P1 INFA010 - 2021.2

DANIEL STULBERG HUF - 2920408 - TURMA 3WA

QUESTÃO 1

(a) Para tal algoritmo, o pior caso será quando tiverem sido realizados todas as iterações em sequência, ou seja, que a operação de soma tenha sido feita $(for_1) * (for_2) * (for_3)$ vezes.

→ Analisando cada for em separado:

① For (i=1; i<n; i*=2)

↳ i é multiplicado por dois a cada operação, → até chegar a n computando $\log_2(n-1)$ vezes

② For (j=n; j>0; j/=2)

↳ j é dividido por 2 a cada operação até chegar a zero, computando $\log_2(n)$ vezes

③ For (K=j; K<n; K+=2)

↳ K é somado de dois em dois, computando $\frac{(n-j)}{2}$ vezes

Temos que, no pior dos casos, a soma será feita $\log(n) \cdot \log(n) \cdot n$ vezes

↓

$$\boxed{O(\log(n)^2 \cdot n)}$$

(b) Para tal algoritmo, o pior caso será quando tiverem sido realizados todas as iterações em sequência, ou seja, que as operações de soma tenham sido feitas $|for_1| * |for_2| * |for_3 + for_4|$ vezes

→ Analisando cada for em separado:

① For(i = 1; i <= n; i << 2)

↳ i é multiplicado por 4 a cada operação até chegar em n, computando $\log_4(n)$ vezes.

② For(j = 0; j < i; j++)

↳ j é incrementado até chegar a i, computando i vezes

③ For(K = 0; K < n; K += 2)

↳ K é somado de dois em dois, computando $\frac{n}{2}$ vezes

④ For(K = 1; K < n; K *= 2)

↳ K é multiplicado por dois a cada operação, computando $\log_2(n-1)$ vezes

temos que, no pior dos casos, a soma será feita

$\log(n) \cdot n \cdot (n + \log(n))$ vezes

$=$

$n^2 \log(n) + n \log^2(n)$ vezes

$\approx$ $>>$

$\boxed{O(n^2 \cdot \log(n))}$

P1 INF1010 – 2021.2

DANIEL STULBERG HUF – 1920468 – TURMA 3WA

QUESTÃO 2

→ A função não é correta porque ela não permite encontrar um nó através de um percurso que faça um caminho do tipo direita-esquerda ou esquerda-direita-esquerda (como se fosse um "zigue-zague" na árvore). Isso acontece porque a função só permite que a busca seja feita percorrendo tudo o que é necessário para a esquerda e depois tudo o que é necessário à direita devido à sua estrutura sequencial de whiles.

→ Como por exemplo, na árvore binária exibida, não seria possível encontrar a chave 50, visto que ela está à esquerda de sua antecessora, mas essa antecessora está à direita de sua antecessora (nó raiz)

Simulando a busca da chave 50:

Início → m = nó com chave 40

1º while
- no != NULL ? SIM e
- 40 > 50 ? NÃO → sai do 1º while

2º while
- no != NULL ? SIM e
- 40 < 50 ? SIM
  - → m = m→dir = nó com chave 60
- no != NULL ? SIM e
- 60 < 50 ? NÃO → sai do 2º while

no == NULL ? NÃO

RETORNA m→dados = 60 → Nesse exemplo, a função retorna 60 em vez de retornar 50

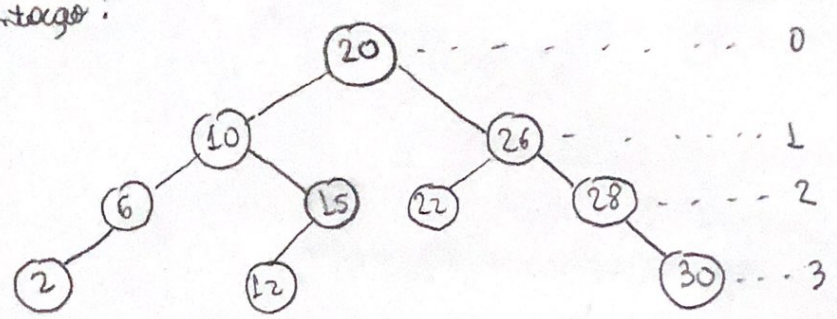
P1 INF1010 - 2021.2

DANIEL STULBERG MUF - 1920468 - TURMA 3WA

QUESTÃO 3

Árvore ABB → sequência de inserção: 20-10-6-2-15-12-26-22-28-30

Representação:



(a) Uma possibilidade que passou à cabeça foi que, à medida que os nós forem sendo inseridos nas árvores, os campos que contém a posição de cada nó em ordem crescente forem atualizados. Por exemplo, o primeiro nó (valor 20) a ser inserido estaria na posição 1, mas com a inserção do segundo nó (valor 10), o primeiro nó iria ser atualizado com a posição 2.

Entretanto, pensei em uma outra alternativa mais simples de ser implementada ao meu ver (porém talvez menos otimizada computacionalmente), que consiste em reconstruir a árvore da maneira habitual e sem preencher os campos com a posição. Em seguida, haveria um percurso em ordem simétrica que irá preencher de forma incremental o campo da posição de cada nó acessado no percurso.

O pseudocódigo está descrito abaixo:

PSEUDO CÓDIGO:

```
INSERE (NO, CHAVE):
    IF (NO É NULO):
        INICIALIZA NO;
        NO → CHAVE = CHAVE;
        NO → ESQ = NO → DIR = NULO;
        NO → POS = NULO;

    ELSE IF ( CHAVE < NO → CHAVE)
        IF (NO → ESQ = NULO)
            NO → ESQ = CRIA NO A ESQUERDA
        ELSE:
            NO → ESQ = INSERE (NO → ESQ , CHAVE)
    ELSE IF (CHAVE > NO → CHAVE)
        IF (NO → DIR = NULO)
            NO → DIR = CRIA NO A DIREITA
        ELSE:
            NO → DIR = INSERE (NO → DIR, CHAVE)
    RETORNA NO

PREENCHE_POSICAO_CRESCENTE (NO, POS):
    IF ( NO != NULO) :
        PREENCHE_POSICAO_CRESCENTE (NO → ESQ, POS)
        NO → POS = POS
        POS += 1
        PREENCHE_POSICAO_CRESCENTE (NO → DIR, POS)
```

SEQ = { 20, 10, 6, 2, 15, 12, 26, 22, 28, 30 }

```
RECONSTROI_ABB (SEQ):
    NO = CRIA_ÁRVORE
    FOR ( i = 0 ; i < TAM(SEQ) ; i++)
        INSERE ( NO, SEQ[i]),
    PREENCHE_POSICAO_CRESCENTE (NO, POS)
```

INICIALIZADO EM 1

OBS: Em um código formal, pos poderia ser uma variável global ou um ponteiro passado como parâmetro e sempre atualizado.

(b) PSEUDO CÓDIGO:

```
FROM LETRA A IMPORT INSERE
```

A função insere é a mesma, somente trocando a linha:

NO→POS = NULO

por

NO→N_SUBS_DIR = NULO

NO→N_SUBS_ESQ = NULO

```
CONTA_SUBS (NO):
    SOMA = 0
    IF (NO = NULO):
        NO→N_SUBS_DIR = 0
        NO→N_SUBS_ESQ = 0
        RETURN SOMA
    IF (NO→ESQ = NULO):
        NO→N_SUBS_ESQ = 0
    ELSE:
        NO→N_SUBS_ESQ = 1 + CONTA_SUBS (NO→ESQ)
    IF (NO→DIR = NULO):
        NO→N_SUBS_DIR = 0
    ELSE:
        NO→N_SUBS_DIR = 1 + CONTA_SUBS (NO→DIR)

    SOMA = NO→N_SUBS_ESQ + NO→N_SUBS_DIR
    RETURN SOMA
```

