Aula 03 - Árvores Binárias de Busca

Estrutura de dados e seus algoritmos

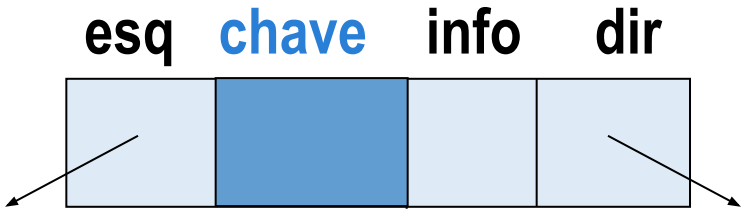
Profa: Taiane C. Ramos

Busca

- Buscas devem ser eficientes (rápidas).
- Podemos usar uma árvore binária para ganhar velocidade.
- A árvore deve ser organizada usando uma “chave”
- A chave tem que ser uma informação única e que tenha uma relação de ordem entre os nós (quem vem primeiro e quem vem depois)
- Exemplos de chave:
  - CPF
  - número de matrícula
  - url

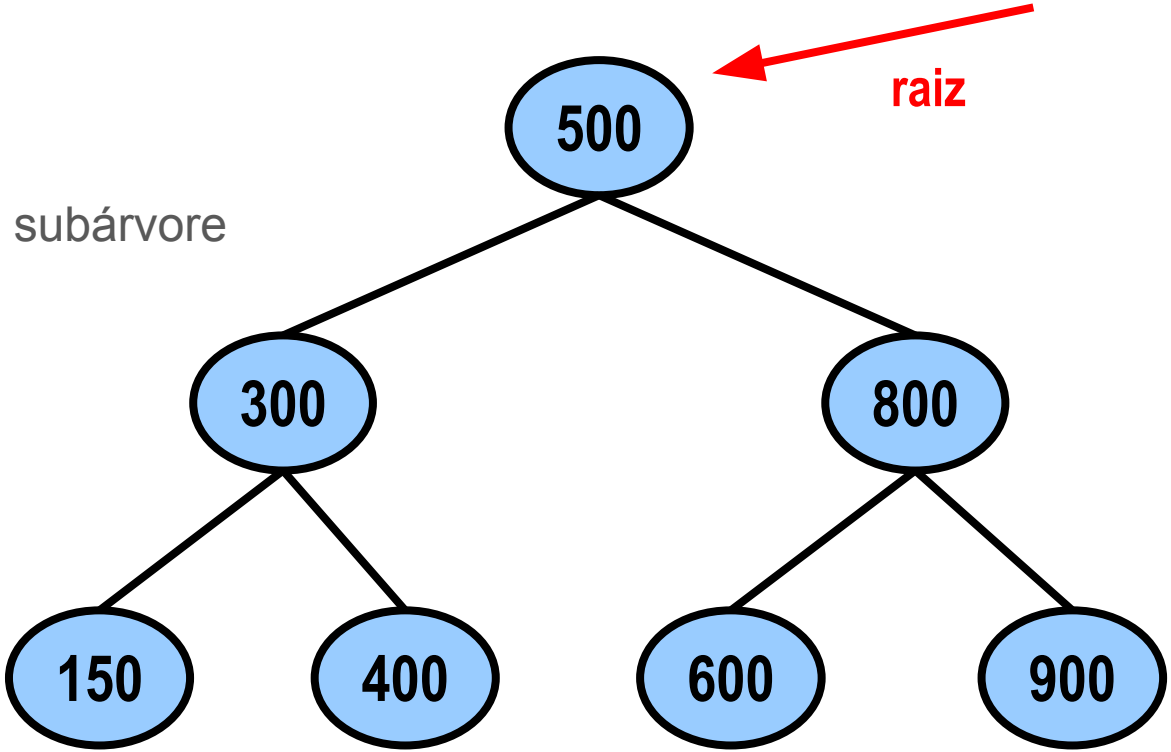
O nó da árvore binária de busca

```
typedef struct sNoA {
    char info;
    int chave;
    struct sNoA* esq;
    struct sNoA* dir;
} TNoA;
```



