Roteiro 13 - Laboratório de Programação 2

Nome: Geraldo Arthur Detomi

1.1 Execute o código fornecido para o MergeSort externo fornecido em sala (disponível no drive) e explique de forma breve e resumida o que faz cada uma das funções que estão presentes nesse código. Obs: Não precisa apresentar o código ou a tela no relatório, apenas as respostas solicitadas.

```
void criaArquivoTeste(char* nome, int ini, int fim)
```

Esta função é responsável por criar um arquivo com o **nome** especificado e preenchê-lo com 1000 valores aleatórios dentro do intervalo definido por **ini** e **fim**, vale mencionar que cada valor aleatório gerado ocupa uma linha do arquivo gerado.

```
void troca(int* a, int *b);
int particao(int *v, int ini, int fim);
void QuickSort(int *v, int ini, int fim);
```

A função **QuickSort** ordena um intervalo de um array, delimitado por **ini** e **fim**. As funções **troca** e **particao** são auxiliares ao **QuickSort**.

```
void salvaArquivo(char *nome, int *v, int tam, int mudaLinhaFinal);
```

Esta função cria ou edita um arquivo existente, mantendo seus dados. Ela escreve o vetor fornecido **v** de tamanho **tam** no arquivo especificado por **nome**. O parâmetro **mudaLinhaFinal** controla a inserção de uma quebra de linha após o último elemento.

```
int criaArquivosOrdenados(char *nome);
```

A função lê os números preenchidos em um arquivo, armazena os dados lidos em um vetor temporário de tamanho **N**, ordena esse vetor usando o **quicksort**, e salva os valores em arquivos temporários separados. Cada arquivo temporário contém N números já ordenados. Retorna a quantidade total de arquivos temporários gerados.

```
void preencheBuffer(Arquivo* arq, int T)
```

Esta função tenta preencher o buffer do arquivo arq com até T valores inteiros, lidos sequencialmente do arquivo associado. Os campos pos e max são atualizados para indicar o número de valores lidos e a posição atual de leitura. Se o arquivo chegar ao fim, ele é fechado e o ponteiro é marcado como NULL.

```
int procuraMenor(Arquivo* arq, int K, int T, int* menor)
```

Essa função percorre os buffers de **K** arquivos temporários procurando o menor valor

disponível entre todos eles. Se encontrar algum valor, armazena em *menor e avança o ponteiro da posição atual no buffer de onde esse valor veio. Se esse buffer acabar, ele é recarregado com **preencheBuffer**. Retorna 1 se um valor foi encontrado, ou 0 se todos os arquivos estiverem esgotados.

```
void multiWayMerge(char *nome, int K, int T)
```

Executa o processo de merge externo. Abre os **K** arquivos temporários criados anteriormente, aloca buffers para cada um e realiza um multi-way merge.
A cada iteração, escolhe o menor elemento disponível entre os buffers, armazena em um vetor temporário de saída e, quando esse vetor atinge tamanho **T**, grava os dados no arquivo de saída **nome**. Ao final, grava os dados restantes e libera toda a memória alocada.

```
void MergeSortExterno(char* nome)
```

Função principal do algoritmo de ordenação externa. Ela primeiro chama **criaArquivosOrdenados** para dividir e ordenar blocos do arquivo original.
Depois, remove esse arquivo original para dar lugar ao novo, já ordenado, que será produzido por **multiWayMerge**.
Por fim, todos os arquivos temporários são removidos do sistema, deixando apenas o arquivo final ordenado.

```
int main()
```

Função principal do programa. Cria um arquivo de teste chamado **dados.txt** com números aleatórios, faz uma cópia de segurança **dados2.txt** e chama o algoritmo de ordenação externa **MergeSortExterno** para ordenar os dados de forma eficiente, e criar um novo arquivo só que ordenado

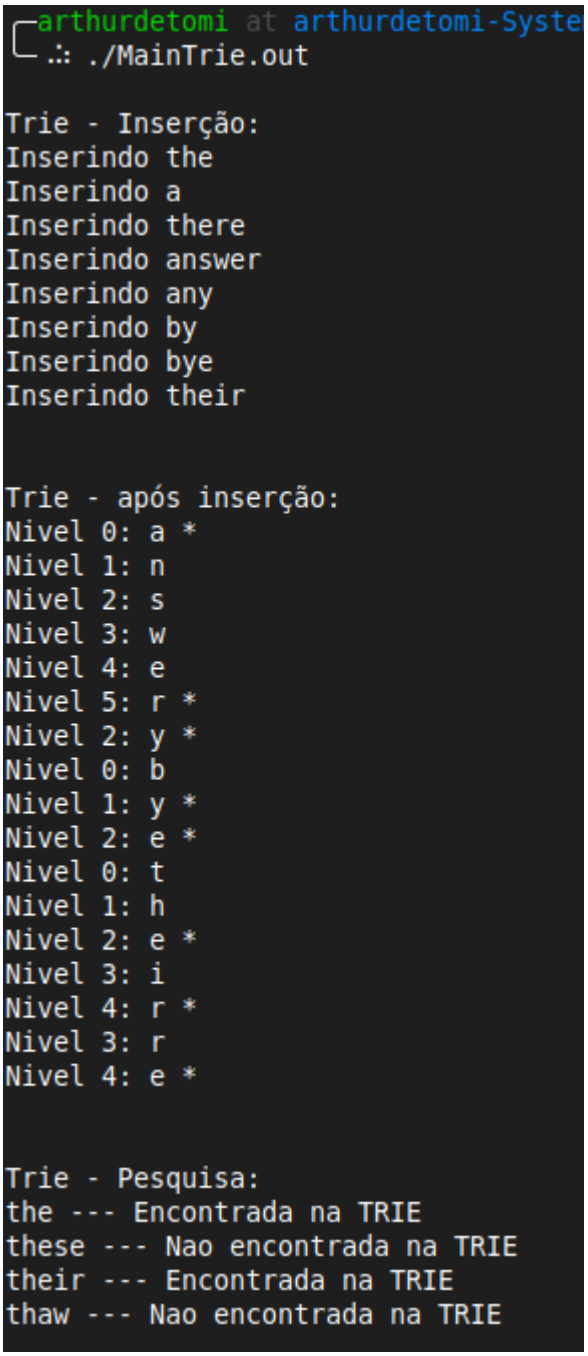
1.2 Faça a execução dos códigos fornecidos em aula (disponível no drive) para os TADs das três árvores (Árvore B, Trie e Patricia) e altere a respectiva Main para mostrar o resultado de cada árvore, apenas após fazer todas as inserções, os retornos das respectivas operações de busca (se houver), e o estado final da árvore após todas as remoções (se houver). Obs: Não precisa incluir o código no relatório, apenas a tela de saída de cada execução.

