Universidade Estadual de Santa Cruz
Disciplina: Estrutura de Dados
Professora: Jacqueline Midlej
Data: 14/12/2023
Nome: _____
Nº matrícula: _____

Prova 2 - GABARITO

Para as questões Q1 a Q3, utilize como elementos de um nó da árvore apenas os atributos:

- **valor**
- **ponteiro para nó esquerdo**
- **ponteiro para nó direito**

Q1. (1.5 ponto) Implemente uma função que retorne a soma das chaves de todos os nós intermediários (não folhas) de árvore binária. Não use variáveis globais. Pode ser implementada em C ou Python.

```
def soma (raiz:  no)-> int:
        if raiz is None:
                return 0
        elif raiz.esq is None and raiz.dir is None:
                return 0
        else:
                return soma(raiz.dir)+soma(raiz.esq)+raiz.valor
```

Q2.(0.5 ponto) Escreva uma função que retorna o valor máximo em uma **árvore binária de busca**. Não use variáveis globais. Pode ser implementada em C ou Python.

```
def max (raiz: no) -> int:
        while raiz is not None and raiz.dir is not None:
                raiz=raiz.dir
        return raiz.valor
```

Q3.(0.5 ponto) Escreva uma função que retorna o valor máximo em uma **árvore binária arbitrária**. Não use variáveis globais. Pode ser implementada em C ou Python.

```
def maximo (raiz: no)->int:
        if raiz is None:
                return 0
        v_esq=maximo(raiz.esq)
        v_dir=maximo(raiz.dir)
        v_raiz=raiz.valor

        return max([v_esq, v_dir, v_raiz])
```

ou

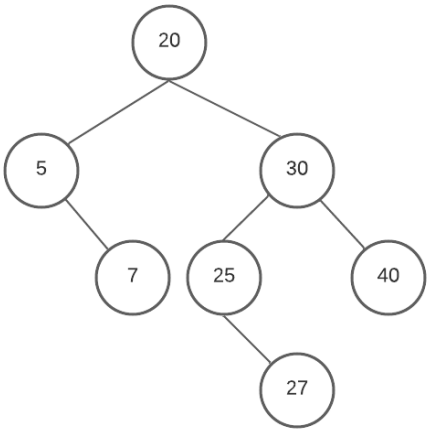
```
def maximo (raiz: no)->int:
        if raiz is None:
                return 0
        v_esq=maximo(raiz.esq)
        v_dir=maximo(raiz.dir)
        v_raiz=raiz.valor

        if v_esq>v_dir and v_esq>v_raiz:
                return v_esq
        elif v_dir>v_esq and v_dir>v_raiz:
                return v_dir
```

