checklist

fiz incompleto não consegui

Pilhas e filas

1 2 3 4 5 6 7 8 9

Apontadores, listas ligadas e árvores

1 2 3 4 implementar funções

Árvores de busca binária balanceadas

1 2 3 4

Lista 1 - MAC0323

Pilhas e Filas

1. As operações de colocar e tirar os n vagões do estacionamento podem ser codificadas concisamente usando a letra E para "empilhar" (ou colocar um vagão no estacionamento) e D para "desempilhar" (ou tirar um vagão). Chamamos uma sequência de E's e D's de *admissível* se contém n E's, n D's e as operações codificadas podem ser realizadas. Por exemplo, a sequência EDEEEEDDEEEDDDDD é admissível, enquanto a sequência EDDEEEDD não é admissível. Formule uma regra que permita diferenciar as sequências admissíveis das que não são.

Essas operações são similares às implementações de pilhas e filas.

Uma regra possível que permita diferenciar as sequências admissíveis das que não são seria, inicialmente, guardar o número de vagões já empilhados (quantos E's na sequência) e depois verificar a cada D's se o numero de D's > E's, se for verdade então a sequência não é admissível, caso contrário a sequência é admissível. Ainda mais, o número de D's e E's deve ser igual.

2. A partir do exercício acima, derive uma fórmula simples que, dado n, você possa calcular o número de sequências de n E's e n D's válidas.

3. Mostre que é possível obter uma permutação $p_1, p_2, \ldots, p_n$ a partir de $1, 2, \ldots, n$ usando uma pilha se e somente se não existirem índices i, j e k tais que $i < j < k$ e $p_k < p_i < p_j$.

- Mostrar que uma permutação $p_1\ p_2\ p_3 \ldots p_n$ é admissível se e somente se não existirem índices i, j, k satisfazendo

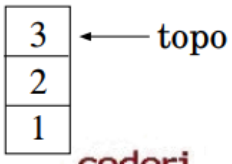
  $i < j < k$ e $p_j < p_k < p_i$

  Exemplo: $p_1\ p_2\ p_3 = 3\ 1\ 2$ não é admissível, pois $p_2 < p_3 < p_1$

4. Faça um algoritmo que dado uma permutação $p_1, p_2, \ldots, p_n$ de $1, 2, \ldots, n$ verifique se a permutação pode ser obtida utilizando uma pilha e caso seja, retorne a sequência de operações (E's e D's) que devem ser realizadas.

Seja S a sequência fornecida pelos inteiros 1, 2, ..., n, cujos elementos são inseridos em uma pilha P, em ordem crescente. As remoções de P podem ocorrer de forma qualquer, respeitando a condição de underflow. Uma permutação admissível é uma sequência formada pelos elementos de S, que corresponde a alguma ordem válida de remoção de P.

- Exemplo: se S = 1, 2, 3 então
  - 3 2 1 é admissível, pois
    I 1, I 2, I 3, R, R, R => 3 2 1
  - 2 3 1 é admissível, pois
    I 1, I 2, R, I 3, R, R => 2 3 1
  - 3 1 2 não é admissível, pois



cederj

5. Passe a expressão aritmética abaixo para a notação posfixa, indicando para cada caractere lido o conteúdo da pilha de operadores.

$$A + B * (C + D)/E - B - D$$

na notação posfixa os operadores são escritos depois dos operandos

1. Crie uma pilha vazia chamada `opstack` para manter os operadores. Cria uma lista vazia para a saída.
2. Converta a string infixa input para uma lista usando o método `split()`.
3. Examine os itens da lista da esquerda para a direita.
    - Se o item é um operador, coloque-o no final da lista da saída.
    - Se o item é um abre parêntese, insira-o (`push()`) na pilha `opstack`.
    - Se o item é um fecha parênteses, remova (`pop()`) os itens de `opstack` até que o abre parêntese correspondente seja removido. Coloque cada operador removido no final da lista da saída.
    - Se i item é um operador, *, /, +, or -, insira-o na pilha `opstack`. Entretanto, remova antes os operadores que estão na pilha que têm precedência maior ou igual ao operador encontrado e coloque-os na final da lista da saída.
4. Quando a expressão tiver sido completamente examinada, verifique `opstack`. Qualquer operador que ainda está na pilha deve ser removido e colocado na lista da saída.

https://panda.ime.usp.br/panda/static/pythonds_pt/03-EDBasicos/09-ExpressoesInfixaPrefixaPosfixa.html

→ A + B * (C + D) / E - B - D

infixa	pilha	posfixa
A		A
A +	+	A
A + B	+	AB
A + B *	+ *	AB
A + B * (	+ * (	AB
A + B * (C	+ * (	ABC
A + B * (C +	+ * (+	ABC
A + B * (C + D	+ * (+	ABCD
A + B * (C + D)	+ *	ABCD +
A + B * (C + D) /	+ /	ABCD + *
A + B * (C + D) / E	+ /	ABCD + * E
A + B * (C + D) / E -	-	ABCD + * E / +

A+B*(C+D)/E-B -

A+B*(C+D)/E-B- -

A+B*(C+D)/E-B-D -

ABCD+*E/+B

ABCD+*E/+B-

ABCD+*E/+B-D

→ ABCD+*E/+B-D-

6. Resolva a expressão posfixa do exercício acima utilizando como valores: $(A, B, C, D, E) \rightarrow (7, 10, 3, 9, 4)$. Mostre o conteúdo da estrutura de dados que você utilizar após cada passo.

