EXERCÍCIOS SOBRE ÁRVORES/ORDENAÇÃO

1. Implemente o ABB estática (vetor) e dinâmica (encadeada) com as operações de construção, destruição, inserção, busca, remoção, reinicialização, percursos (em-ordem, pré-ordem, pós-ordem, largura), contagem de folhas e semi-folhas, determinação da altura da árvore, etc…

2. Realize as mesmas implementações, da questão anterior, aplicando apenas estratégias recursivas.

3. Considerando as funções abaixo *main( )*, *funcY ( ) e funZ ( )* e que NoABB corresponde ao nó raiz de uma árvore binária qualquer. Execute um teste de mesa sobre a ABB na Figura 1 e identifique o significado estrutural do valor exibido pelo *printf* na função *main*?

```
int main( )
{  descABB *p;
   … criaABB(&p,...)
   …
   printf("%i",
   funcY(p));
   ...
}
```

```
int funcY (descABB *pt)
{ int c=0, m=0;
   if (pt->raiz == NULL )
         return 0;
  else
        funcZ(p->raiz,&c,&m);
  return m;
}
```

```
int funcZ(NoABB *no, int *C, int *M)
{ if (no==NULL)
return 0;
   else
   {   *C=*C+1;
       if (*C > *M)
                *M=*C;
       *C=funcZ(no -> esq,C,M)+*C;
       *C=funcZ(no -> dir,C,M)+*C;
        return -1;
    }
 }
```

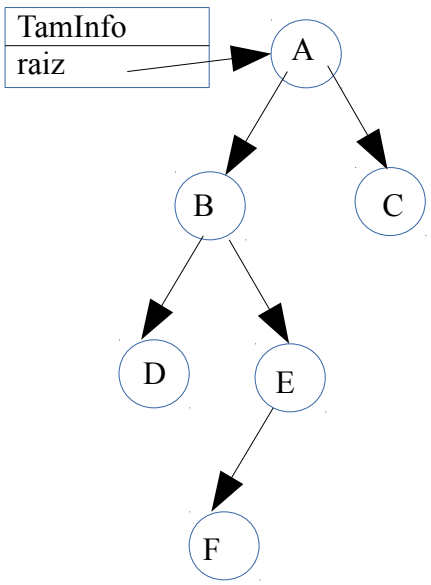

Figura 1: Árvore Binária para teste de mesa.

4. Uma ABB completa é uma árvore perfeitamente balanceada onde todos os nós, exceto as folhas, possuem os dois filhos. Essa árvore com ‘n’ folhas tem altura igual a log_2^n.

   No pior caso, com ausência total de balanceamento, quando há uma degeneração da árvore para a forma de uma lista encadeada, uma ABB poderá ter altura proporcional ao número total de nós: *m*.

   Explique como se chega a essas duas ordens de grandeza para as respectivas alturas citadas.

5. A análise de complexidade de um *algoritmo* busca estimar quanto tempo (e eventualmente quanta memória) esse *algoritmo* gasta de acordo com o tamanho de sua entrada. Dizemos que o tempo de execução é "Big-O de g(n)" significando que a partir de uma constante c e uma entrada de tamanho n_0, o limite superior da performance do algoritmo será de ordem g(n), observe exemplo na Figura 2.

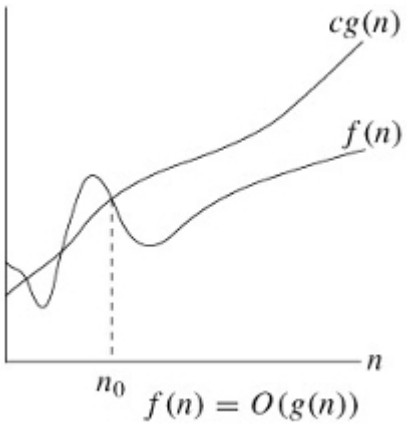

Figura 2: Representação gráfica para a notação Big-O.

   No caso da ABB-AVL o tempo de execução de uma busca será $O(log_2^n)$. Com isso, queremos dizer que log_2^n é o limite superior do tempo de busca na ABB-AVL:

   > “o tempo de execução da busca cresce no máximo nessa ordem, porém, eventualmente pode ser mais rápido”.

   a) Qual a utilidade da análise teórica de algoritmos para a comparação de soluções? É necessário implementar algoritmos para compará-los?

   b) Porquê a complexidade de execução da operação de busca sobre ABB depende diretamente da altura da árvore?