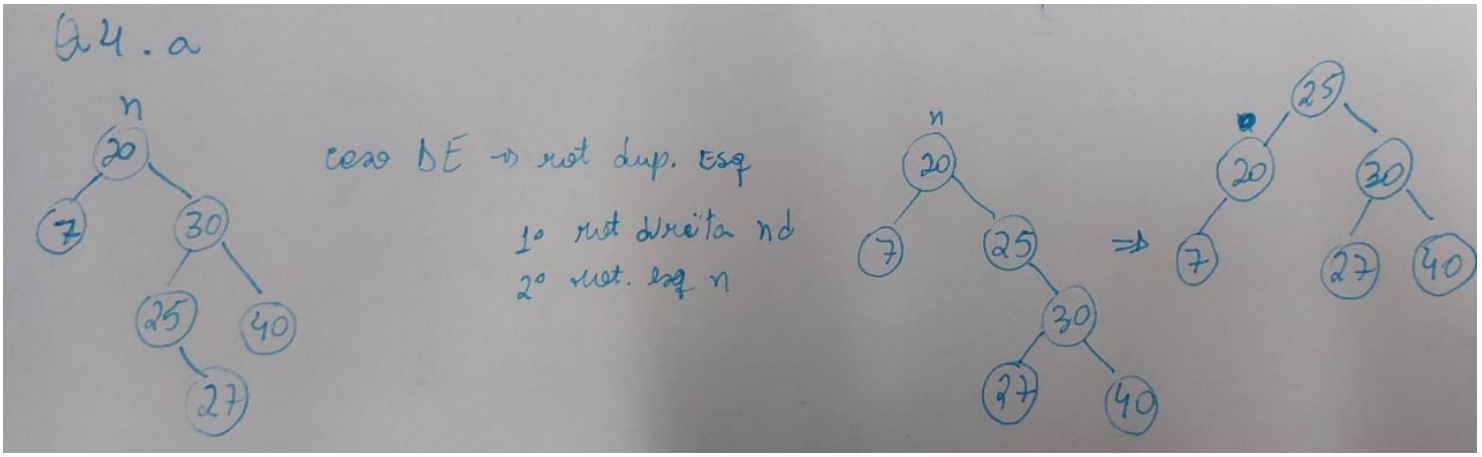
```
else:
        return v_raiz
```

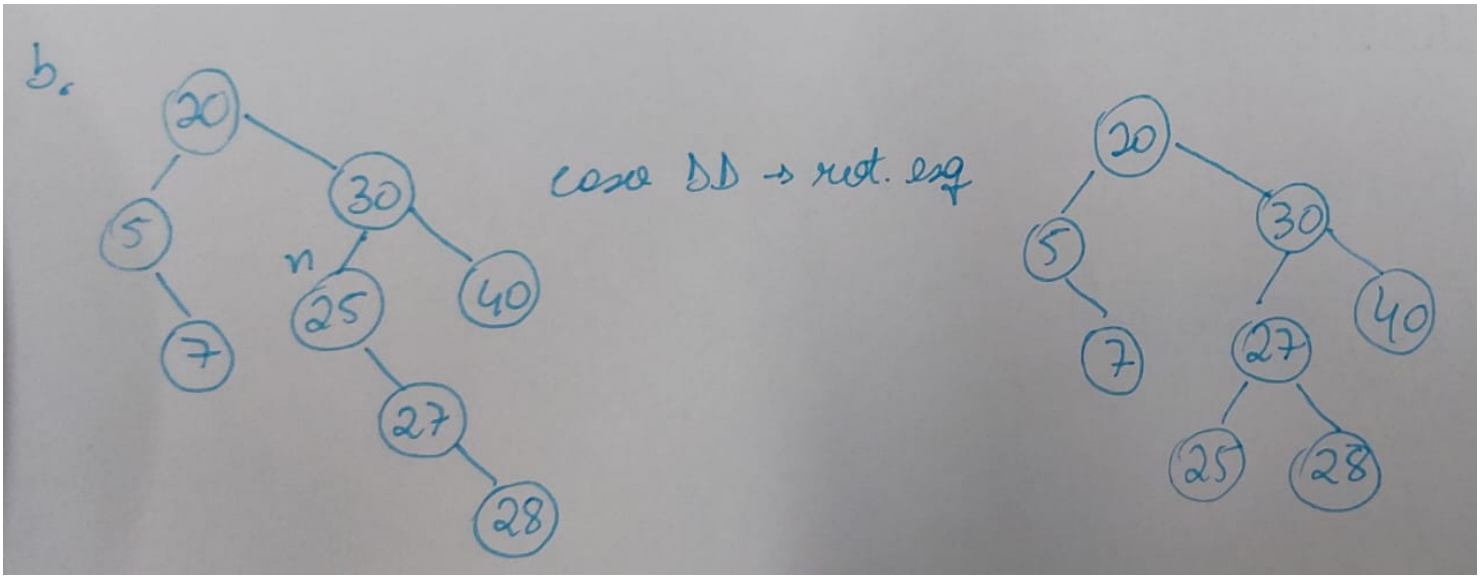
Q4. (2.0) Para a seguinte **árvore AVL,** faça o procedimento que se pede, e mostre o **resultado final**. Se houver necessidade de rotação, aplique e **indique qual rotação** foi utilizada. OBS: nas remoções usar **sucessor**, caso precise.