A organização da árvore binária de busca

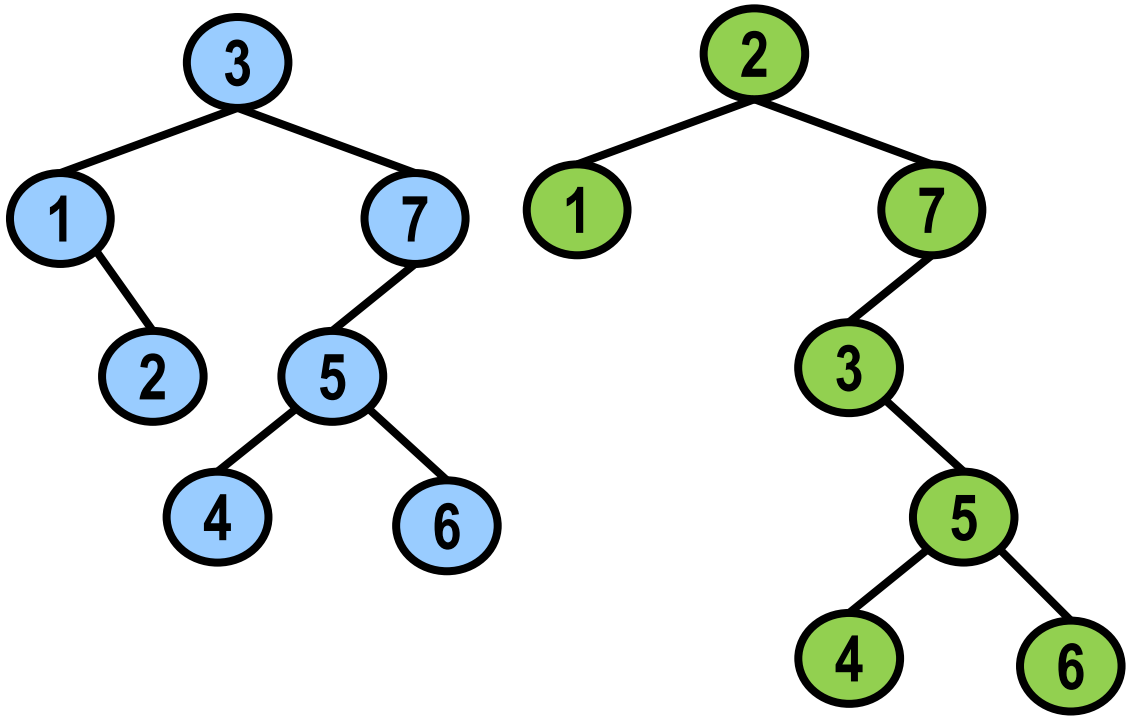
- Filhos da esquerda < raiz
- Filhos da direita > raiz
- A definição vale para toda subárvore



A organização da árvore binária de busca

- Mesmas chaves podem gerar árvores diferentes.

A busca será mais eficiente na árvore de menor altura.



Operações

- Buscar um nó
- Inserir um novo nó
- Remover um nó

-> As operações de inserção e remoção devem garantir que a árvore continue sendo uma árvore binária de busca