6. Execute:
   a) A inserção sequencial das seguintes chaves em uma ABB e em uma ABB-AVL:
      225, 200,100, 290, 250, 500, 900, 700, 300, 220, 215, 800, 600 e 295

   b) Para a ABB e ABB-AVL obtidas no item anterior execute as remoções das seguintes chaves em sequência: 100, 300, 250, 290 e 225. Discuta as consequências da utilização do substituto pelo sucessor e pelo antecessor.

   c) Para a ABB obtida no item anterior: percorra em pós-ordem a ABB e, para cada nó visitado, insira a respectiva chave em uma nova ABB-AVL. Faça o mesmo para os percursos pré-ordem, em-ordem e em largura.

7. Resolva os exercícios de inserção/remoção contidos no arquivo “AVL_WIRTH.pdf” na pasta “Exercícios do Livro do Niklaus Wirth” no Moodle. Discuta as consequências da utilização do substituto pelo nó sucessor e pelo antecessor em ordem.

8. Compare a aplicação da busca sequencial sobre a LDE_RefMov (cujo descritor possui referencial móvel para a lista) com a aplicação de busca sobre a ABB-AVL.

9. A Figura 3 representa uma AVL. Cada triângulo (A, B, C e D), por sua vez, representa uma subárvore completa (ou seja, todos os níveis preenchidos, de modo que qualquer inserção acarrete um acréscimo em sua altura). Sabe-se ainda que as alturas das subárvores A, B, C e D são, respectivamente, h+1, h, h e h+1.
   Pede-se:
   (a) Calcule o fator de balanceamento (diferença entre alturas de suas duas subárvores) para os nós x, y e z.
   (b) Desenhe a AVL novamente, supondo que ocorra uma inserção na subárvore A.
   (c) Desenhe a AVL novamente, supondo que ocorra uma inserção na subárvore B ou na subárvore C.

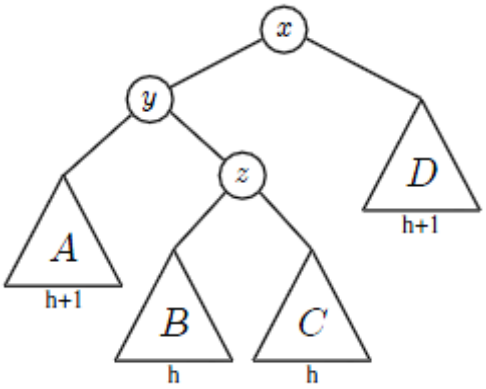

Figura 3: Exemplo de ABB-AVL.

10. Quais as características de uma árvore Multidirecional (n-ária)? Descreva uma B-tree.

11. Resolva os exercícios de inserção/remoção contidos no arquivo “B-Tree_WIRTH.pdf” na pasta “Exercícios do Livro do Niklaus Wirth” no Moodle.

12. Execute a ordenação das seguintes chaves utilizando: (a) Bubble-Sort (Bolha), (b) Quick-Sort, (c) Merge-Sort, (d) Seleção Direta e (e) Heap-Sort:
    225, 200, 100, 290, 250, 500, 900, 700, 300, 220, 215, 800, 600 e 295

13. Recorra à literatura e descreva as complexidades (tempo) dos métodos de ordenação citados anteriormente.

14. Implemente o TDA Arvore-Heap construída sobre um vetor e o algoritmo de ordenação Heap-Sort. Execute a ordenação das seguintes chaves utilizando o Heap-Sort:

    225, 200,100, 290, 250, 500, 900,7 00, 300, 220, 215, 800, 600 e 295

15. Implemente a fila de prioridade como uma aplicação sobre a árvore-Heap.

16. Discuta sobre outras implementações de árvores balanceadas. A árvore red-black é um exemplo?

Referencias:

[1] http://www.lcad.icmc.usp.br/~nonato/ED/AVL/node67.html;

[2] https://pt.khanacademy.org/computing/computer-science/algorithms/asymptotic-notation/a/big-o-notation

[3] Azeredo, P. A. Métodos de Classificação de Dados e Análise de suas Complexidades. Ed. Campus.

[4] Szwarcfiter, Jayme L. & Markenzon L. “Estruturas de Dados e Seus Algoritmos”. Rio de Janeiro. Ed. LTC. 1994.

[5] Tenembaum, Aaron M. e outros. “Estruturas de Dados Usando C”. São Paulo. Ed.Makron Books. 1995.

[6] Wirth, Niklaus. “Algorthms + Data structures = Programs”, Ed. Prentice Hall, 1976.

[7] Gersting, Judith L. “Fundamentos Matemáticos para a Ciência da computação”. Ed. LTC.

[8] Veloso, Paulo et al. “Estruturas de Dados”. Ed. Campus

Notas de aula e demais livros da bibliografia recomendada para a disciplina.