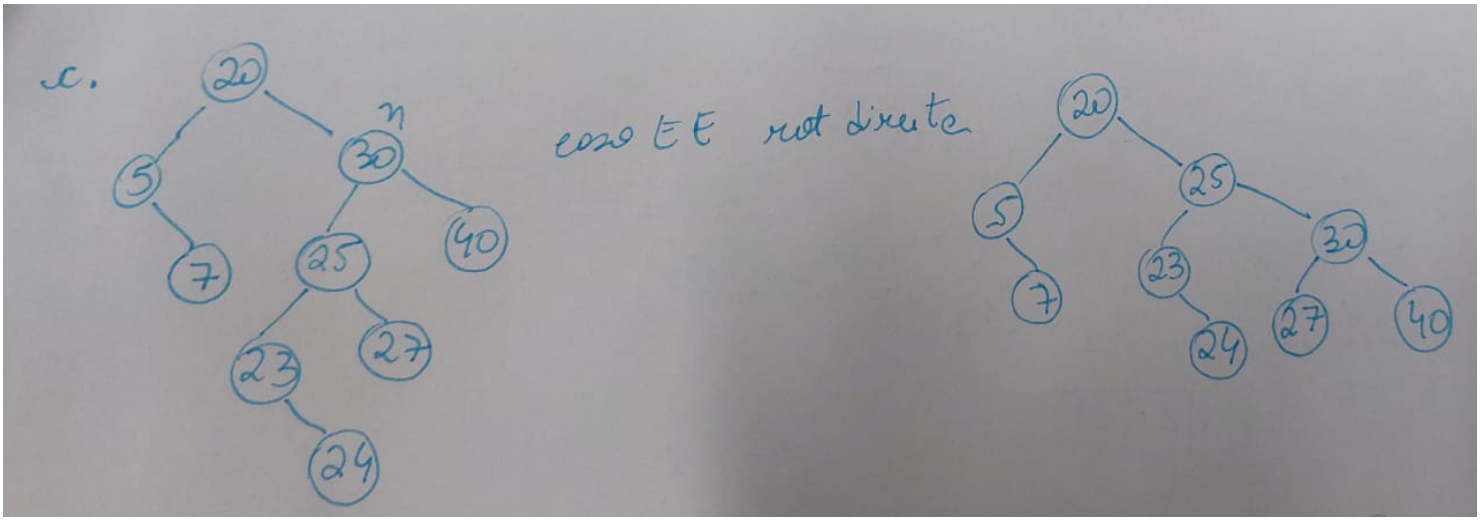
Árvore inicial

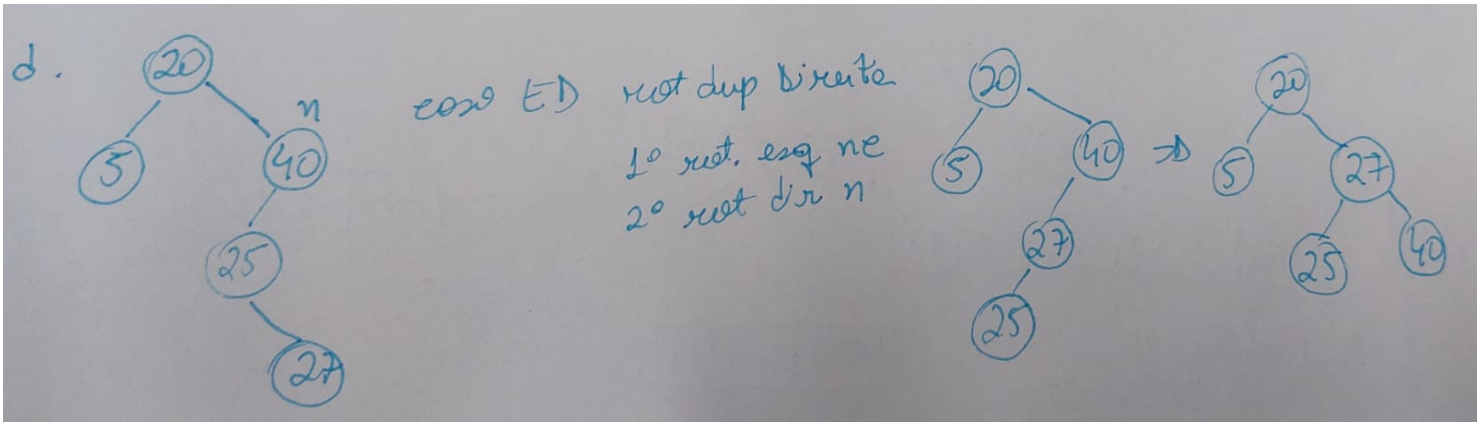


a) A partir da árvore inicial, remover 5
b) A partir da árvore inicial, inserir 28
c) A partir da árvore inicial, fazer: Inserir 23; inserir 24; (nesta ordem)
d) A partir da árvore inicial, remover 30



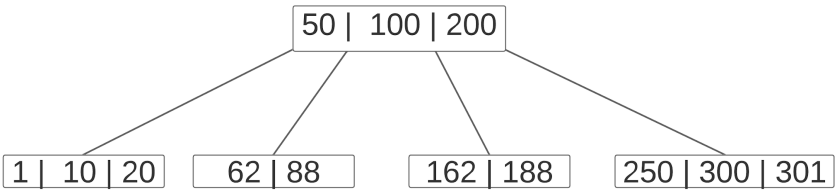






na d, esqueci de desenha o nó 7. Ele nao muda. Permanece no mesmo lugar em todas as árvores, a direita do 5.

