FACULDADE CAMPO LIMPO PAULISTA
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

Projeto e Análise de Algoritmos II
Lista de Exercícios 2

Prof. Osvaldo.

1. Desenvolva algoritmos para as operações abaixo e calcule a complexidade de tempo (pior caso) para os mesmos. Use um vetor como estrutura de dados.
   a) inserir um elemento em uma dada posição de um vetor.
   b) inserir *n* elementos em um vetor (por exemplo, lendo um a um a partir de uma entrada padrão).
   c) inserir um o elemento *x* em um vetor ordenado, mantendo a ordenação. Isto é eficiente?

2. Uma pilha é uma estrutura de dados que que admite operações de inserção e remoção segundo uma disciplina *LIFO* (*last input, first output*), i.e., o último elemento a entrar na estrutura é o primeiro a sair. Explique como se pode implementar duas pilhas em um vetor [1 .. *n*] de tal modo que nenhuma pilha "transborde" a menos que o número total de elementos nas pilhas seja igual a *n*. As operações de inserção e remoção devem ter complexidade de tempo (pior caso) igual a $O(1)$.

3. Mostre como implementar uma fila usando duas pilhas. Analise a complexidade de tempo (pior caso) das operações de inserção e remoção na fila implementada desta forma.

4. Calcule a complexidade de tempo (pior caso) para as seguintes operações realizadas sobre uma lista ligada simples:
   a) inserir um elemento na última posição da lista.
   b) remover o elemento da primeira posição da lista.
   c) remover o elemento de valor igual a *x* da lista.

5. Uma lista ligada circular é uma lista ligada em que os elementos podem ser percorridos circularmente. Desenvolva algoritmos para as operações de inserção, remoção e busca de um elemento em uma lista circular. Quais são as complexidades de tempo (pior caso) destes algoritmos?

6. Escreva um algoritmo para inverter a ordem dos elementos de uma lista ligada simples. Calcule a complexidade de tempo (pior caso) do seu algoritmo.

7. Uma palavra é um **palíndromo** se a seqüência de letras que a forma é a mesma, quer seja lida da esquerda para a direita ou da direita para a esquerda (como exemplos: ovo, raiar e osso). Desenvolva uma estrutura de dados baseada em listas e escreva um algoritmo $O(n)$ para resolver o problema de testar se uma palavra é um palíndromo.

8. Projete uma estrutura de dados dinâmica (listas) para representar Pilhas. A sua estrutura de dados deve permitir implementar em O (1) os algoritmos para empilhar e desempilhar.

9. Projete uma estrutura de dados dinâmica (listas) para representar Filas. A sua estrutura de dados deve permitir implementar em O (1) os algoritmos para inserir e retirar da fila.

10. Seja A uma **matriz esparsa** n x m, isto é, boa parte dos seus elementos são nulos ou irrelevantes. Projete uma estrutura de dados que represente A e cujo espaço total seja O (k) em vez de O (n m), onde k é o número total de elementos não irrelevantes de A.

11. Desenvolva um algoritmo para localizar um elemento a_{ij} da matriz A, armazenada segundo a estrutura de dados obtida na solução do exercício anterior. Determine a sua complexidade.

Um *pool* é uma estrutura de dados que aceita as seguintes operações:
- Inserção (x): insere um elemento x na estrutura de dados;
- Remoção: remove algum elemento da estrutura de dados.

Os três exercícios a seguir referem-se à estrutura de dados *pool*.

12. Projete os algoritmos que implementam a estrutura de dados *pool* de tal forma que as operações de inserção e remoção tenham complexidade de tempo (pior caso) igual a O (1). Admita que a estrutura de dados pode conter elementos duplicados.

13. Modifique a implementação da estrutura de dados *pool* da seguinte forma: assuma que todo elemento pode aparecer no máximo uma vez na estrutura de dados. A operação de inserção deve agora verificar a existência de duplicatas. Havendo uma duplicata, o elemento não deve ser inserido. Qual a complexidade de tempo (pior caso) de cada operação?

14. Uma outra variante da implementação da estrutura de dados *pool* é a seguinte: assuma que todos os elementos são identificados por inteiros no intervalo de 1 a n, e n é pequeno o suficiente para se poder alocar memória de tamanho O (n). Cada elemento pode aparecer no máximo uma vez. Projete algoritmos para as operações de inserção e remoção que trabalhem com complexidade de tempo (pior caso) igual a O (1).

15. Uma operação de união toma dois conjuntos disjuntos S_1 e S_2 como entrada e retorna um conjunto $S = S_1 \cup S_2$. Os conjuntos S_1 e S_2 são usualmente destruídos na operação. Mostre como a operação de união pode ser desenvolvida tendo uma complexidade de tempo igual a O (1).

16. Desenhe árvores binárias de busca inserindo na ordem em que aparecem os seguintes elementos:
    a) 30 15 40 7 8 38 45 18.
    b) 7 8 15 18 30 38 40 45.

17. O percurso de uma árvore em *preorder* resultou na impressão da seqüência A B C F H D L M P N E G I, e o percurso da mesma árvore em *inorder* resultou em F C H B D L P M N A I G E. Construa uma árvore que satisfaça esses percursos. Ela é única?

18. (ENADE 2011): Suponha que se queira pesquisar a chave 287 em uma árvore binária de busca com chaves entre 1 e 1 000. Durante uma pesquisa como essa, uma seqüência de chaves é examinada. Cada seqüência abaixo é uma suposta seqüência de chaves examinadas em uma busca da chave 287.

    I. 7, 342, 199, 201, 310, 258, 287
    II. 110, 132, 133, 156, 289, 288, 287
    III. 252, 266, 271, 294, 295, 289, 287
    IV. 715, 112, 530, 249, 406, 234, 287

    É válido apenas o que se apresenta em
    A) I.
    B) III.
    C) I e II.
    D) II e IV.
    E) III e IV.

