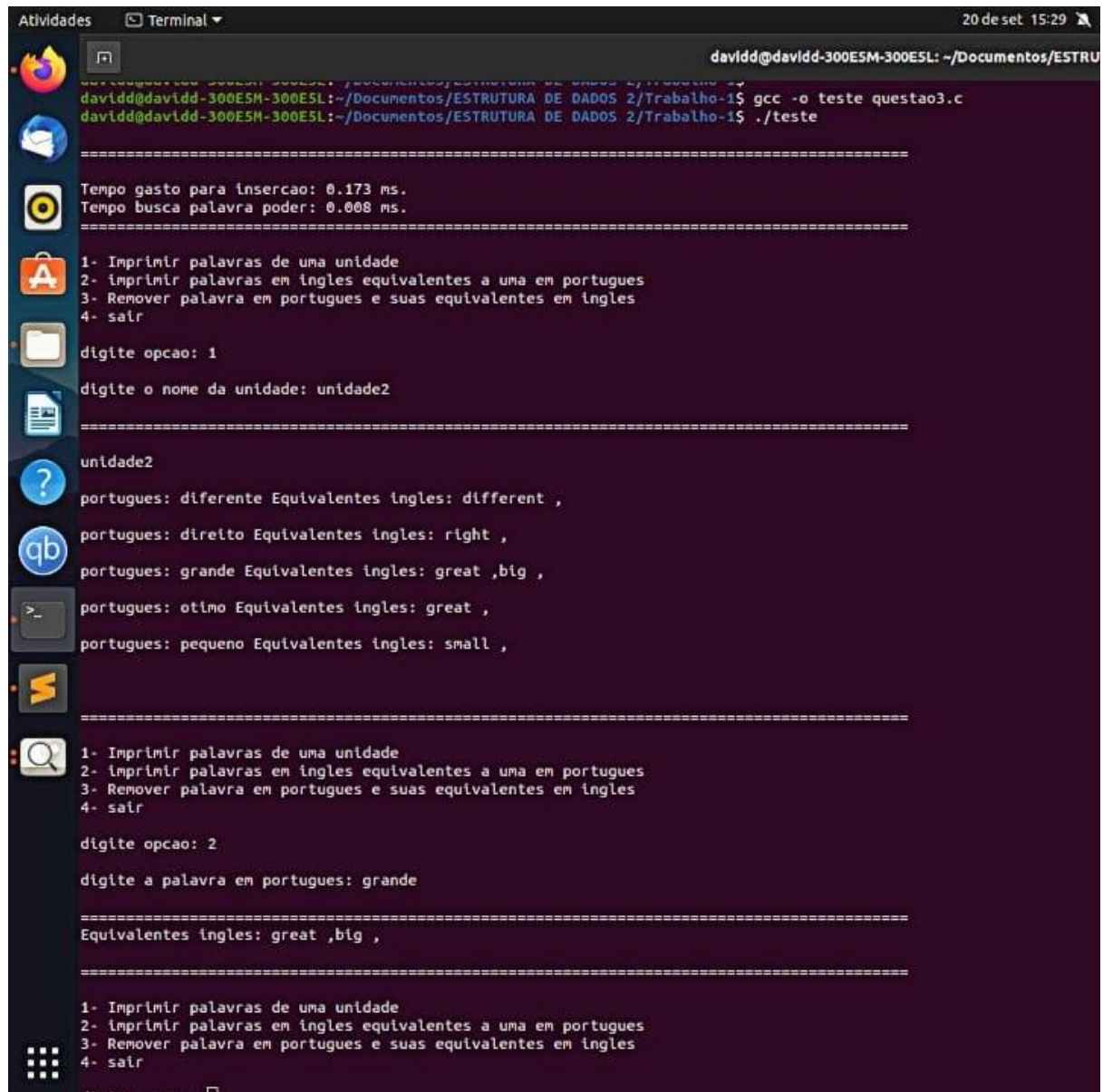
Arvore: 30
Tempo gasto insercao: 23.884 ms.
Tempo busca numero 19674: 0.001 ms
maior profundidade: 11
menor profundidade: 8

Quantidade de vezes em que diferenca entre as profundidades das Arvores foi 4: 27
Quantidade de vezes em que diferenca entre as profundidades das Arvores foi 3: 3

daviddd@daviddd-300E5M-300E5L:~/Documentos/ESTRUTURA DE DADOS 2/Trabalho-1$
```

Na imagem acima é possível ver o teste da questão 2 , e mostra como o código da sua saída de acordo com os valores recebidos de entrada.

Questão 3:



```
Atividades Terminal 20 de set 15:29
daviddd@daviddd-300ESM-300ESL: ~/Documentos/ESTRUTURA DE DADOS 2/Trabalho-1
daviddd@daviddd-300ESM-300ESL:~/Documentos/ESTRUTURA DE DADOS 2/Trabalho-1$ gcc -o teste questao3.c
daviddd@daviddd-300ESM-300ESL:~/Documentos/ESTRUTURA DE DADOS 2/Trabalho-1$ ./teste
=====
Tempo gasto para insercao: 0.173 ms.
Tempo busca palavra poder: 0.008 ms.
=====
1- Imprimir palavras de uma unidade
2- imprimir palavras em ingles equivalentes a uma em portugues
3- Remover palavra em portugues e suas equivalentes em ingles
4- sair
digite opcao: 1
digite o nome da unidade: unidade2
=====
unidade2
portugues: diferente Equivalentes ingles: different ,
portugues: direito Equivalentes ingles: right ,
portugues: grande Equivalentes ingles: great ,big ,
portugues: otimo Equivalentes ingles: great ,
portugues: pequeno Equivalentes ingles: small ,
=====
1- Imprimir palavras de uma unidade
2- imprimir palavras em ingles equivalentes a uma em portugues
3- Remover palavra em portugues e suas equivalentes em ingles
4- sair
digite opcao: 2
digite a palavra em portugues: grande
=====
Equivalentes ingles: great ,big ,
=====
1- Imprimir palavras de uma unidade
2- imprimir palavras em ingles equivalentes a uma em portugues
3- Remover palavra em portugues e suas equivalentes em ingles
4- sair
```

Na imagem acima é possível ver o teste da questão 3 , e mostra como o código da sua saída o usuário escolhe as opções 1 e 2 do menu.

Questão 4:

```
davidd@davidd-300E5M-300E5L:~/Documentos/ESTRUTURA DE DADOS 2/Trabalho-1$ gcc -o teste questao4.c
davidd@davidd-300E5M-300E5L:~/Documentos/ESTRUTURA DE DADOS 2/Trabalho-1$ ./teste

=====
Tempo gasto para insercao: 0.372 ms.
Tempo busca palavra poder: 0.01 ms.
=====

1- Imprimir palavras de uma unidade
2- imprimir palavras em ingles equivalentes a uma em portugues
3- Remover palavra em portugues e suas equivalentes em ingles
4- sair

digite opcao: 1

digite o nome da unidade: unidade2

=====
unidade2

portugues: diferente Equivalentes ingles: different ,
portugues: direito Equivalentes ingles: right ,
portugues: grande Equivalentes ingles: great ,big ,
portugues: otimo Equivalentes ingles: great ,
portugues: pequeno Equivalentes ingles: small ,

=====

1- Imprimir palavras de uma unidade
2- imprimir palavras em ingles equivalentes a uma em portugues
3- Remover palavra em portugues e suas equivalentes em ingles
4- sair

digite opcao: 2

digite a palavra em portugues: grande

=====
Equivalentes ingles: great ,big ,
=====
```

Na imagem acima é possível ver o teste da questão 4 , e mostra como o código da sua saída o usuário escolhe as opções 1 e 2 do menu.

Análise de desempenho:

Os códigos acima foram desenvolvidos e testados utilizando o Notebook com o processador Intel I3 e com memória RAM de 4GB.

Questão 1 e 2:

Desempenho árvore Binária:

- inserção menor tempo: 0.28 ms.
- inserção maior tempo: 1.109 ms.
- Busca menor tempo: 0.001 ms
- Busca maior tempo: 0.006 ms

Desempenho árvore AVL:

- inserção menor tempo: 22.549 ms.
- inserção maior tempo: 25.586 ms.
- Busca menor tempo: 0.001 ms
- Busca maior tempo: 0.002 ms

Analisando os valores obtidos com a execução dos códigos podemos observar que na árvore binária temos um tempo de inserção bem curto, já quando analisamos o tempo de inserção da árvore AVL obtemos um tempo bem maior do que a árvore binária.

Analisando os valores obtidos na busca, observa-se que na árvore binária temos uma certa diferença entre seu menor tempo de busca e seu maior tempo, já na árvore AVL temos uma diferença muito curta entre seus tempos.

Conclusão, analisando os dados podemos concluir que na inserção a árvore binária possui um melhor desempenho em relação a inserção de valores do que a árvore AVL, já na busca a árvore AVL obteve um melhor desempenho do que a binária, porém diferentemente da inserção, onde a diferença de desempenho das árvores foi bem grande, na busca não temos uma grande diferença de desempenho entre as árvores.

obs: Para todas as árvores geradas na questão 1 e 2 o tempo de busca foi calculado, buscando o número 19674 que pode ou não ter sido inserido nas árvores.

Questão 3 e 4:

Desempenho árvore Binária:

- Tempo gasto para inserção: 0.173 ms.
- Tempo busca palavra poder: 0.008 ms.

Desempenho árvore AVL:

- Tempo gasto para inserção: 0.372 ms.
- Tempo busca palavra poder: 0.01 ms.

Analizando os valores obtidos com a execução dos códigos podemos observar que na árvore binária temos um tempo de inserção bem curto, já quando analisamos o tempo de inserção da árvore AVL obtemos um tempo maior do que a árvore binária.

Analizando os valores obtidos na busca observamos que a árvore AVL possui um melhor desempenho do que a árvore binária.

Conclusão, analisando os dados podemos concluir que na inserção a árvore binária possui um melhor desempenho que a árvore AVL, já na busca a árvore AVL obteve um melhor desempenho do que a binária, porém tanto na busca quanto na inserção, não temos uma grande diferença de desempenho entre as árvores.

obs: Para cálculo do tempo de busca na questão 3 e 4 foi calculado, buscando a palavra **poder** que foi inserido em ambas árvores das questões.

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, então podemos ver que através desta atividade prática foi possível aumentar o nosso conhecimento sobre a linguagem de programação C e sobre conceitos importantes para a disciplina de estruturas de dados 2.