Pesquisa e Classificação de Dados - Prova 2 - C

2020/1 - ADNP – Prof. Ricardo Oliveira

Instruções: Envie um pacote (**zip** ou **tar.gz**) contendo:

- Para as questões 2, 3 e 5, arquivos legíveis (como arquivos de texto, PDFs, imagens, fotos, etc);
- Para as questões 1 e 4, dois programas em C (`.c`), um para cada questão.

Certifique-se de estar resolvendo a versão correta da prova, de acordo com seu RA. Envie um pacote **zip** ou **tar.gz**; pacotes em outros formatos que não puderem ser abertos pelo professor não serão corrigidos. Respostas detectadas como plágios ou cópias receberão nota zero.

---

1. (25 pontos) Escreva um programa em C que lê do usuário um inteiro N seguido de uma sequência de N inteiros que representa o percurso *pos-order* de uma BST, e determina se esta BST é ou não uma AVL.

Exemplos de entrada	**Exemplos de saída**
`8` `5 4 7 6 50 71 23 15`	`nao eh AVL`
`7` `4 6 5 23 71 50 15`	`eh AVL`

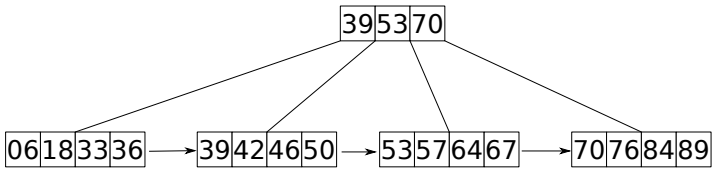
   Você pode assumir que o usuário sempre entrará com um percurso válido, que não haverá inteiros repetidos, e que $N \leq 100$.

2. (20 pontos) Tipicamente, a *struct* de um nodo de uma trie contém um vetor de $|\Sigma|$ ponteiros (sendo $|\Sigma|$ o tamanho do alfabeto), um para cada eventual filho do nodo. Esta *struct* não é ótima *em espaço* – note que um nodo com um ou nenhum filho, por exemplo, ainda ocupa $\Theta(|\Sigma|)$ espaço, o que leva a um desperdício de memória nestes casos.

   (a) apresente uma *struct* alternativa para um nodo de uma trie, otimizando o melhor que conseguir sua complexidade *de espaço*. Justifique informalmente (com suas palavras) porque sua *struct* é boa neste quesito;

   (b) apresente em pseudo-código o algoritmo de *busca* de uma chave em uma trie que utiliza sua *struct*, e analise sua complexidade de tempo de pior caso.

3. (20 pontos) O programa `geradorC.c`, em anexo à prova, recebe um número de sete dígitos e gera cinco inteiros distintos (não importa como). Usando o seu RA como entrada do programa, sejam A, B, C, D e E os cinco inteiros que ele gera. Considere a seguinte árvore B+ de ordem 5:



   (a) Insira A e apresente a árvore resultante. Indique o número de leituras e de escritas realizadas;

   (b) Na árvore obtida no item anterior, insira B e apresente a árvore resultante. Indique o número de leituras e de escritas realizadas;

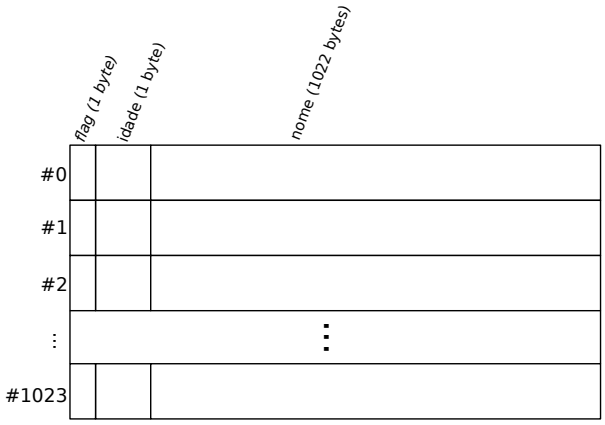
   (c) Na árvore obtida no item anterior, insira C e apresente a árvore resultante. Indique o número de leituras e de escritas realizadas;

   (d) Da árvore obtida no item anterior, liste todas as chaves entre D e E, inclusive. Indique o número de leituras e escritas realizadas;

   Em toda a questão, assuma que a raiz está em disco.

