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-Pi, Outubro 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: ler palavras de um arquivo texto e gerar um árvore onde cada Nó da árvore terá uma palavra e uma lista que armazena a linha onde a palavra se encontra no texto e desenvolver um programa que simule o gerenciamento de memória de um Sistema Operacional. 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 vermelha e preta e árvores 2-3.

- **Seções específicas:**

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

  Problema 1 e 2:

  Resumo do problema:

  Faça um programa em C de referência cruzada que construa uma árvore binária de busca vermelha-preta, com todas as palavras incluídas a partir de um arquivo texto, e armazene o número de cada linha em que a palavra foi usada. Os números das linhas devem ser armazenados em uma lista encadeada associada ao nó da árvore. Depois do arquivo de entrada ter sido processado, imprima em ordem alfabética todas as palavras do arquivo texto, junto com os números das linhas onde foi usada. O programa deve permitir ao usuário buscar uma palavra e o programa deve responder em quais linhas do texto ela foi utilizada, para cada palavra buscada mostre o número de passos até alcançar a palavra na árvore. Além disso, ele deve permitir excluir uma palavra de uma linha bem como acrescentar.

Resumo da solução:

Para solução do problema proposto foi desenvolvido as seguintes funções:

questão 1 aloca(): Essa função irá receber como entrada um ponteiro para uma estrutura do tipo arvoreVP, que armazenará as informações de um Nó da árvore, esse ponteiro irá apontar para um espaço alocado na memória para um Nó , e esse ponteiro será devolvido pela função por referência.

Questão 1 insere(): Essa função recebe a Raiz e um Nó com uma palavra e a linha onde a palavra foi encontrada no texto e insere o Nó na árvore se a palavra não tiver sido inserida na árvore, se já tiver sido inserida apenas insere a linha na lista do No.

Questão 1 remover(): Recebe a Raiz da árvore e a palavra que o usuário deseja remover, esta função irá buscar e remover o Nó com a palavra escolhida.

Na árvore vermelha e preta utilizada na questão 1, toda vez que removemos ou inserimos um Nó, é verificado após a operação se é necessário fazer o balanceamento da árvore, para isso são utilizadas funções auxiliares.

Questão 2 insere(): Essa função recebe um No pai inicia sempre com NULL a Raiz da árvore e uma info contendo a palavra que queremos inserir e a lista da palavra, então essa função insere a palavra na árvore.

Questão 2 remover(): Essa função recebe um No pai inicia sempre com NULL a Raiz da árvore e a palavra que queremos remover e a função irá buscar e remover a palavra se for encontrada.

Na árvore 2-3 utilizada na questão 2 a inserção é feita de forma que a árvore fique sempre balanceada , para isso é utilizado algumas funções auxiliares e a remoção também é feita de uma forma que ao remover uma info a árvore se reorganiza para não se desbalancear para isso também utilizadas funções auxiliares

As funções descritas abaixo seguem praticamente a mesma lógica, tanto para a questão 1 quanto para a questão 2.

trata_frase(): Recebe uma string contendo uma frase do texto quebra a string dividindo pelos espaços da string e armazena as palavras num vetor e devolve o vetor e a quantidade de palavras inseridas no vetor.

Ler_dados(): Essa função é responsável por fazer o processamento do arquivo texto, ela recebe o arquivo e a Raiz da árvore, então será lido as linhas do arquivo e é armazenada numa string, a cada linha lida é chamada a função **trata_frase():** que recebe a string e devolve um vetor com as strings da linha, após isso é chamada a função **Inserir_palavras():** passando o vetor

com as strings e insere as palavras do vetor na árvore.

Inserir_palavras(): recebe a raiz da árvore e um vetor de strings com as palavras de uma linha do texto e a linha correspondente e percorre o vetor insere todas as palavras na árvore.

busca_plv(): Recebe a Raiz da árvore e uma palavra e busca a palavra na árvore imprime e mostra quantos passos foram percorridos para encontrar ou não a palavra

remover_linha(): Recebe a raiz da árvore, a palavra que queremos remover e a linha que queremos remover e a função irá buscar a palavra e irá remover da lista da palavra a linha em que a palavra aparece, por fim é devolvido 0 se a linha ou palavra não foi encontrada é devolvido 1 se a linha foi removida e a lista da palavra ainda possui valores é devolvido -1 se a linha foi removida e a lista não mais possui valores.

remover_info(): Recebe a raiz da árvore, a palavra que queremos remover e a linha onde a palavra se encontra, então buscamos e removemos a linha em que a palavra aparece utilizando a função **remover_linha():** se após remover a linha foi devolvido -1 quer dizer que a lista da palavra que queremos remover está vazia então chamamos a função **remove()** para remover a palavra da árvore.