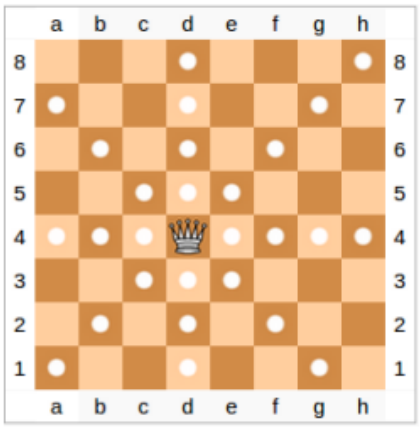
expressão posfixa: ABCD+ * E / + B - D -

posfixa	pilha	comentário
7	7	-
10	7 10	-
3	7 10 3	-
9	7 10 3 9	-
+	7 10 12	soma 3 e 9
*	7 120	multiplica 10 e 12
4	7 120 4	-
/	7 30	divide 120 e 4
+	37	soma 7 e 30
10	37 10	-
-	27	subtrai 37 e 10
9	27 9	-
-	18	subtrai 27 e 9

1. Leia um caractere da notação.
2. Se for um operando, empilhe-o, isto é, coloque-o no topo da pilha.
3. Mas se ele for um operador, retire o penúltimo e o último operando da pilha. Faça a operação que se pede entre eles e coloque o resultado de volta na pilha.
4. Repita todos os passos acima até que toda a notação seja lida.
5. O resultado será o que sobrou na pilha.

https://pt.wikipedia.org/wiki/Notação_polonesa_inversa

7. O **problema das 8 rainhas** consiste no seguinte: dado um n, determinar se existe uma maneira de colocar n rainhas num tabuleiro de xadrez $n \times n$ sem que nenhuma delas ataque a outra. Escreva um algoritmo que resolva o problema das 8 rainhas. Caso exista uma maneira de colocar n rainhas, seu algoritmo deve imprimir as posições das n rainhas para uma destas maneiras válidas de colocá-las. (No xadrez, uma rainha ataca qualquer peça que esteja na mesma linha, coluna ou diagonal que ela. Veja figura abaixo).



8. Um famoso *site* de relacionamentos está desenvolvendo uma nova *feature*. Eles informam a você uma pessoa que é bastante distante do seu círculo de amigos. Para isso desenvolveram a seguinte métrica, chamada *grau de separação*:
Eles têm a lista de todos os seus amigos. O valor do seu grau de separação para cada amigo é 1. Para cada amigo dos seus amigos, seu grau de separação é 2. E assim por diante.
Mas você não quer esperar, e resolveu achar essas pessoas "distantes" por conta própria. Faça uma função que receba uma lista de todas as pessoas do *site* de relacionamentos e a lista de amigos de cada uma dessas pessoas e que devolva uma das pessoas com maior grau de separação em relação a você mesmo.

árvore binária

9. Digamos que nosso alfabeto é formado pelas letras **a**, **b** e **c**. Considere o seguinte conjunto de cadeias de caracteres sobre nosso alfabeto:

`c, aca, bcb, abcba, bacab, aacaa, bbcbb, ...`

Qualquer cadeia deste conjunto tem a forma `WcM`, onde `W` é uma sequência de letras que só contém `a` e `b` e `M` é o inverso de `W`, ou seja, `M` é `W` lido de trás para frente. Escreva um algoritmo que determina se uma cadeia `X` pertence ou não ao nosso conjunto, ou seja, determina se `X` é da forma `WcM`.

1. Crie uma função iterativa que receba um ponteiro para o início de uma lista, inverta-a, e devolva o ponteiro para o novo início da lista. Faça também uma versão recursiva. Atenção, você não pode modificar o valor dos nós, apenas os ponteiros.

2. Crie uma função que receba o ponteiro para o início de uma lista e ordene-a em ordem crescente do conteúdo dos nós. Atenção, você não pode modificar o valor dos nós, apenas os ponteiros.

3. Faça uma função que receba ponteiros para duas listas de vértices de uma árvore binária visitados em "in-ordem" e "pre-ordem" e constrói a árvore binária correspondente.

4. Escreve uma função que receba uma lista encadeada e devolva um ponteiro para o nó que esteja o mais próximo possível do meio da lista. Faça isso sem contar explicitamente o número de nós da lista.

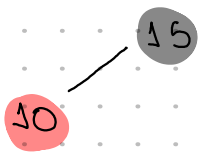
Árvores de busca binária balanceadas

1. Mostre o formato de uma árvore rubro-negra inicialmente vazia após a inserção de cada elemento abaixo:

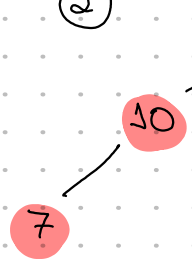
15 10 7 18 25 11 23 29 33

- a raiz de uma árvore RN é sempre preta
- um nó é sempre inserido na cor vermelha

①

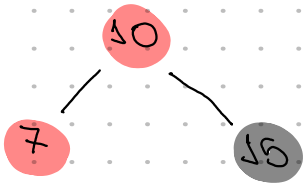


②



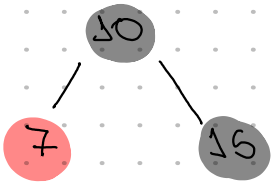
vermelho não tem filho vermelho → rotação