4. (20 pontos) Considere uma tabela com campos de idade (inteiro) e nome (*string*). Escreva um programa em C que gera em disco um *arquivo direto* (binário, chamado `pessoas.dat`) contendo registros desta tabela.

   - O arquivo gerado deve ter capacidade para até 1024 registros. Cada registro deve ocupar exatamente 1024 bytes (1 Kb), sendo 1 byte para a *flag* (`0x00` para registro inválido, `0x01` para registro válido), 1 byte para a idade, e 1022 bytes para o nome. Note que o arquivo gerado deve ter exatamente $1024 \times 1\text{Kb} = 1$ Mb de tamanho. A figura abaixo apresenta um esquema de como o arquivo deve ser organizado:



   - O programa deve criar um arquivo inicialmente sem registros, ler do usuário um inteiro N seguido de uma sequência de N registros, e inserí-los no arquivo na ordem dada na entrada. Como exemplo, se a entrada for

     ```
     3
     10 Ash
     12 Brock
     11 Misty
     ```

     o arquivo gerado deve conter (apenas) os registros $\{(10, Ash), (12, Brock), (11, Misty)\}$. Você pode assumir que nenhum nome conterá espaços nem terá mais de 1000 caracteres, e também que $N \leq 1024$;
   - Escolha como função *hash* a função que julgar ser a mais adequada. *Justifique sua escolha* na saída padrão do programa (`printf("Escolhi esta funcao hash porque ... ");`). Sua função deve "levar em conta" os dois campos do registro, e não apenas um;
   - Resolva colisões com sondagem linear;
   - Faça leituras e escritas no arquivo a cada registro informado pelo usuário. **Não** construa uma matriz em memória principal para apenas escrevê-la inteira no disco ao final do programa!
   - Dicas:
     - abra o arquivo com `r+b` (`fopen("pessoas.dat", "r+b")`) para simultaneamente usar `fread` e `fwrite` no mesmo arquivo binário;
     - `fread` e `fwrite` compartilham do mesmo "cursor" no arquivo (desta forma, por exemplo, se a função `fwrite` escrever 1 byte no início do arquivo e a função `fread` for chamada em seguida para ler um byte, o segundo byte será lido por ela, e não o primeiro);
     - a função `fseek(f, B, SEEK_SET)` move o "cursor" para (de forma que a próxima chamada de `fread` ou `fwrite` será feita a partir de) a posição B bytes após o início do arquivo; em particular, como cada registro ocupa 1024 bytes, chamar `fseek(f, 1024*i, SEEK_SET)` coloca o "cursor" no início do $i-$ésimo registro do arquivo;
     - Nesta questão, não se preocupe com *big* ou *little-endian*. Você pode, se quiser, ler/escrever mais de 1 byte de "uma só vez" com `fread`/`fwrite`.

5. (15 pontos) Para ajudar a combater a pandemia em Toledo, o COE está reunindo informações a respeito das pessoas infectadas na cidade, em uma tabela no seguinte formato:

ID	**Nome**	**Data de Infecção**	**Curado**
210	Carlitos Alberto	20/05/2020	Sim
124	Mirian Eduarda	22/05/2020	Sim
042	Bruno Parker	04/07/2020	Não
157	Joaquina Conceição	01/06/2020	Sim

Todas as consultas que o COE realiza nesta tabela são do tipo "Dada uma data D, quais pessoas foram infectadas no dia D?".

(a) Qual dos tipos de arquivos estudados é o melhor para armazenar esta tabela? Justifique.

(b) Apresente um esquema do arquivo citado no item anterior contendo a tabela dada acima.