Problema 3 e 4:

Resumo do problema:

Suponha que uma memória seja dividida em blocos lógicos de 1Mbyte, e que o primeiro bloco é o bloco 0, suponha também que o gerenciador de memória de um Sistema Operacional mantenha uma árvore 2-3 com nós para blocos livres e ocupados da memória. Cada nó da árvore 2-3 contém os seguintes campos: O- para ocupado ou L- para livre, o número do bloco inicial, o número do bloco final, endereço inicial (correspondente ao endereço inicial do primeiro bloco do nó) e endereço final (correspondente ao endereço final do último bloco do nó). Os nós da árvore será organizados de forma que um nó ocupado está entre 2 nós livres e vice-versa.

Resumo da solução:

Para solução do problema proposto foi desenvolvido as seguintes funções:

insere(): Essa função recebe um No pai inicia sempre com NULL a Raiz da árvore e uma info contendo o início do bloco o fim do bloco e o estado do bloco ('o' = ocupado , 'l' = livre) , então essa função insere a info na árvore.

inserir_valores(): Recebe a Raiz da árvore e a quantidade de memória que o usuário deseja armazenar em Mbytes, então é pedido para que o usuário escolha o estado do primeiro bloco se é livre ou ocupado, então a partir do bloco 0 é pedido para que o usuário digite o final do bloco e é armazenado numa estrutura com o início do bloco , o fim do bloco e o estado e é chamado a função inserir para **insere()** para inserir a informação na árvore após isso é mudado o estado do bloco e pedido para que o usuário digite o final do próximo bloco e assim sucessivamente até que o usuário preencha todos os blocos da memória.

busca_Menor_No(): Recebe um Nó e busca a partir do Nó o Nó com a menor informação e devolve este No.

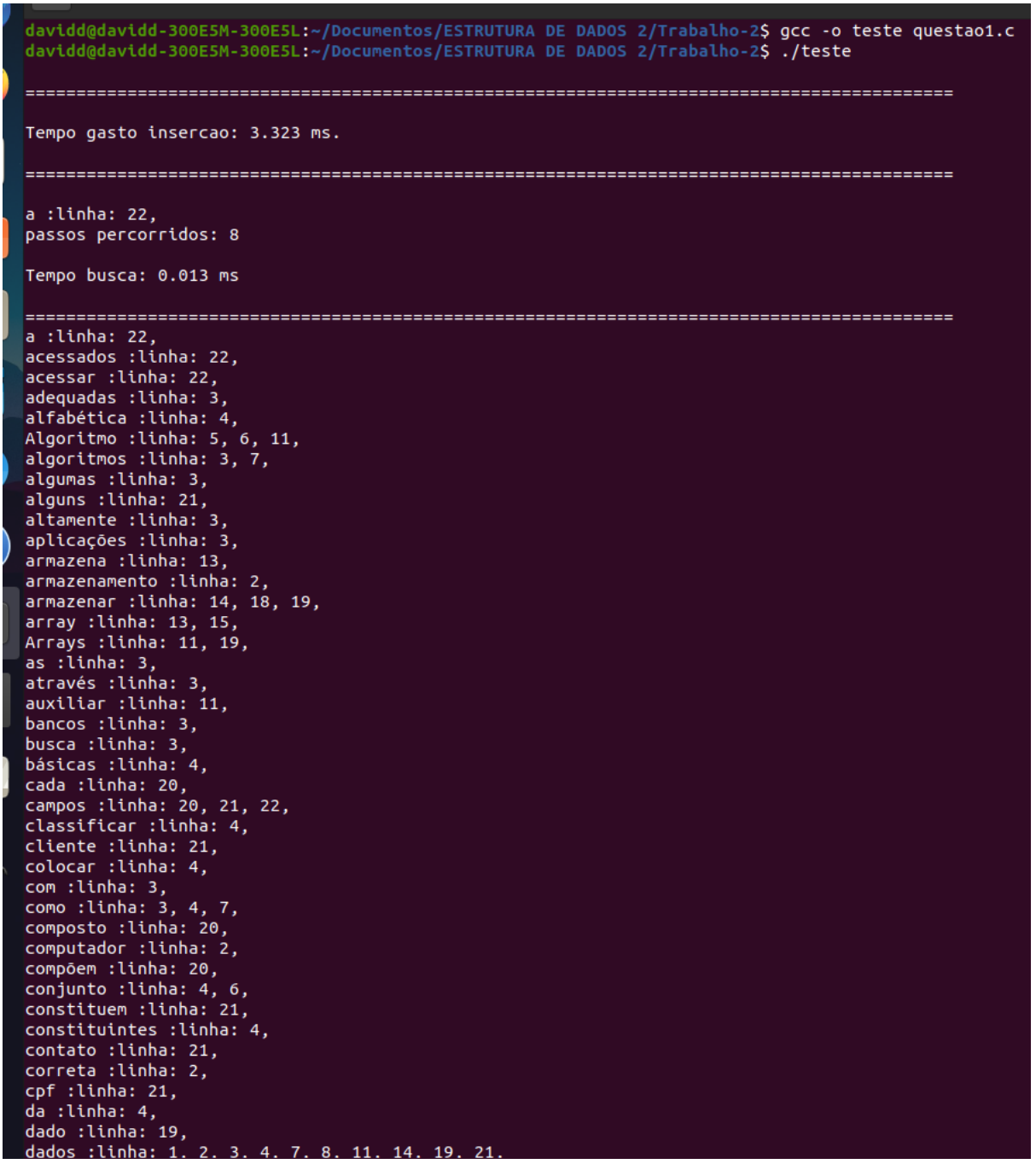
ocupa_espaco(): Recebe o pai que inicia com Null, a raiz da árvore e a quantidade de blocos da memória que o usuário se deseja ocupar, esta função irá buscar o No com a info que possua a quantidade blocos de memória livre necessária para que consigamos ocupar o espaço escolhido pelo o usuário, se for encontrado é ocupado o espaço devido e é devolvido 1 pela função para indicar que a operação foi concluída com sucesso, se não é devolvido 0 para indicar que não foi encontrado um Nó com a quantidade de memória necessária para ocupar o espaço escolhido.

