Lista 02

```
typedef struct no {
    int v;
    struct no *esq, *dir;
} No;
```

1. Escreva uma função que realiza um percurso em largura na árvore. Você pode usar uma fila auxiliar. `void largura(No* t);`

2. Escreva uma função que calcula o número de nós de uma árvore binária.

   `int conta_nos(No* t);`

3. Escreva uma função recursiva que conta o número de folhas em uma árvore binária.

   `int conta_folhas(No* t);`

4. Escreva uma função não recursiva para o percurso em-ordem de uma árvore binária. Dica: utilize uma pilha. Assuma que nao há mais do que 100 elementos em sua árvore

   `int n_rec_inorder(No* t);`

5. Escreva uma função recursiva que verifica se um valor v está presente na árvore t (considere que não há nenhuma garantia a respeito da ordem dos valores na árvore).

   `int busca(No* t, int v);`

6. Escreva uma função `espelho(t)` que retorna uma nova árvore `t`, mas com as sub-árvores esquerda e direita de **todos** os nós trocadas. A árvore original não deve ser alterada.

   `No* espelho(No* t);`

7. Em uma árvore de expressão, um nó que é um operando (uma letra entre `a` e `z`) não deve ter filhos e um nó que é um operador (`+`, `-`, `*` ou `/`) deve ter duas sub-árvores não vazias que correspondem a duas expressões válidas. Escreva uma função que verifica se uma árvore de expressão é válida, considerando também que uma árvore vazia é inválida e que a presença de quaisquer outros caracteres também tornam uma árvore inválida.

   `int expr_valida(ap_no t);`

8. Escreva a árvore binária correspondente à seguinte expressão: $(a+b)*(c-d)-e*f$. Escreva os elementos da árvore considerando os seguintes percursos pré-ordem e pós-ordem.

9. Desenhe a árvore binária que tem os percursos descritos abaixo.

   Pré-ordem: C E A D H K J B M F L G I

   In-ordem: D A K H E C B J L F M G I

   (a) Desenhe uma outra árvore que tenha o mesmo percurso em pré-ordem.

   (b) Descreva o percurso em pós-ordem e em-ordem das duas árvores.

```
typedef struct no {
    int chave;
    struct no *esq, *dir;
} No;
```

10. Escreva uma função **não-recursiva** que recebe uma árvore binária de busca `t` como parâmetro e retorna o apontador para o nó cuja chave possui o valor mínimo ou NULL caso a árvore esteja vazia.

    ```
    No* minimo(No* t);
    ```

11. Escreva uma função **não-recursiva** que verifica a existência de algum elemento com chave negativa na árvore.

    ```
    int existe_chave_negativa(No* t);
    ```

12. Considerando os algoritmos vistos em sala de aula, desenhe a árvore binária de busca resultante da inserção dos seguintes elementos (nesta ordem): $20, 25, 10, 5, 12, 22$ e 23. Remova a raiz da árvore. Quais elementos podem ser utilizados para substituir a raiz?

13. Suponha que as chaves 50, 30, 70, 20, 40, 60, 80, 15, 25, 35, 45, 36 são inseridas, nesta ordem, numa árvore de busca que está inicialmente vazia. Desenhe a árvore que resulta. Em seguida remova o nó que contém 30.

14. Considere que dado um vetor ordenado você precisa construir uma árvore de busca que contenha os mesmos elementos. Como você faria a construção da árvore para evitar que esta ficasse desbalanceada?

    ```
    No* vet2arv(int vet[ ], int n);
    ```

15. Seja `bal` o fator de balanceamento de um nó dado pela fórmula $h_d - h_e$, onde h_e é a altura da sub-árvore esquerda e h_d é a altura da sub-árvore direita. Escreva uma função recursiva que calcula o `bal` de todos os nós da árvore.

    ```
    typedef struct no {
        int v;
        int bal;
        struct no *esq, *dir;
    } No;

    void calcula_bal(No* t);
    ```

    Seria eficiente utilizar uma função auxiliar `altura` para calcular os fatores de balanceamento? Caso você ache que não, escreva uma solução mais eficiente.

16. Dada uma árvore com fatores de balanceamento calculados de acordo com a definição anterior, escreva uma função recursiva que retorna o valor máximo de **bal** na árvore.

    ```
    int max_bal(No* t);
    ```

17. Desenhe a árvore AVL resultantes da inserção dos seguintes elementos (nesta ordem): 30, 10, 20, 40, 50, 35 e 5. Coloque o fator de balanceamento e em caso de rotação dupla, desenhe os dois passos.

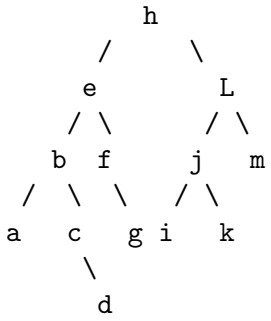
18. Em cada opção abaixo, insira as chaves na ordem mostrada de forma a construir uma árvore AVL. Se houver rebalanceamento de nós, mostre qual(is) transformações será realizada.

    (a) a,z,b,y,c,x
    (b) a,z,b,y,c,x,d,w,e,v,f
    (c) a,v,l,t,r,e,i,o,k
    (d) m,te,a,z,g,p

19. Refaca o exercício anterior, removendo as chaves na ordem FIFO (First In, First OUT). Ou seja, a primeira chave inserida, e' a primeira a ser removida. Indique, quando houver, as operacoes de balanceamento.

20. Seja a árvore AVL dada abaixo. Remova cada uma das chaves da lista abaixo. A cada remoção, recomece pela árvore original..

    (a) k
    (b) c
    (c) j
    (d) a
    (e) g
    (f) m
    (g) h



21. Escreva uma função que retorna a altura da árvore AVL percorrendo somente um único caminho da raiz até a folha.