Algoritmos e Estruturas de Dados LC 2017/2

Lista de Exercícios para VA 2

Rodrigo de Souza

Setembro 2017

1 Instruções

A presente lista L compõe a nota da VA 2 juntamente de dois EP's, *EP3* e *EP4*. A nota final é a média simples entre L, *EP3* e *EP4*.

A maior parte dos exercícios foi tirada do material de P. Feofiloff; outros são meus. Mas não vou mencionar a autoria em cada um deles.

Vocês devem fazer um único documento, em formato pdf, com cabeçalho ou capa, trazendo como título "Lista de Exercícios para VA 2", indicando também disciplina, semestre, seu curso, seu nome. Em cada questão, não precisa reproduzir o enunciado; basta escrever o número e sua resposta. Deixe margens grandes e espaço entre as questões para a correção. Vocês devem submeter o arquivo no AVA, em local específico que vou abrir lá.

Vocês podem usar qualquer editor de textos para fazer as respostas. O melhor mesmo é usar LaTeX, é uma boa aproveitar a oportunidade para aprender. No limite, vocês podem resolver no caderno, à mão, depois tirar foto e colar no Word. Mas só faça assim se não tiver condições de usar um editor, e mesmo assim certifique-se de que o resultado é legível.

As questões que pedem código podem ser feitas de duas formas: algum pseudocódigo bem claro, simples, sem ambiguidades ou funções esdrúxulas cujo significado é duvidoso, ou em C. Em ambos os casos, descreva claramente o que as funções que você vai escrever recebem e o que fazem ou devolvem.

2 Exercícios

1. Considere a implementação circular de uma fila em um vetor. Escreva o código das funções `colocanafila`, `tiradafila`, `filavazia` e `filacheia`. Escreva uma função que devolva o comprimento (ou seja, o número de elementos) da fila.

2. Considere o algoritmo de Dijkstra para construção do vetor de distâncias em um grafo a partir de um vértice s. Digamos que desejamos saber não somente as distâncias, mas também um caminho mínimo de s a cada um dos demais vértices. Descreva uma estrutura de dados que represente esse conjunto de caminhos. Descreva a construção dessa estrutura (dica: construa a estrutura durante a execução do algoritmo de Dijkstra, pense em algum dado que pode ser atribuído a cada vértice no momento em que for descoberto, e que permita a recuperação desses caminhos).

3. Uma fila dupla (= *deque*, pronuncia-se *deck*) permite inserção e remoção em qualquer das duas extremidades da fila. Implemente uma fila dupla (em um vetor ou uma lista encadeada) e codifique as funções de manipulação da estrutura.

4. Descreva uma função que recebe duas listas ligadas, com cabeça, cada uma representando uma sequência *crescente* de números inteiros, e constrói uma nova lista, contendo os inteiros de ambas as listas intercalados em ordem crescente (semelhante ao que se faz na etapa de intercalação do `Mergesort`). Essa nova lista deve aproveitar os nós das listas recebidas (que portanto deixam de existir individualmente).

5. Escreva uma função que troque de posição duas células de uma mesma lista encadeada. Faça duas versões: uma para lista simples, outra para listas duplamente encadeadas.

6. Suponha dado um conjunto $A_1, A_2, \ldots, A_n$ de vetores numéricos crescentes. (Você pode trocar "vetor" por "lista encadeada" ou por "arquivo" se desejar.) Digamos que cada A_i tem t_i elementos. Queremos imprimir todos os $t = t_1 + t_2 + \ldots + t_n$ números, em ordem decrescente. Dê um algoritmo que consuma tempo $\mathcal{O}(t \lg n)$ para fazer o serviço. Dica: use filas de prioridade.

7. Digamos que h é a altura e p é a profundidade de um nó x em uma árvore binária. É verdade que $h + p$ é igual à altura da árvore? Justifique ou dê um contra-exemplo.