- **Resultados da Execução do Programa:**

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

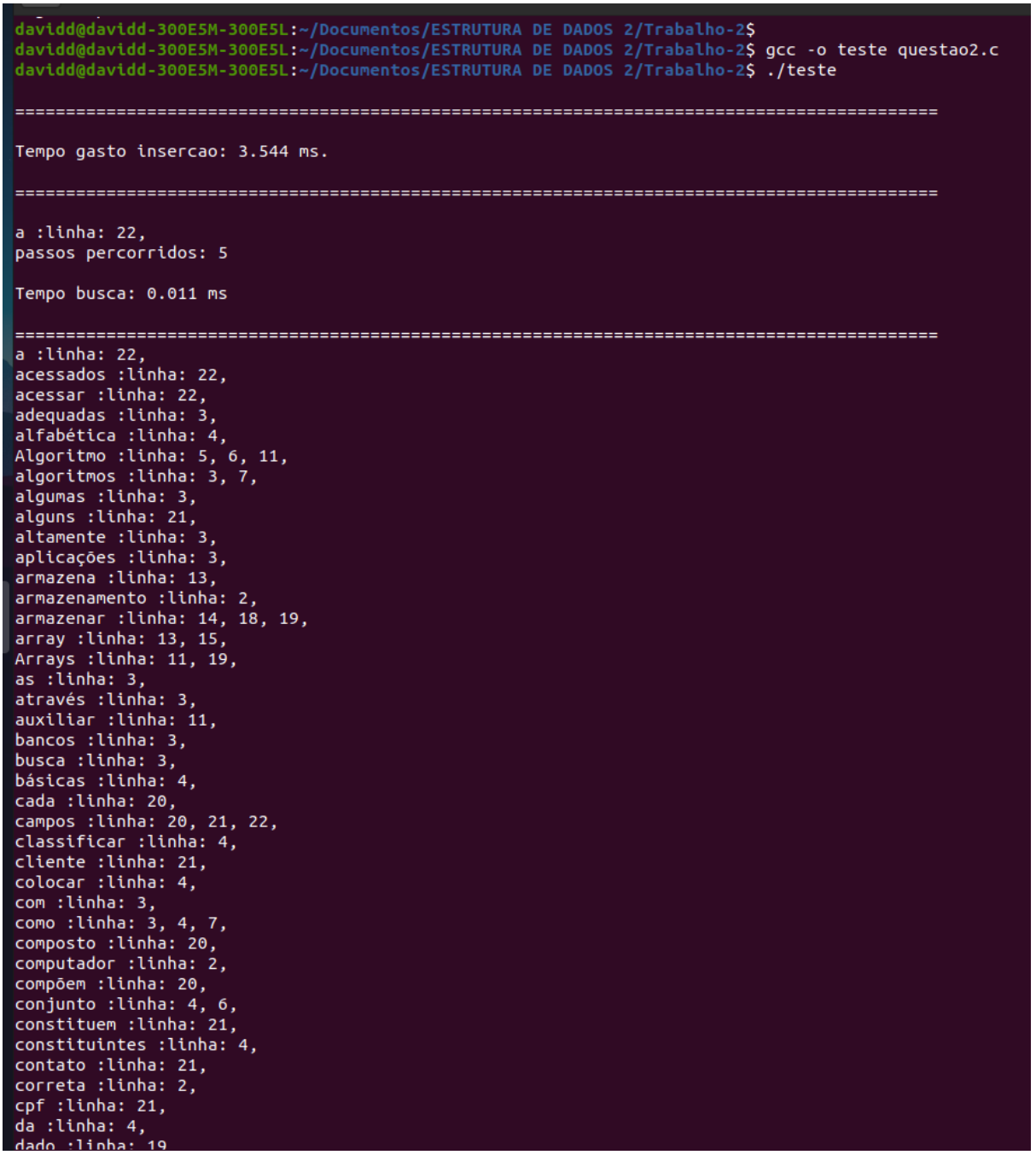
Todas as questões a seguir foram executadas pelo terminal linux.