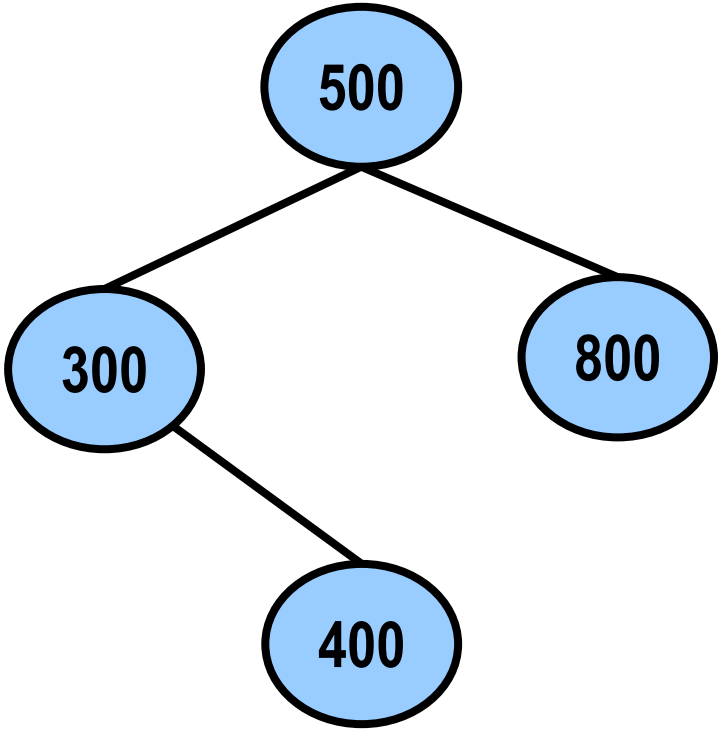
Inserção

- Se árvore vazia, insere na raiz
- Se não for vazia
  - Se chave < nó, vai pra esquerda
  - Se chave > nó, vai pra direita
- Quando achar um NULL, insere

Exemplo de inserção

Ordem de Inserção:

500 – 800 – 300 - 400



EXERCÍCIOS

1. Inserir em uma ABB inicialmente vazia, os seguintes valores:

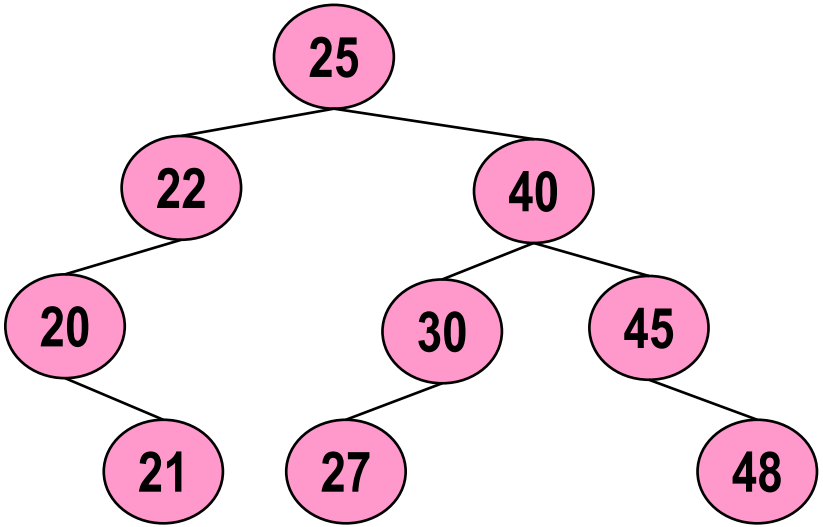
   25, 22, 40, 30, 45, 27, 20, 21, 48

2. Inserir em uma ABB inicialmente vazia, os seguintes valores:

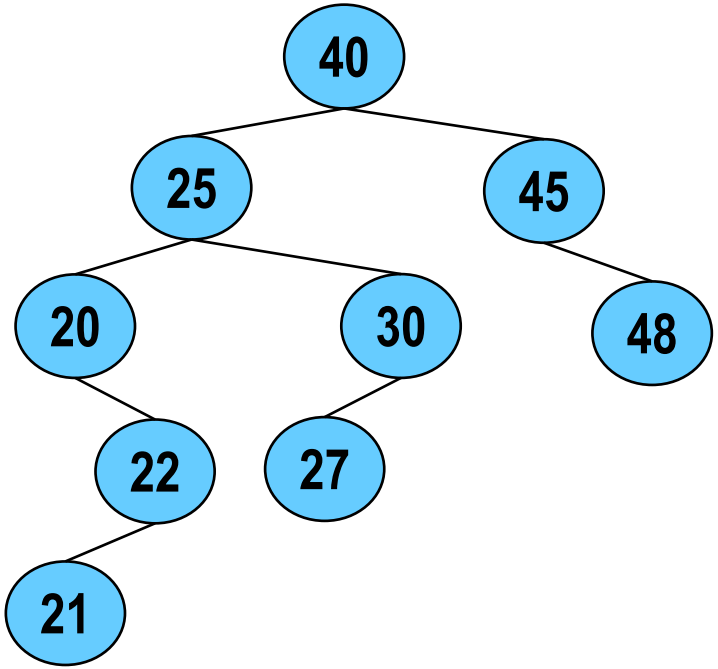
   40, 25, 20, 30, 45, 27, 22, 21, 48

INSERÇÃO

25 22 40 30 45 27 20 21 48



40 25 20 30 45 27 22 21 48



Inserção - Implementação (código insere.c)

```
TNoA *insere(TNoA *no, int chave) {
    if (no == NULL) {
        no = (TNoA *) malloc(sizeof(TNoA));
        no->chave = chave;
        no->esq = NULL;
        no->dir = NULL;
    } else if (chave < (no->chave))
              no->esq = insere(no->esq, chave);
           else if (chave > (no->chave))
                   no->dir = insere(no->dir, chave);
                else {
                   printf("Inserção inválida! "); // chave já existe
                   exit(1);
                }
    return no;
}
```

Criando uma árvore balanceada

Podemos criar a árvore já balanceada

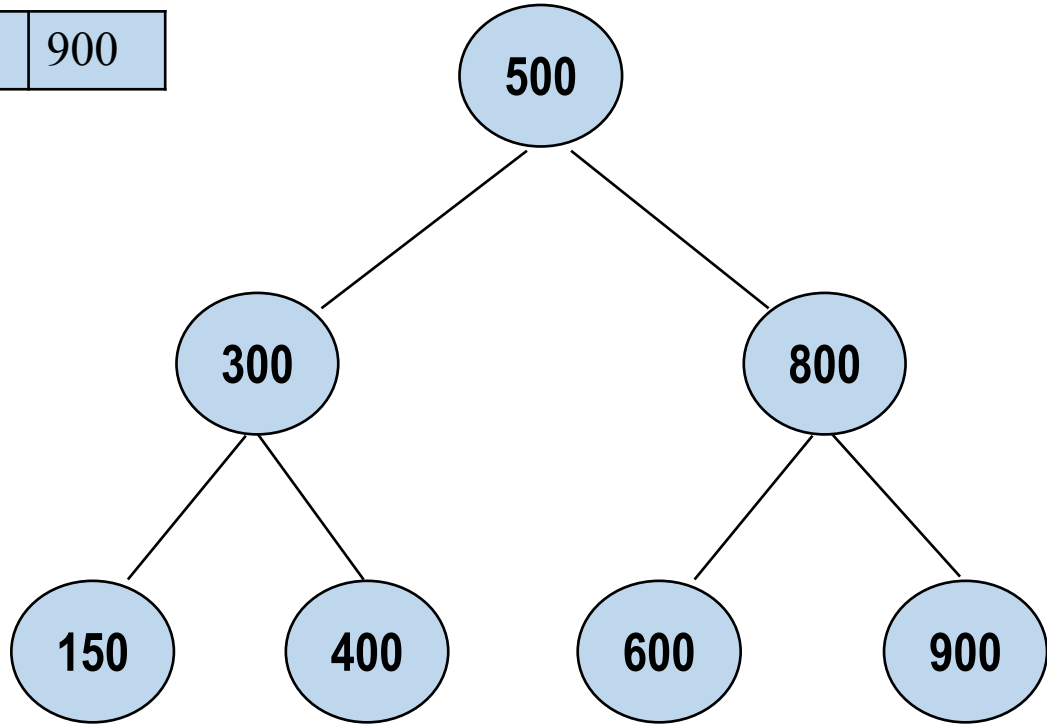
Ordenar as chaves em um vetor (assumimos que já está ordenado)

Criar um nó raiz com a chave central do vetor

Chamar recursivamente para os pedaços direito e esquerdo do vetor

Exemplo

150	300	400	500	600	800	900



Cria árvore balanceada (código balanceada.c)