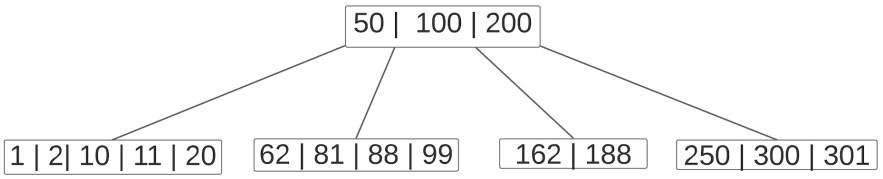
Q5. (1.0) Seja a seguinte **árvore B** com **t=3**. Faça as inserções a seguir, e **mostre o resultado final**.



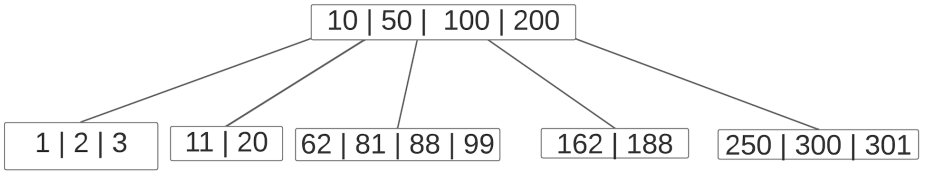
a) Inserir nessa ordem, os elementos: 2, 99, 81, 11, 3, 24, 49, 98, 51, 500, 150
b) A partir do resultado da letra a, inserir nesta ordem, os elementos: 77, 400, 600, 601

min=3-1=2
max=2*3-1=5
a)