Questão 1:



Na imagem acima podemos ver como o código da sua saída, quando executado, podemos ver que ao executar o código é mostrado o tempo de inserção das palavras na árvore e o tempo de busca de uma palavra, após isso é mostrado todas as palavras inseridas na árvore.

Questão 2:



Na imagem acima podemos ver como o código da sua saída, quando executado, podemos ver que ao executar o código é mostrado o tempo de inserção das palavras na árvore e o tempo de busca de uma palavra, após isso é mostrado todas as palavras inseridas na árvore.

Questão 3:

```
davidd@davidd-300E5M-300E5L:~/Documentos/ESTRUTURA DE DADOS 2/Trabalho-2$ gcc -o teste questao3.c
davidd@davidd-300E5M-300E5L:~/Documentos/ESTRUTURA DE DADOS 2/Trabalho-2$ ./teste

================================================================================================

digite o tamanho da memoria em Mbyte: 100

================================================================================================
digite o estado do primeiro bloco(o = ocupado ou l = livre): l

================================================================================================

inicio do bloco: 0
digite o fim do bloco: 20

================================================================================================

inicio do bloco: 21
digite o fim do bloco: 50

================================================================================================

inicio do bloco: 51
digite o fim do bloco: 80

================================================================================================

inicio do bloco: 81
digite o fim do bloco: 99

================================================================================================
1- imprimir arvore
2- Ocupar spaco
3- sair

digite opcao: 1

================================================================================================
bloco: 0 , 20 -l
bloco: 21 , 50 -o
bloco: 51 , 80 -l
bloco: 81 , 99 -o

================================================================================================
1- imprimir arvore
2- Ocupar spaco
3- sair

digite opcao: 2

================================================================================================

digite o tamanho do bloco que voce deseja ocupar: 10

Operacao concluida!!
```

Na imagem acima podemos ver como o código da sua saída, quando executado, podemos ver que ao executar o código é pedido para que o usuário digite a quantidade de memória que o usuário deseja alocar e também é pedido para que o usuário escolha o estado do primeiro bloco.

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 árvores:

- Tempo de inserção da árvore vermelha e preta: 3.323 ms.
- Tempo de busca da árvore vermelha e preta: 0.13 ms.
- Tempo de inserção da árvore 2-3: 3.544 ms.
- Tempo de busca da árvore 2-3: 0.011 ms.

Analisando os valores obtidos com a execução dos códigos podemos observar que na árvore vermelha e preta temos um tempo de inserção um pouco menor do que na árvore 2-3.

Analisando os valores obtidos na busca, observa-se que na árvore 2-3 temos um tempo de busca menor do que a árvore vermelha e preta.

Conclusão, analisando os tempos de busca e inserção, tanto na arvore 2-3 quanto na árvore vermelha e preta, podemos notar que ambas neste problema possuíam desempenhos parecidos, mostrando que na inserção a árvore vermelha e preta obteve uma leve vantagem de desempenho que a árvore 2-3 e na busca a árvore 2-3 teve um desempenho um pouco melhor.

obs: Para o cálculo do tempo de busca questão 1 e 2 o tempo de busca foi calculado, buscando a string “a” inserida na árvore.

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 vermelha e preta e árvore 2-3, 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.