③



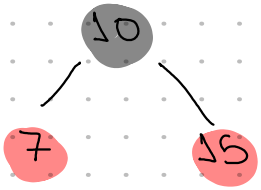
raiz tem que ser preta

④

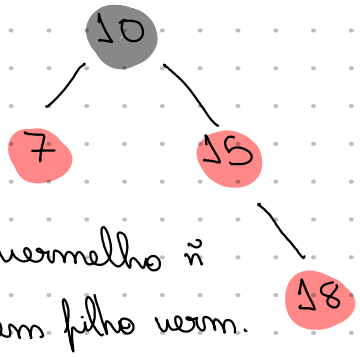


os níveis não tem o mesmo nº de pretos

⑤

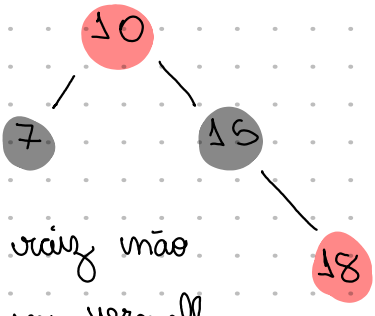


⑥



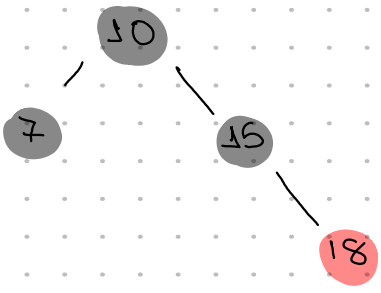
vermelho não tem filho verm. e pai e tio são vermelhos

⑦

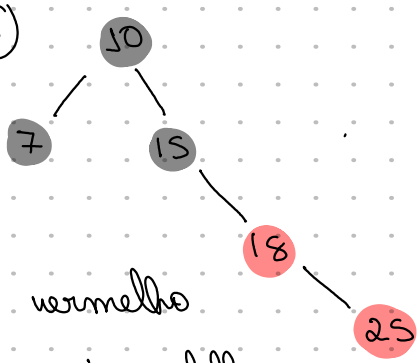


a raiz não pode ser vermelha

⑧

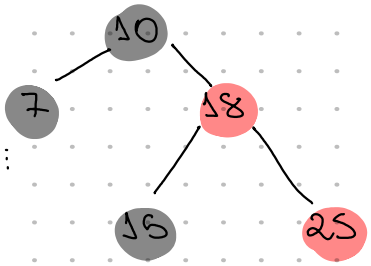


⑨



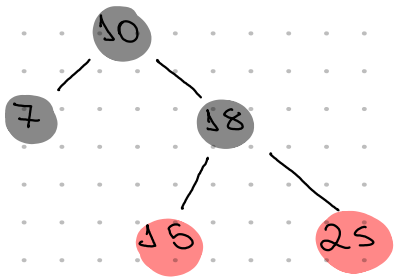
vermelho não tem filho vermelho e o tio não é vermelho → rotação

(10)



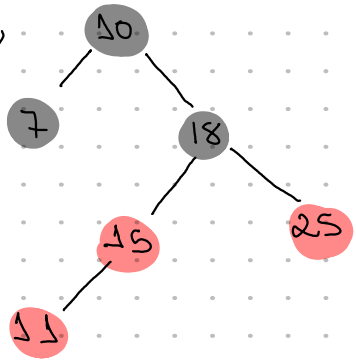
- não pode ter filho vermelho
- balancear o nº de pretos até a raiz

(11)



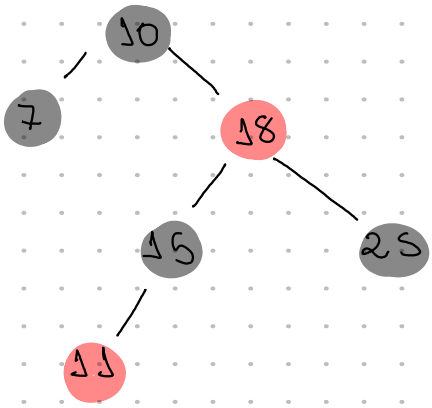
insere o (11)

(12)



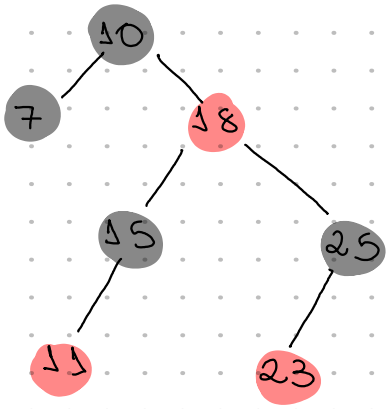
filho vermelho e tio vermelho → troca cor pai - tio - avô

(13)



insere 23

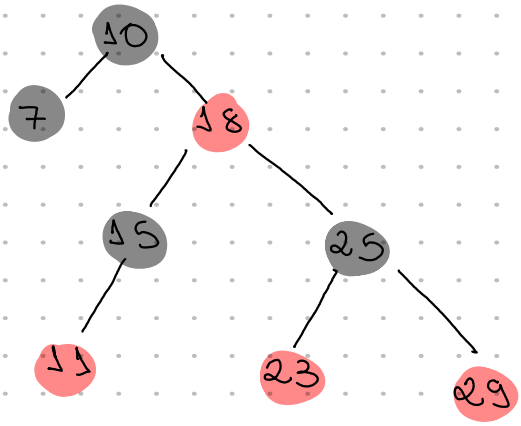
(14)



o pai é preto então OK

insere 29

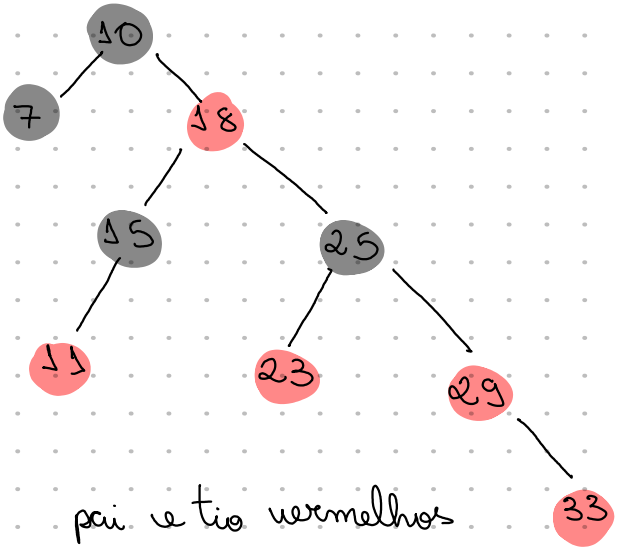
(15)