8. Considere o seguinte problema sobre uma árvore binária: encontrar o endereço do primeiro nó da árvore na ordem e-r-d. É claro que o problema só faz sentido se a árvore não é vazia. Eis uma função que resolve o problema:

```
// Recebe uma árvore binária não vazia r
// e devolve o primeiro nó da árvore
// na ordem e-r-d.

noh *primeiro (arvore r) {
    while (r->esq != NULL)
       r = r->esq;
    return r;
}
```

   Escreva uma versão recursiva dessa função.

9. Escreva uma função que encontre o último nó na ordem e-r-d.

10. Escreva uma função que receba o endereço de um nó x de uma árvore binária e encontre o endereço do nó anterior a x na ordem e-r-d. Você pode supor que cada nó da árvore tem um campo `pai` que representa o pai desse nó.

11. Descreva uma função que recebe uma expressão aritmética na notação posfixa e constrói a árvore binária representando essa expressão. Sugestão: use uma pilha representando "pedaços" da árvore, e proceda de forma semelhante ao cálculo do valor da expressão.

12. Escreva uma função que decida se uma dada árvore binária é ou não é de busca.

13. Escreva uma função que transforme um vetor crescente em uma árvore binária de busca que seja balanceada.

14. Suponha que as chaves 50 30 70 20 40 60 80 15 25 35 45 36 são inseridas, nesta ordem, numa árvore de busca inicialmente vazia. Desenhe a árvore que resulta.

15. Uma árvore binária quase completa com campo pai é hierárquica se

    ```
    x->conteudo >= x->esq->conteudo
    ```

    e

    ```
    x->conteudo >= x->dir->conteudo
    ```

    para todo nó x (desde que os filhos existam). Escreva uma implementação de fila de prioridades baseada em árvore hierárquica. A implementação deve ter funções para criar uma fila vazia, retirar um elemento máximo da fila, e inserir um elemento na fila. (Dica: veja a implementação de fila de prioridades baseada em heap.)

16. O fator de balanceamento de um nó em uma árvore binária é a diferença entre as alturas da subárvore à esquerda e a subárvore à direita. Suponha que cada nó tem um campo `fb` destinado a armazenar esse valor. Apresente um algoritmo recursivo que preenche esse campo de cada nó.

17. Apresente um algoritmo para transformar uma árvore binária de busca em uma nova árvore de busca, mas *balanceada*. Seu algoritmo deve ter complexidade $\mathcal{O}(n)$, onde n é o número de chaves armazenadas. Dica: transforme a árvore original em um vetor em ordem crescente.

18. Resolva o problem da contagem para o fluxo de chaves 17 21 19 4 26 30 37 usando hashing com encadeamento. A tabela de dispersão deve ter tamanho 13 e a função de espalhamento deve ser o resto da divisão da chave por M. Faça uma figura do estado final da tabela de dispersão. Repita para sondagem linear.

19. Escreva um programa que leia um arquivo de números inteiros positivos, elimine os números repetidos, e grave uma versão do arquivo sem os repetidos. Não altere a ordem relativa dos números. O seu programa deve ter caráter de filtro, ou seja, deve gravar o resultado à medida que o arquivo de entrada for sendo lido. Qual é a complexidade de sua solução, em função da quantidade de números já lidos?

20. Neste exercício estamos interessados em implementar um sistema que fornece em tempo real dados sobre número de ocorrências de palavras em um texto. Seu sistema recebe um fluxo de palavras e, a cada instante, fornece as seguintes operações:

- Impressão da lista de palavras já recebidas, com a quantidade de ocorrências de cada uma;
- Dado um inteiro p, impressão da lista de palavras que ocorrem exatamente p vezes.

Dizer que seu sistema funciona em tempo real (*online*) significa que o sistema atualiza a cada palavra lida suas estruturas de dados de forma a permitir que essas operações sejam realizadas eficientemente. Claro que a escolha da estrutura de dados tem impacto direto na eficiência de sua solução. Descreva detalhadamente sua solução, ou seja, as estruturas e manipulações necessárias das mesmas.