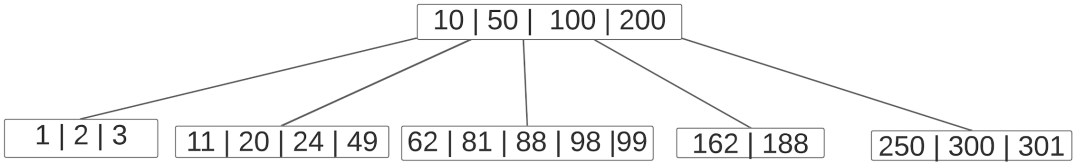
add: 2, 99, 81, 11



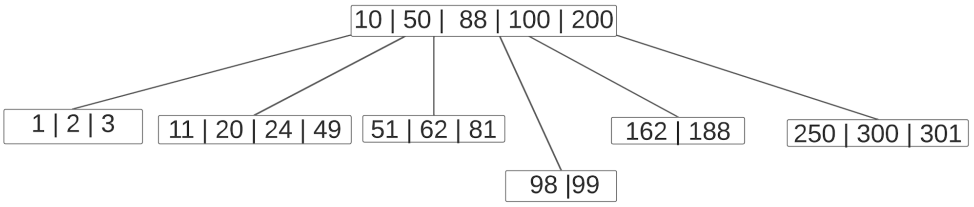
add: 3



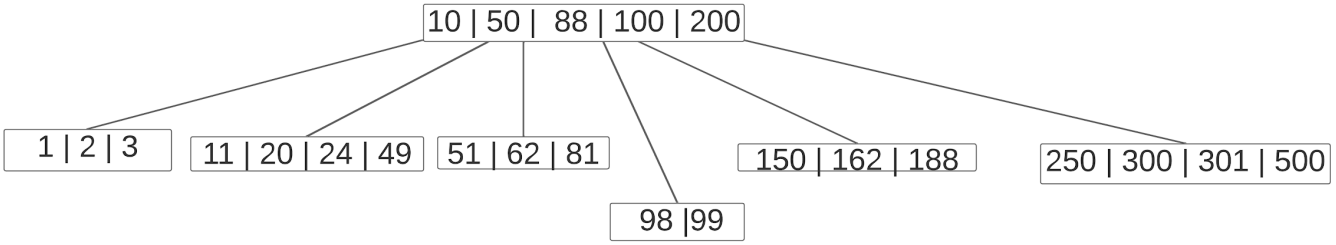
add: 24, 49, 98,



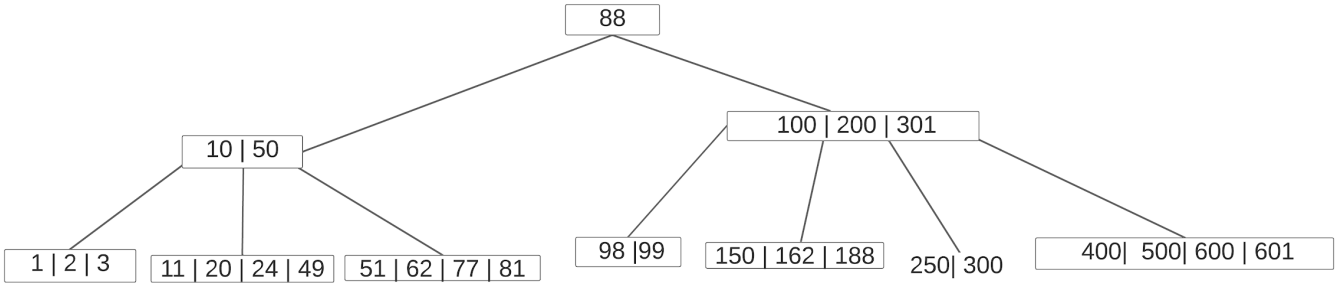
add 51,



add 500, 150



b)



Q6. (1.0 ponto) Escolha uma alternativa para responder: Letra A ou B. Caso escolha as duas, ficará com 0 na questão.

A) O método **intercalar**, do algoritmo de **merge sort,** ordena um vetor de n posições, onde os subvetores já estão ordenados. Os índices dos subvetores são: ini até meio; meio+1 até fim.

Seja:
i variável para percorrer a primeira metade do vetor;
j a variável para percorrer a segunda metade do vetor;
k a variável para percorrer o vetor auxiliar;

Preencha os trechos da função intercalar:

```
int intercalar(int vetor[], int ini, int meio, int fim) {
    int auxiliar[MAX];
    int i = ini, j = meio + 1, k = 0;

    // intercala
    while(i <= meio && j <= fim) {
        if (_____________)
            auxiliar[k++] = vetor[j++];
        else
        auxiliar[k++] = vetor[i++];
    }

    // copia resto de cada subvetor
    while (_____________) auxiliar[k++] = vetor[j++];
    while (_____________) auxiliar[k++] = vetor[i++];

    // copia de auxiliar para vetor
    for (i = ini, k=0; i <= fim; i++,k++)
        vetor[i] = auxiliar[k];
}
```

1. vetor[j] < vetor[i]
2. j<fim
3. i<meio+1

B) O algoritmo de **ordenação por seleção** faz as seguintes etapas:

1. seleciona o primeiro elemento do vetor desordenado, (linha 5)
2. busca a posição em que ele deveria estar no vetor ordenado, (linha 6)
3. faz o deslocamento para frente, em 1 posição, dos elementos maiores que ele, para criar espaço para a inserção, (linha 7)
4. e adiciona o elemento na posição correta (linha 8)

Esse procedimento está descrito nas linhas 5 a 8 do código a seguir. **Você deve implementar a função da etapa 3, deslocamento do subvetor.** Os parâmetros da função são: vetor, índice da posição inicial, índice da posição final. Todos os elementos nesse intervalo devem ser deslocados 1 posição para frente.

```c
int ordenar_insercao(int vetor[], int n) {
    int i, posicao;
    int elemento;
    for (i = 1; i < n; i++) {
        elemento = vetor[i];
        posicao = posicao_elemento(vetor, i-1, elemento);
        deslocar_subvetor(vetor, posicao, i-1);
        vetor[posicao] = elemento;
    }
}
```

(SLIDE DE ORDENAÇÃO)

Q7. (0.5 ponto) Para a árvore inicial da Q4, mostre o resultado dos percursos:

pós-ordem: 7, 5, 27, 25, 40, 30, 20

largura: 20, 5, 30, 7, 25, 40, 27

Extra (0.5 ponto): Indique duas variações que visam otimizar o método do bubblesort. Explique-as.

(SLIDE DE ORDENAÇÃO: 1. oscilar a ordenação com maior no final e menor no inicio. 2. verificar se já está ordenado, se não houve troca na ultima iteração)