perfeito! 😊 pai preto também

insere 33

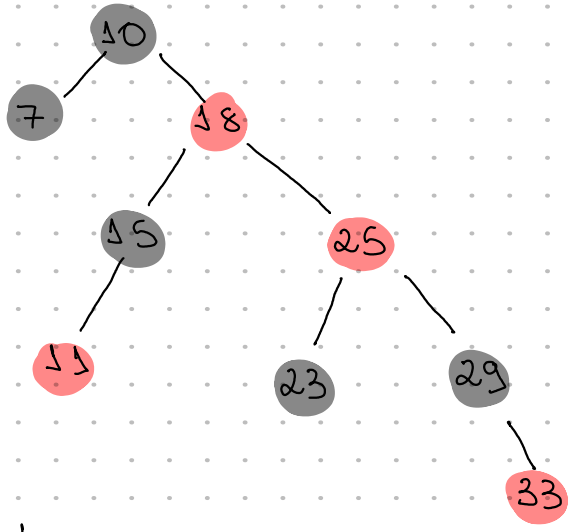
(16)



pai e tio vermelhos

trocar cor do pai - tio - avô

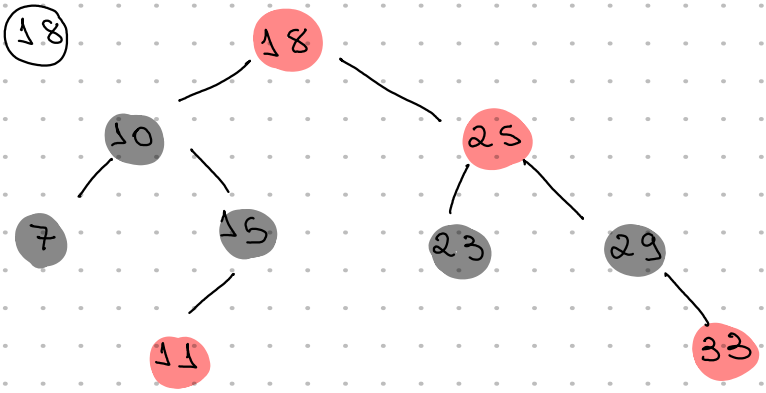
17



pai do 25 é vermelho

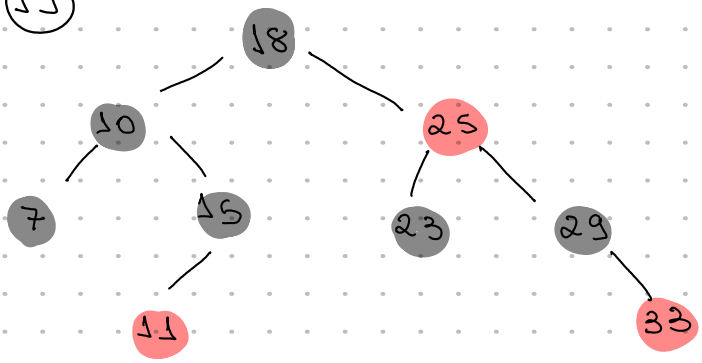
→ rotaciona para a esquerda

18



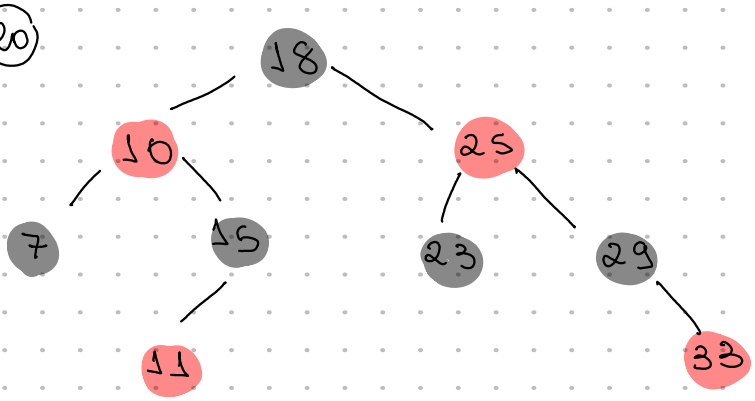
raiz tem que ser preta

19



muda cor do 10 para balancear o nº de pretos até as folhas

20



acabou! uhu

2. Desenhe uma árvore rubro-negra com $n = 20$ vértices de altura máxima.

$2(\log_2 n + 1)$

3. Considere uma árvore 2–3.
   a. Se a árvore tem $n = 30000$ elementos, qual é a altura mínima e máxima que ela poderá ter?
   b. Descreva a estrutura de dados para representar uma árvore 2–3.

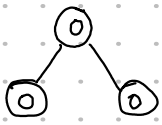
a) mínima $\log_3 30.000 = x$

máxima $\log_2 30000$

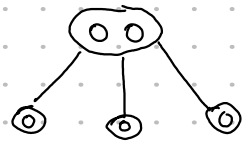
b)

<chave1, val>	<chave2, val2>	dois nos → bool
p1	p2	p3

explicar 2 nós:



3 nós:



4. a. Simule o algoritmo de inserção em árvores 2–3 para a montagem da árvore correspondente à sequência abaixo, mostrando detalhadamente os rearranjos que ocorrerem.