SEQ = {20, 10, 6, 2, 15, 12, 26, 22, 28, 30}

```
RECONSTROI_ABB (SEQ):
    NO = CRIA_ARVORE
    FOR (i=0; i< TAM(SEQ); i++)
        INSERE (NO, SEQ[i]);
    CONTA_SUBS (NO);
```

PA INFA010 - 2021.2

DANIEL STULBERG HUF - 1920468 - TURMA 3WA

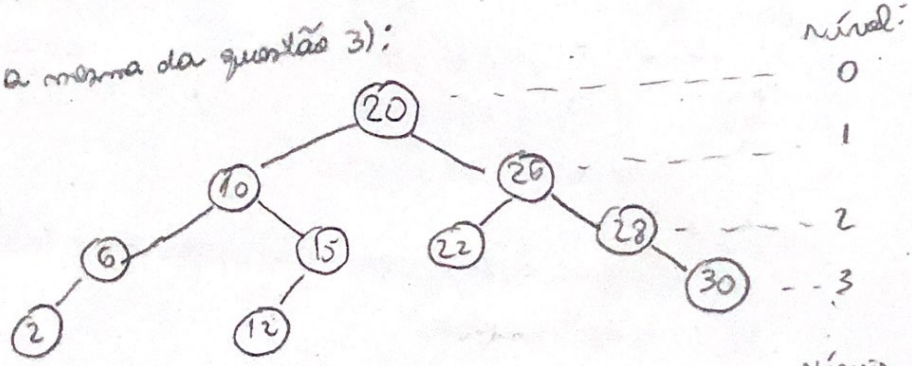
10/10

QUESTÃO 4

SIM, para pré-sim e sim-pós
NÃO necessariamente, para pré-pós

Usarei como exemplo uma árvore binária arbitrária que será reconstruída para cada par possível de percursos.

Representação (a mesma da questão 3):



Percursos:

Ordem simétrica: $\overset{(3)}{2} - \overset{(2)}{6} - \overset{(1)}{10} - \overset{(3)}{12} - \overset{(2)}{15} - \overset{(0)}{20} - \overset{(2)}{22} - \overset{(1)}{26} - \overset{(2)}{28} - \overset{(3)}{30}$ → Níveis

Pré-ordem: $\overset{(0)}{20} - \overset{(1)}{10} - \overset{(2)}{6} - \overset{(3)}{2} - \overset{(2)}{15} - \overset{(3)}{12} - \overset{(1)}{26} - \overset{(2)}{22} - \overset{(2)}{28} - \overset{(3)}{30}$

Pós-ordem: $\overset{(3)}{2} - \overset{(2)}{6} - \overset{(3)}{12} - \overset{(2)}{15} - \overset{(1)}{10} - \overset{(2)}{22} - \overset{(3)}{30} - \overset{(2)}{28} - \overset{(1)}{26} - \overset{(0)}{20}$

(1) Reconstruindo a árvore com Ordem Simétrica e Pré-ordem

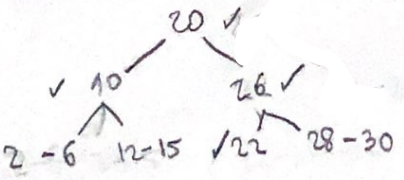
→ Raíz da árvore = elemento à esquerda do pré-ordem = 20

→ Olhando para a ordem simétrica:

$$\underbrace{2 - 6 - 10 - 12 - 15}_{\text{sub árvore esquerda}} - 20 - \underbrace{22 - 26 - 28 - 30}_{\text{sub-árvore direita}}$$

→ Olhando o percurso pré-ordem: 10 é o próximo elemento = raíz da sub árvore à esquerda

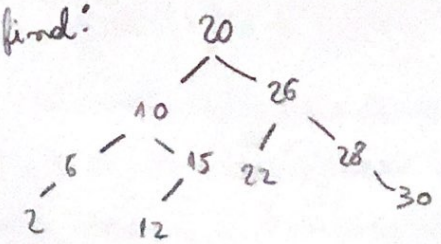
26 é o próximo elemento = raíz da sub-árvore à direita



→ Olhando o pré-ordem, 6 vem antes de 2, 15 vem antes de 12, e 28 vem antes de 30.