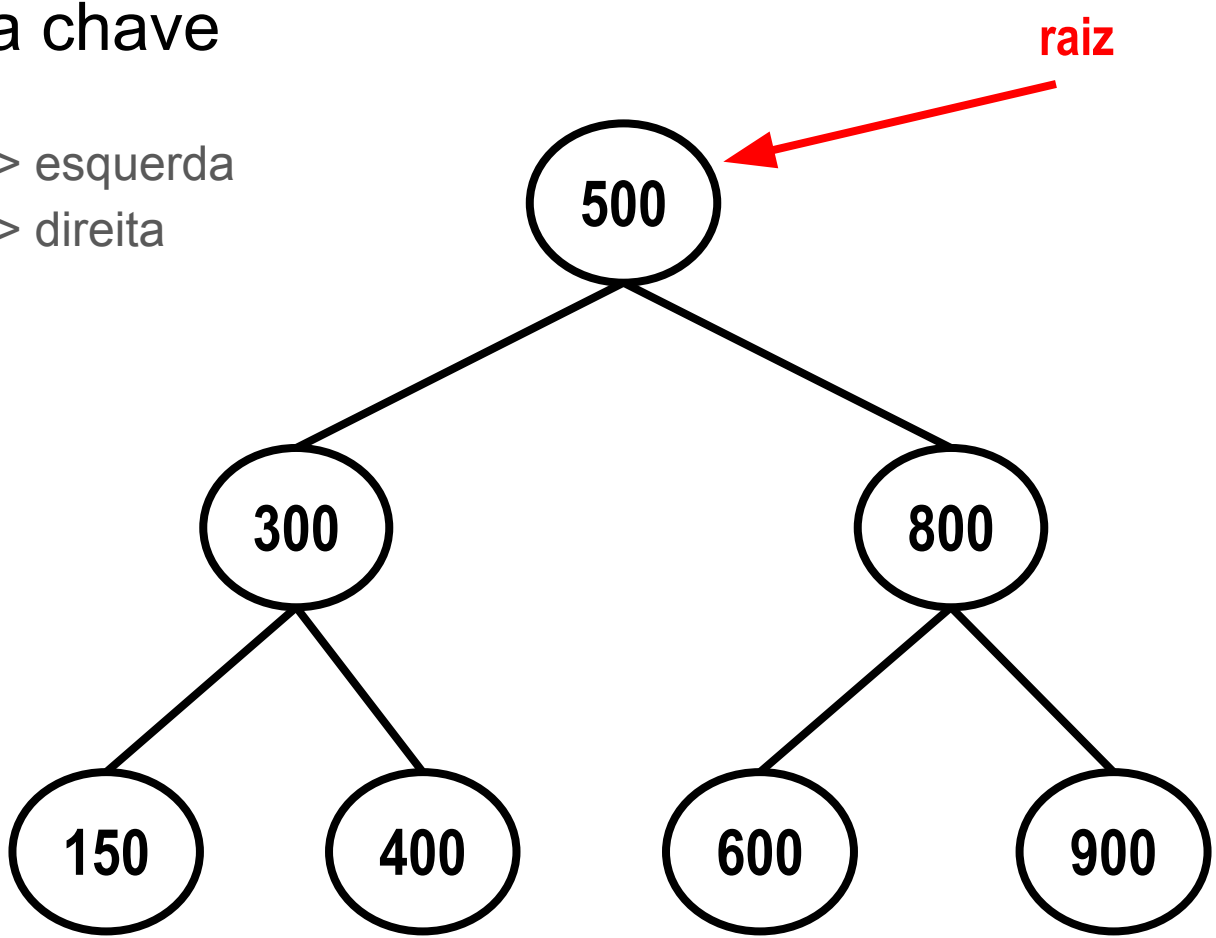
```
TNoA *criaArvoreBalanceada(TNoA *raiz, int *v, int inicio, int fim) {
    if (inicio <= fim) {
        int meio = (inicio + fim) / 2;
        if (raiz == NULL) {//se for primeiro nó a ser inserido tem que atualizar a
raiz da arvore
            raiz = insere(raiz, v[meio]);
        }
        else insere(raiz, v[meio]);
        //constroi subárvores esquerda e direita
        criaArvoreBalanceada(raiz, v, inicio, meio - 1);
        criaArvoreBalanceada(raiz, v, meio + 1, fim);
    }
    return raiz;
}


int main(void) {
    int tam = 7;
    int v[] = {150, 300, 400, 500, 600, 800, 900};
    TNoA *raiz;
    raiz = NULL;
    criaArvoreBalanceada(raiz,v,0,tam-1);
    imprime(raiz, 0);
};
```