43 12 10 4 3 7 9 33 22 19 14 13 17 39

b. Remova da árvore obtida no item acima, sucessivamente os elementos 4, 3, 7 e 9. (simule detalhadamente).

1. 43

2. 12 43

inserimos o 12 e cabe no nó

3. inserimos o 10 na raiz

10 12 43

é muito grande então dividimos

4. 12 / 10 43

→ insere o 4

5. 12 / 4 10 43

→ insere o 3

6. 12

3 4 10 43

tem que dividir

7. 12 / 4 43 / 3 10

podemos colocar o 4 na raiz

8. 4 12 / 3 10 43

→ insere o 7

9. 4 12 / 3 7 10 43

→ insere o 9

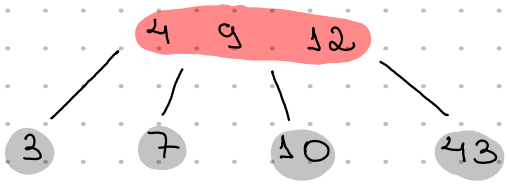
10. 4 12 / 3 7 9 10 43

muito grande, temos que dividir

11. 4 12 / 3 9 43 / 7 10

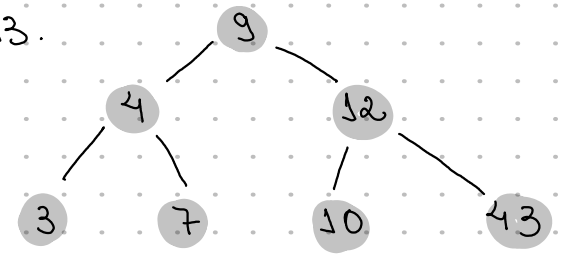
o 9 sobe para balancear

12.



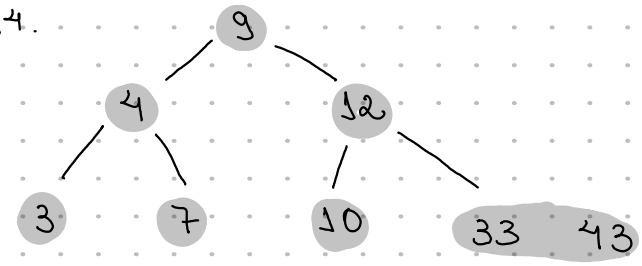
temos que dividir

13.



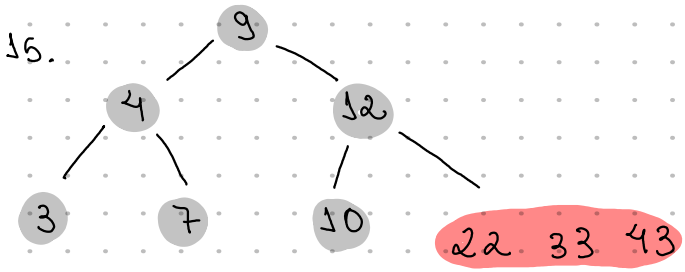
→ insere o 33

14.



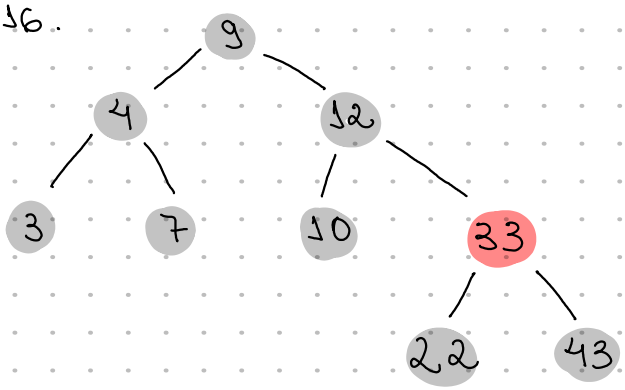
→ insere o 22

15.



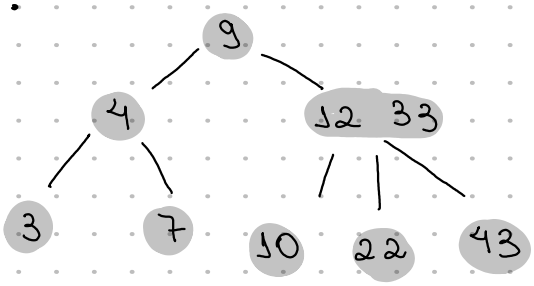
temos que dividir

16.



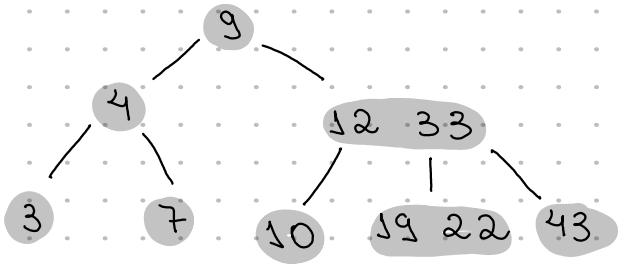
sobe o 33

17.



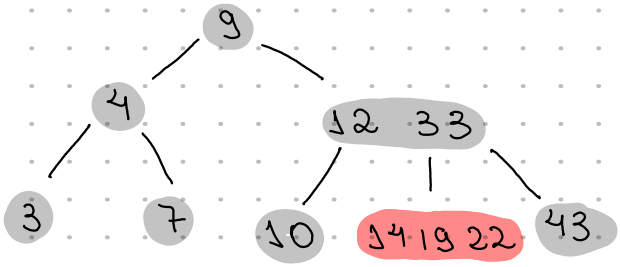
→ insere o 19

18.