As fotos das execuções estão uma em cada página abaixo:

Execução Árvore B:

```
./MainArvoreB.out
Arvore B - Inserção:
Inseriu 20..
Inseriu 10..
Inseriu 40..
Inseriu 50..
Inseriu 30..
Inseriu 55..
Inseriu 3..
Inseriu 11..
Inseriu 4..
Inseriu 28..
Inseriu 36..
Inseriu 33..
Inseriu 52..
Inseriu 17..
Inseriu 25..
Inseriu 13..
Inseriu 45..
Inseriu 9..
Inseriu 43..
Inseriu 8..
Inseriu 48..

Arvore B - Apos inserção:
Nivel 0: 30
Nivel 1: 10 20
Nivel 2: 3 4 8 9
Nivel 2: 11 13 17
Nivel 2: 25 28
Nivel 1: 40 50
Nivel 2: 33 36
Nivel 2: 43 45 48
Nivel 2: 52 55


Arvore B - Pesquisa:
Registro (chave 20) encontrado!
Registro (chave 10) encontrado!
Registro (chave 40) encontrado!
Registro (chave 50) encontrado!
Registro (chave 30) encontrado!
Registro (chave 55) encontrado!
Registro (chave 3) encontrado!
Registro (chave 11) encontrado!
Registro (chave 4) encontrado!
Registro (chave 28) encontrado!
Registro (chave 36) encontrado!
Registro (chave 33) encontrado!
Registro (chave 52) encontrado!
Registro (chave 17) encontrado!
Registro (chave 25) encontrado!
Registro (chave 13) encontrado!
Registro (chave 45) encontrado!
Registro (chave 9) encontrado!
Registro (chave 43) encontrado!
Registro (chave 8) encontrado!
Registro (chave 48) encontrado!

Arvore B - Remoção:
Removendo 45..
Removendo 30..
Removendo 28..
Removendo 50..
Removendo 8..
Removendo 10..
Removendo 4..
Removendo 20..
Removendo 40..
Removendo 55..
Removendo 17..
Removendo 33..
Removendo 11..
Removendo 36..
Removendo 3..
Removendo 9..
Removendo 52..

Arvore B - Após remoção:
Nivel 0: 13 25 43 48
```

Execução Trie:



Execução Patricia:

```
arthurdetomi at arthurdetomi-System-Product-Name i
.:: ./MainPatricia.out

Patricia - Inserção:
18 em binario: 1 0 0 1 0
Inserindo 18..
19 em binario: 1 0 0 1 1
Inserindo 19..
Elemento 19 NAO encontrado!
Bit diferente eh: [6]
24 em binario: 1 1 0 0 0
Inserindo 24..
Elemento 24 NAO encontrado!
Bit diferente eh: [3]
33 em binario: 1 0 0 0 0 1
Inserindo 33..
Elemento 33 NAO encontrado!
Bit diferente eh: [1]
40 em binario: 1 0 1 0 0 0
Inserindo 40..
Elemento 40 NAO encontrado!
Bit diferente eh: [3]
54 em binario: 1 1 0 1 1 0
Inserindo 54..
Elemento 54 NAO encontrado!
Bit diferente eh: [2]
34 em binario: 1 0 0 0 1 0
Inserindo 34..
Elemento 34 NAO encontrado!
Bit diferente eh: [5]

Patricia - Após inserção:
Nivel 0: (INT) 1
Nivel 1: (INT) 3
Nivel 2: (INT) 6
Nivel 3: (EXT) 18
Nivel 3: (EXT) 19
Nivel 2: (EXT) 24
Nivel 1: (INT) 2
Nivel 2: (INT) 3
Nivel 3: (INT) 5
Nivel 4: (EXT) 33
Nivel 4: (EXT) 34
Nivel 3: (EXT) 40
Nivel 2: (EXT) 54


Patricia - Pesquisa:
Elemento 18 encontrado!
Elemento 19 encontrado!
Elemento 24 encontrado!
Elemento 33 encontrado!
Elemento 40 encontrado!
Elemento 54 encontrado!
Elemento 34 encontrado!
```