6 → próximo; 15 → próximo; 28 → próximo

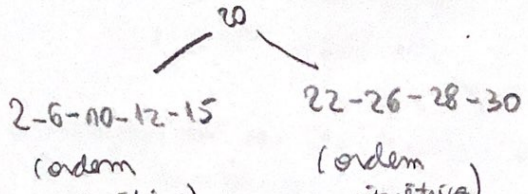
→ Árvore final:



(se houvessem mais nós, o passo a passo recursivo seria o mesmo, onde começamos descobrindo a raíz e escrevendo a ordem simétrica das sub-árvores, e analisemos cada nó da ordem simétrica para determinar seus filhos à direita e à esquerda.

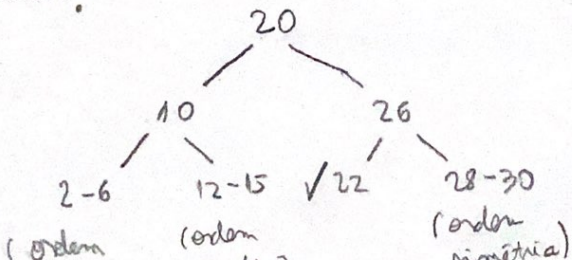
② Reconstruindo a árvore com ordem simétrica e pós-ordem

→ Raíz da árvore = elemento à direita do pós-ordem = 20



mais à direita no pós ordem? 10

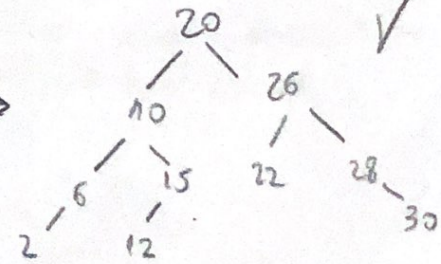
mais à direita no pós ordem? 26



mais a direita? 6

mais a direita? 15

mais à direita? 28



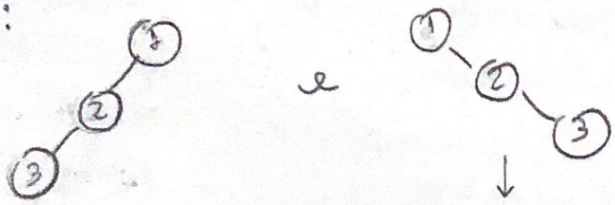
(se houvessem mais nós, o passo a passo recursivo seria o mesmo, começamos descobrindo a raíz e escrevendo a ordem simétrica das sub árvores, e depois olhamos a pós ordem para ver quem está mais a direita, que é o próximo a ser inserido.

③ Reconstruindo a árvore com pré-ordem e pós-ordem.

→ Condição: Para uma árvore ser reconstruída usando esses 2 percursos, os nós dessa árvore só podem possuir 0 ou 2 filhos. Caso contrário, a árvore construída não necessariamente será igual à original.

estritamente binária

Por exemplo:



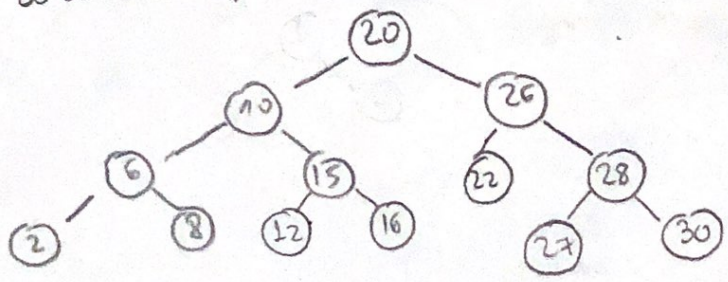
Pré ordem: 1-2-3
Pós ordem: 3-2-1

Pré-ordem: 1-2-3
Pós-ordem: 3-2-1

Como ambos os percursos são iguais nos 2 casos, não iremos reconstruir corretamente a árvore original em algum dos casos, pois não saberemos se o nó irá para a esquerda ou direita.

isso gerou AMBIGUIDADE

→ Portanto, iremos inchar a árvore original para reconstruí-la corretamente:



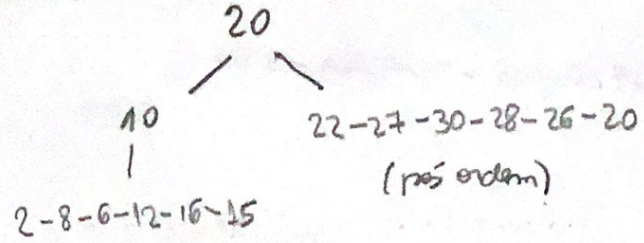
Pré-ordem: 20-10-6-2-8-15-12-16-26-22-28-27-30

Pós-ordem: 2-8-6-12-16-15-10-22-27-30-28-26-20

→ raíz da árvore = elemento à esquerda da pré-ordem = elemento à direita da pós-ordem = 20

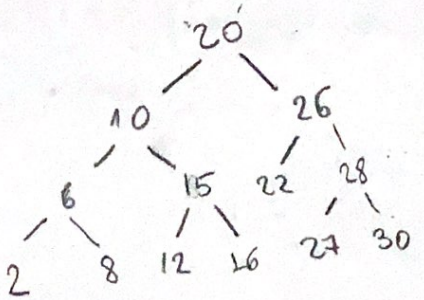
→ Como estamos tratando de uma árvore estritamente binária, temos certeza que o elemento depois de 20 na pré-ordem virá logo à esquerda da raiz (10).

→ Olhando o 10 na pós ordem, separamos as sub-árvores à direita e à esquerda.



→ Prox-elemento pré-ordem?

Percurso Pré-ordem:



6 → necessariamente estará à esquerda de 10

↳ 2 → necessariamente estará à esquerda de 2

↳ 8 → necessariamente à direita de 6 (completar)

↳ 15 → necessariamente à direita de 10

↳ 12 → esquerda de 15

↳ 16 → direita de 15

↳ 26 → direita de 20

↳ 22 → esquerda de 26

↳ 28 → direita de 26

↳ 27 → esquerda de 28

↳ 30 → direita de 28

→ Se houvesse mais nós, o passo a passo recursivo seria o mesmo. Primeiro usamos o percurso pré-ordem para acessar cada nó. Depois, usamos o percurso pós-ordem para encontrar os descendentes de cada nó. Além disso, o fato da árvore ser estritamente binária nos garante a posição (direita ou esquerda) em que devemos colocar o próximo filho acessado na sequência pré-ordem.

P1 INF1010 - 2021.2

DANIEL STULBERG HUF - 1920468 - TURMA 3WA

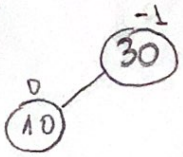
QUESTÃO 5

Árvore AVL → Sequência de inserções: **30 - 10 - 60 - 40 - 50 - 70**

Passo a passo:

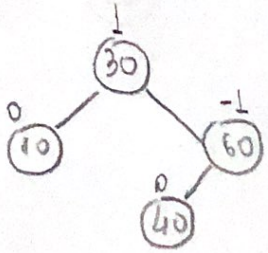
① Inserção da chave 30

② Inserção da chave 10

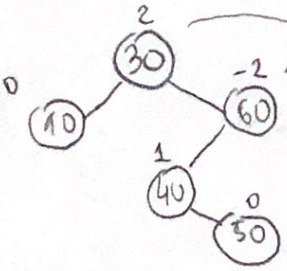


③ Inserção da chave 60

④ Inserção da chave 40

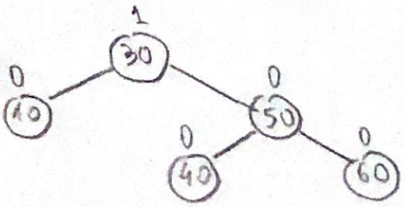


⑤ Inserção da chave 50

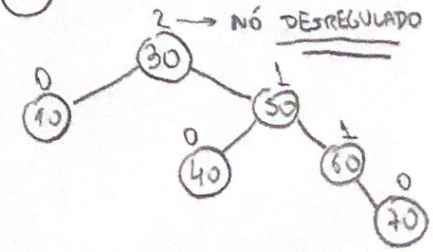


NÓS DESREGULADOS

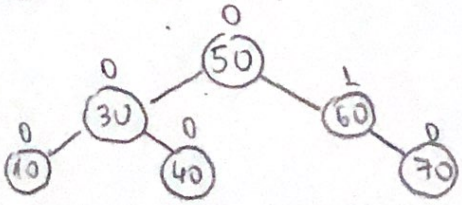
⑥ Rotação esquerda-direita



⑦ Inserção da chave 70



⑧ Rotação à esquerda



P1 INFO010 - 2021.2

DANIEL STULBERG HUF - 1920468 - TURMA 3WA

QUESTÃO 7

(a) **FALSO**

→ Temos que o nível de um nó de uma árvore binária é definido como o nível de seu pai + 1, sendo o nível da raíz = 0. já a altura de uma árvore é o maior nível existente entre todos os nós (que seria o nível da folha mais profunda).

→ Portanto, para um contra-exemplo, basta escolhermos uma árvore binária que possua pelo menos mais um nó além da raíz.

Por exemplo:

Nível da raíz = 0

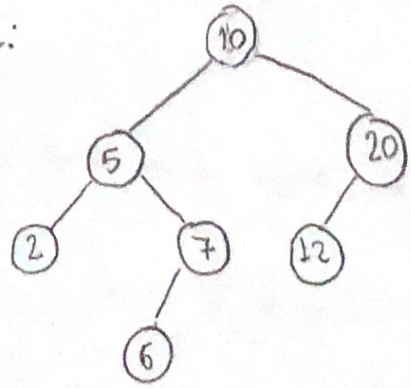
Altura da árvore = 1

$0 \neq 1$ ■

(b) **FALSO**

→ Seguindo a definição de nível de um nó de uma árvore binária e sabendo que a altura de um nó de uma árvore é dado pelo número de arestas que ligam diretamente esse nó à folha mais profunda que seja sua descendente, temos que o nível do nó não necessariamente é igual à sua altura.

Por exemplo:



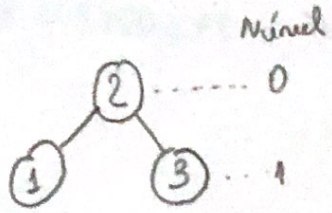
Nível do nó 7 = 2

Altura do nó 7 = 1

$2 \neq 1$ ■

(c) $\boxed{\text{FALSO}}$

Contra-exemplo:



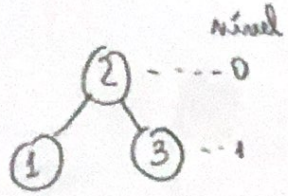
Níveis = 2

Nós = 3

$2^2 = 4 \neq 3$ ■

(d) $\boxed{\text{FALSO}}$

Contra-exemplo:



Níveis = 2

Nós = 3

$2^{2-1} = 2^1 = 2 \neq 3$ ■

É uma árvore estritamente binária, pois cada nó tem 0 ou 2 filhos.