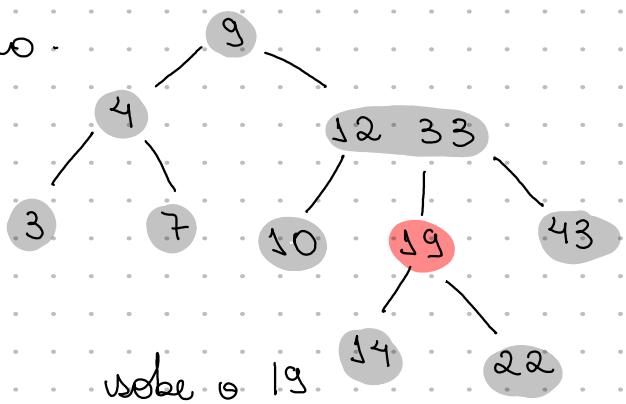
→ insere 14

19.



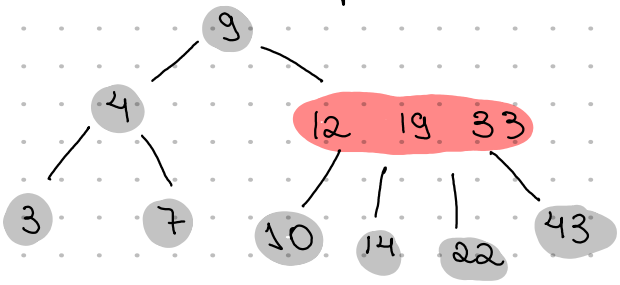
temos que dividir

20.



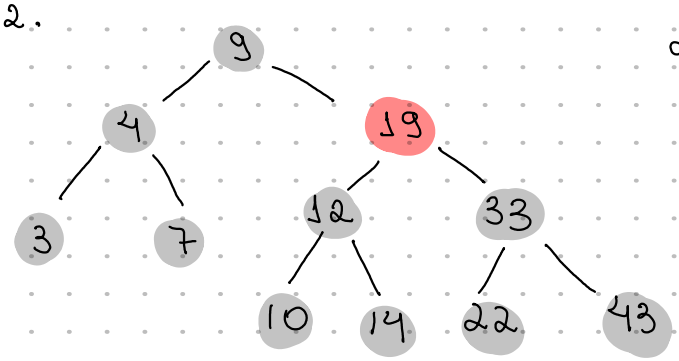
sobe o 19

21.



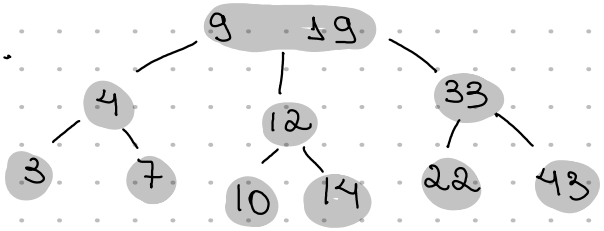
temos que dividir

22.



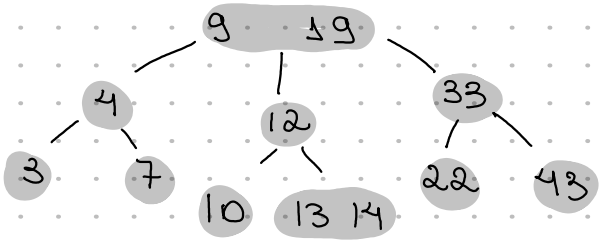
sobe o 19

23.



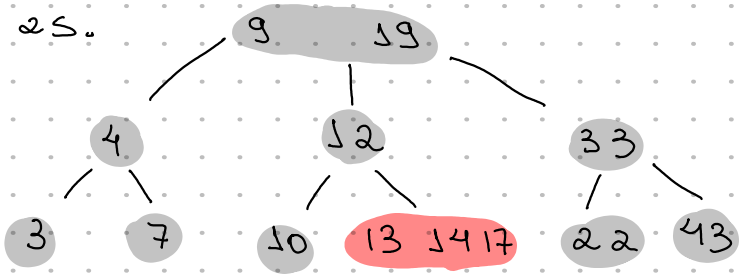
→ insere o 13

24.



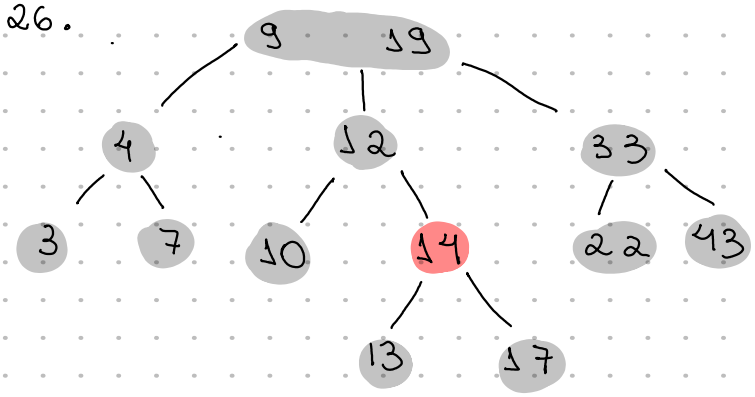
→ insere o 17

25.



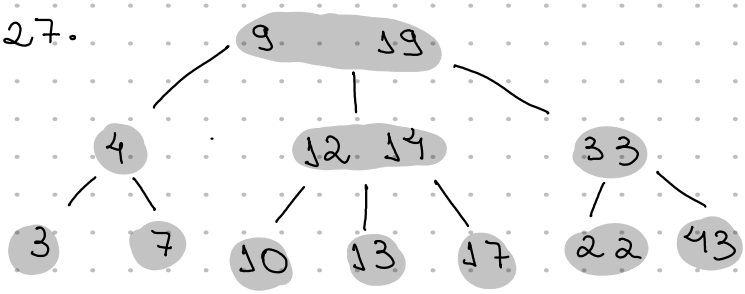
temos que dividir

26.