19. Desenvolver e calcular as complexidades dos seguintes algoritmos sobre árvores binárias de busca:
    a) Mínimo: retorna um ponteiro para o menor elemento da árvore.
    b) Máximo: retorna um ponteiro para o menor elemento da árvore.
    c) Sucessor: retorna um ponteiro para o sucessor de um nó apontado por um ponteiro, digamos *p*.

20. Escreva um algoritmo para retornar a soma dos elementos de uma árvore binária. Qual a complexidade do seu algoritmo?

21. (ENADE 2011) Listas ordenadas implementadas com vetores são estruturas de dados adequadas para a busca binária, mas possuem o inconveniente de exigirem custo computacional de ordem linear para a inserção de novos elementos. Se as operações de inserção ou remoção de elementos forem freqüentes, uma alternativa é transformar a lista em uma árvore binária de busca balanceada, que permitirá a

execução dessas operações com custo logarítmico. Considerando essas informações, escreva um algoritmo recursivo que construa uma árvore binária de busca completa, implementada por apontadores, a partir de um vetor ordenado, V, de *n* inteiros, em que $n = 2^m - 1$, $m > 0$. O algoritmo deve construir a árvore em tempo linear, sem precisar fazer qualquer comparação entre os elementos do vetor, uma vez que este já está ordenado. Para isso,

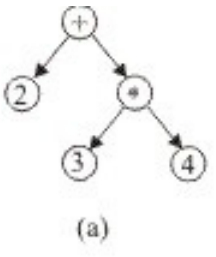
a) descreva a estrutura de dados utilizada para a implementação da árvore.

b) escreva o algoritmo para a construção da árvore. A chamada principal à função recursiva deve passar, como parâmetros, o vetor, índice do primeiro e último elementos, retornando a referência ou apontador para a raiz da árvore criada.

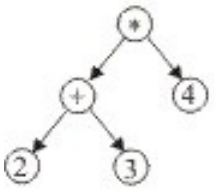
22. Mostre que se um nó em uma árvore binária de busca tem dois filhos, então o seu sucessor não possui filho à esquerda e o seu predecessor não possui filho à direita.

23. Escreva um algoritmo que receba como entrada um ponteiro *y* para um nó e retorne um ponteiro para o pai deste nó em uma árvore binária de busca. O seu algoritmo deve ter complexidade de tempo igual a $O(h)$, onde h é altura da árvore.

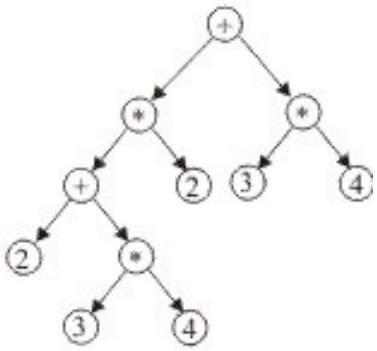
24. Árvores binárias podem ser utilizadas para representar expressões aritméticas identificando, sem ambigüidade, a ordem em que as sub-expressões devem ser avaliadas. Por exemplo, as árvores das figuras abaixo representam as expressões: a) 2 + 3*4; b) (2 + 3)*4 e c) (2 + 3*4)*2 + 3*4. Desenvolva um algoritmo que receba como entrada um ponteiro para a raiz de uma árvore binária, como descrevemos, e retorne a avaliação da expressão, isto é, o algoritmo deverá retornar os valores 14, 20 e 40 para as árvores das figuras (a), (b) e (c), respectivamente. Admita que as expressões conterão somente os operadores de adição e multiplicação. Calcule a complexidade do seu algoritmo.



(a)



(b)



(c)

25. Ilustre (desenho) a operação Inserir (*A*, 12, 3) em um *heap A* = [15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1]. Represente o *heap* na sua ilustração como uma árvore com ponteiros.

26. - Ilustre (desenho) a operação Remover (*A*, 12) em um *heap A* = [15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1]. Represente o *heap* na sua ilustração como uma árvore com ponteiros.

27. Qual a diferença entre as propriedades de um *heap* e de uma árvore binária de busca? É possível imprimir em ordem crescente todos os nós de um *heap* com complexidade de tempo igual a $O(n)$?

28. Mostre como um *heap* pode ser utilizado para criar um algoritmo de ordenação $O(n \log n)$.

29. Descreva como implementar uma fila com disciplina FIFO (*first input*, *first output*) por meio de um *heap*. Qual a complexidade das operações de inserir e retirar da fila neste caso?

30. Ilustre a inserção dos elementos {5, 28, 19, 15, 20, 33, 12, 17, 10} em uma tabela de *hashing* com colisões resolvidas por encadeamento. Considere uma tabela com 9 *slots*, numerados de 0 (zero) a 8, e tome $h(x) = x$ **mod** 9 como função de *hashing*.

31. Desenvolva um algoritmo de complexidade $O(n \log k)$ para intercalar k listas ordenadas em uma única lista, onde n é o número total de elementos em todas as listas de entrada. (Sugestão: use um *hea*p para ajudar na tarefa de intercalar).

32. Projete (apenas diga como será, não é necessário escrever os algoritmos) uma estrutura de dados que suporte as seguintes operações em tempo, no pior caso, igual a $O(\log n)$, onde n é o número de elementos na estrutura de dados:
    - Inserir (x): insere o elemento x na estrutura de dados somente se ele não existir.
    - Remover (x): remove o elemento de valor igual a x da estrutura de dados (se ele existir).
    - Sucessor (x): retornar o menor valor na estrutura de dados que é maior do que x.

    Argumente por que a sua estrutura de dados resolve o problema.