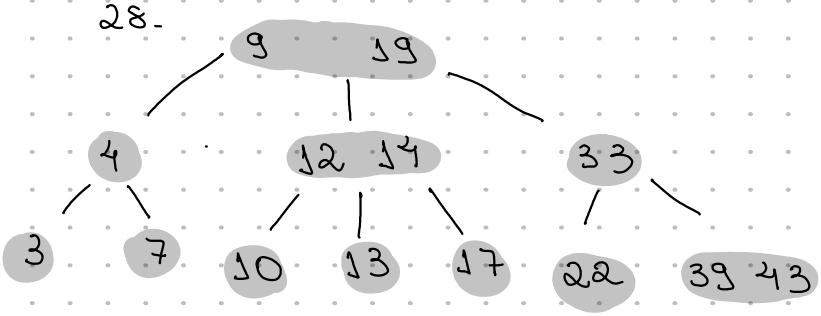
sobe o 14

27.



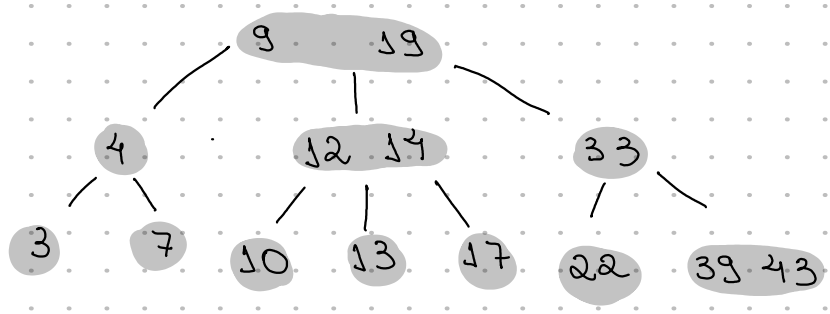
→ insere o 39

28.

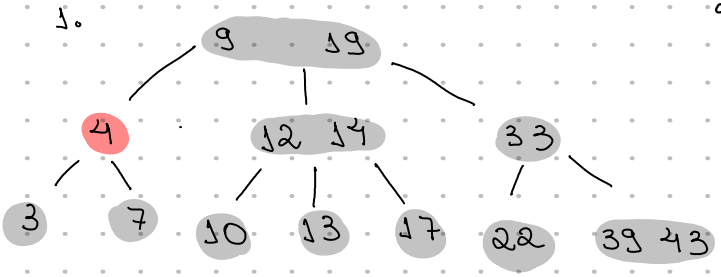


acabou!

b. Remova da árvore obtida no item acima, sucessivamente os elementos 4, 3, 7 e 9. (simule detalhadamente).

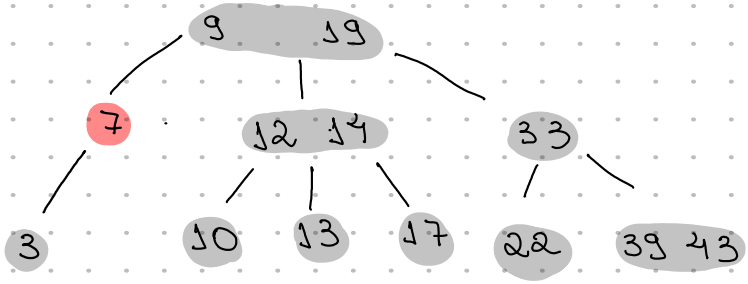


1.



remove o 4

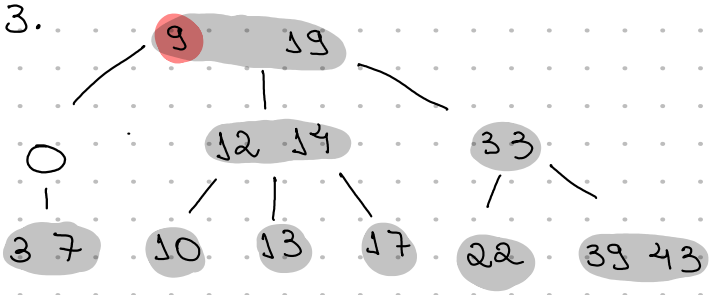
2.



o nó é pequeno

agrupamos o 3 e o 7

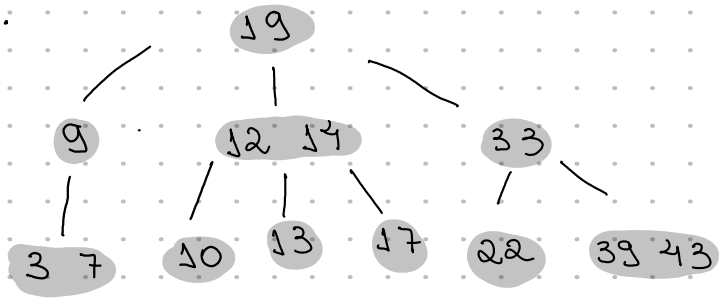
3.



a árvore não está balanceada

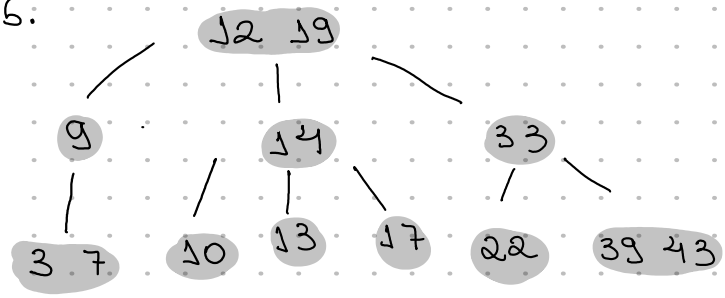
descemos o 9

4.



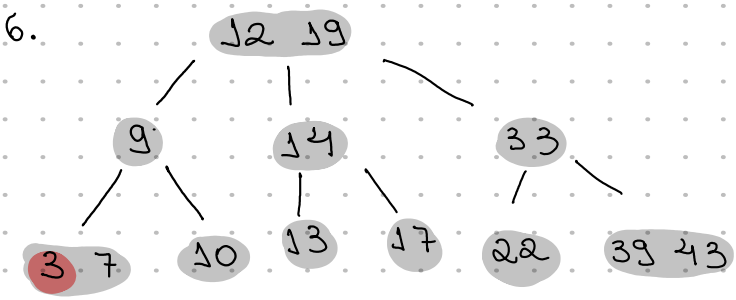
subimos o 12

5.



ajustamos o 10

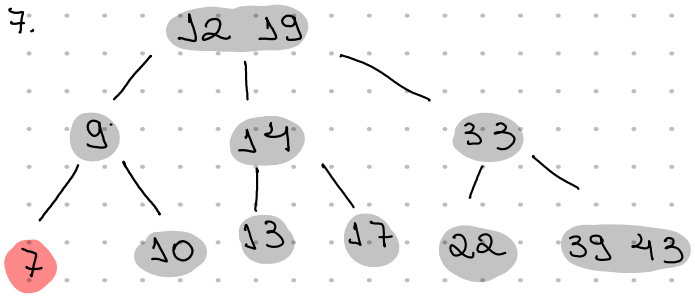
6.



remove o 3

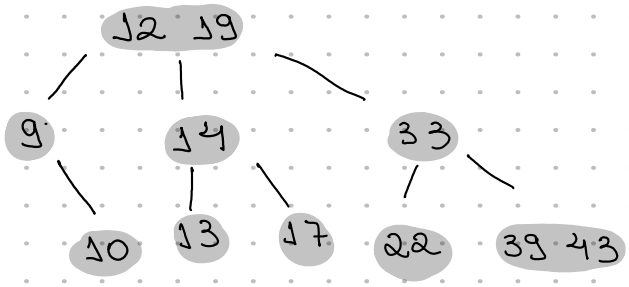
caso fácil

7.



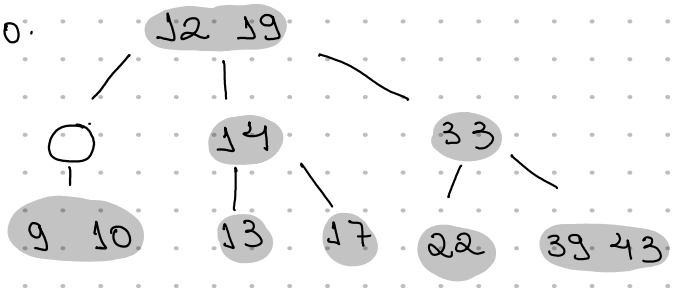
remove o 7

8.



agrupamos o 9 e 10

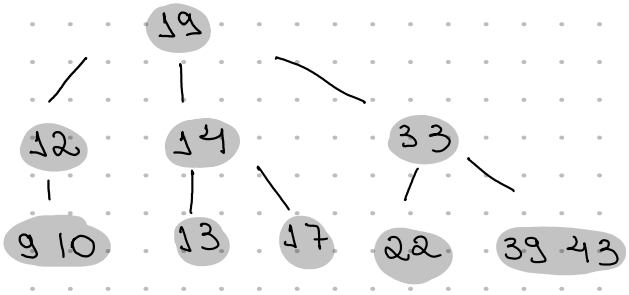
10.



a árvore não está balanceada

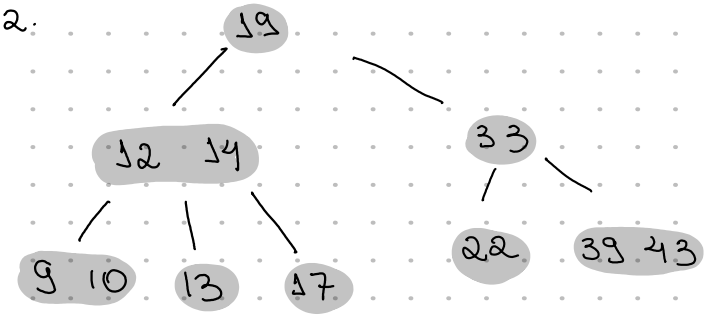
descemos o 12

11.



ajustamos o 14

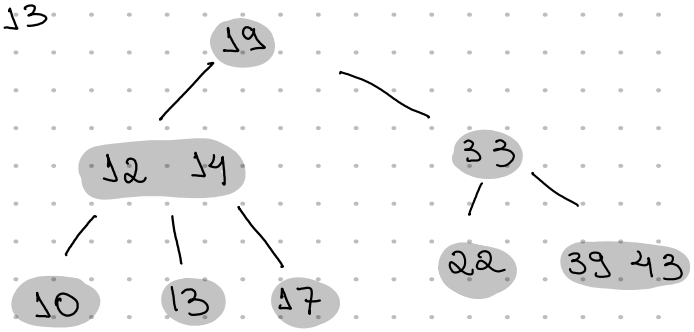
12.



→ remove o 9

caso fácil

13



acabou!