Buscar por uma chave

- Se chave < nó => esquerda
- Se chave > nó => direita

- Exemplos:
  - Buscar por 600
  - Buscar por 200



Função de busca por um nó com uma dada chave
(código busca.c)

```
TNoA* busca(TNoA *no, int chave) {
   //Recebe endereço da raiz e chave procurada.
   //Se a chave é menor que o nó, vai pra esquerda
   //Se a chave é maior que o nó, vai pra direita
   //Retorna ponteiro pro nó encontrado
   //Ou retorna nulo
}
```

Busca - Implementação iterativa (busca_esqueleto.c)

```
TNoA* busca(TNoA *no, int chave) {
    TNoA *aux = no;
    while (aux != NULL) {
        if (aux->chave == chave )
            return aux; //achou
        else
        if (aux->chave > chave)
            aux = aux->esq;
        else
            aux = aux->dir;
    }
    return NULL; //não achou
}
```

Busca - Implementação recursiva (busca.c)

```
TNoA* buscaRecursiva(TNoA *no, int chave) {
    if (no == NULL)
        return NULL;
    else if (no->chave == chave)
        return no;
    else if (no->chave > chave)
        return buscaRecursiva(no->esq, chave);
    else
        return buscaRecursiva(no->dir, chave);
}
```

Complexidade da busca

- Qual é o pior caso da busca?
  -
- Qual é o custo no pior caso?
  -
- Como podemos melhorar o custo do pior caso?
  -

Complexidade da busca

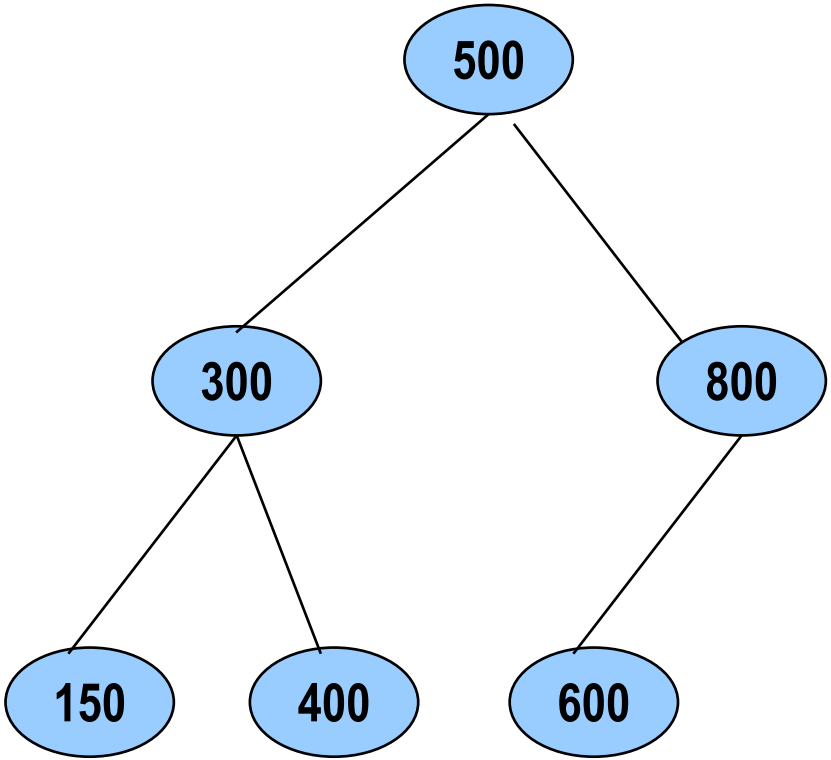
- Qual é o pior caso da busca?
  - Quando a chave é um nó folha
- Qual é o custo no pior caso?
  - Altura da árvore
- Como podemos melhorar o custo do pior caso?
  - Diminuir a altura usando uma árvore balanceada

O(log n)

Exclusão

3 possibilidades:

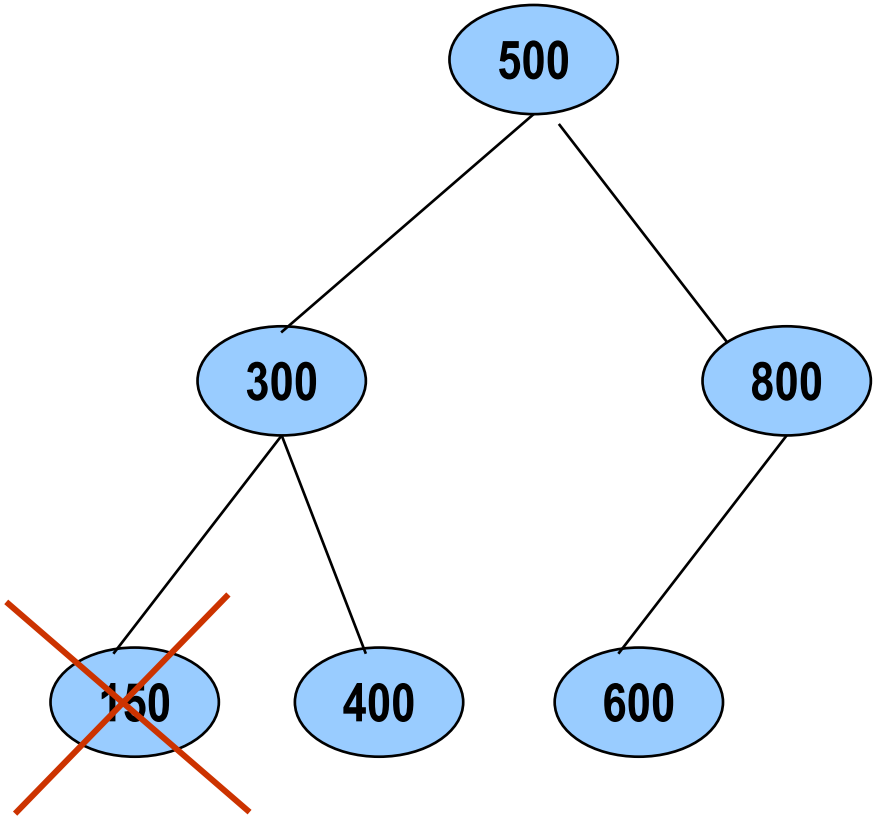
- O nó é folha
- O nó tem 1 sub-árvore
- O nó tem 2 subárvores



Exclusão

3 possibilidades:

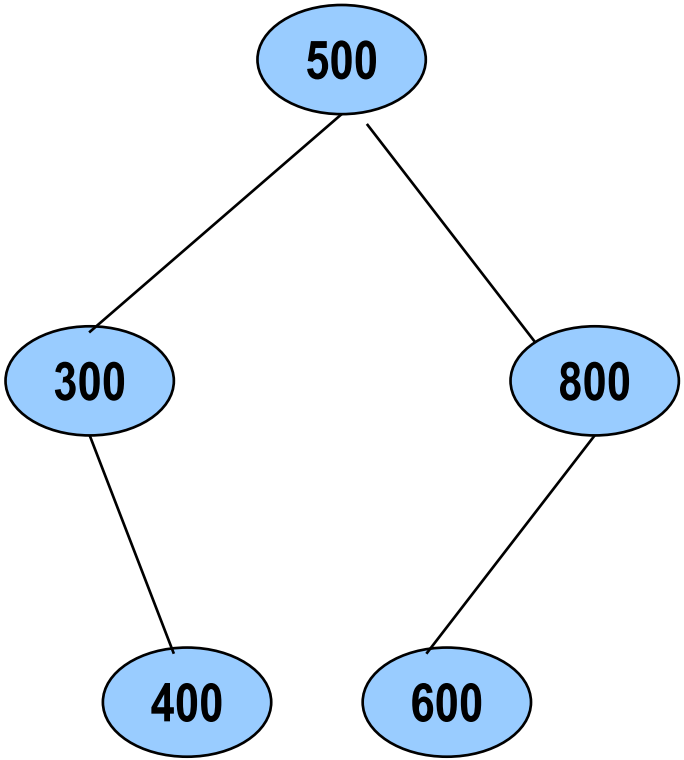
- **O nó é folha**
- O nó tem 1 sub-árvore
- O nó tem 2 subárvores



Exclusão

3 possibilidades:

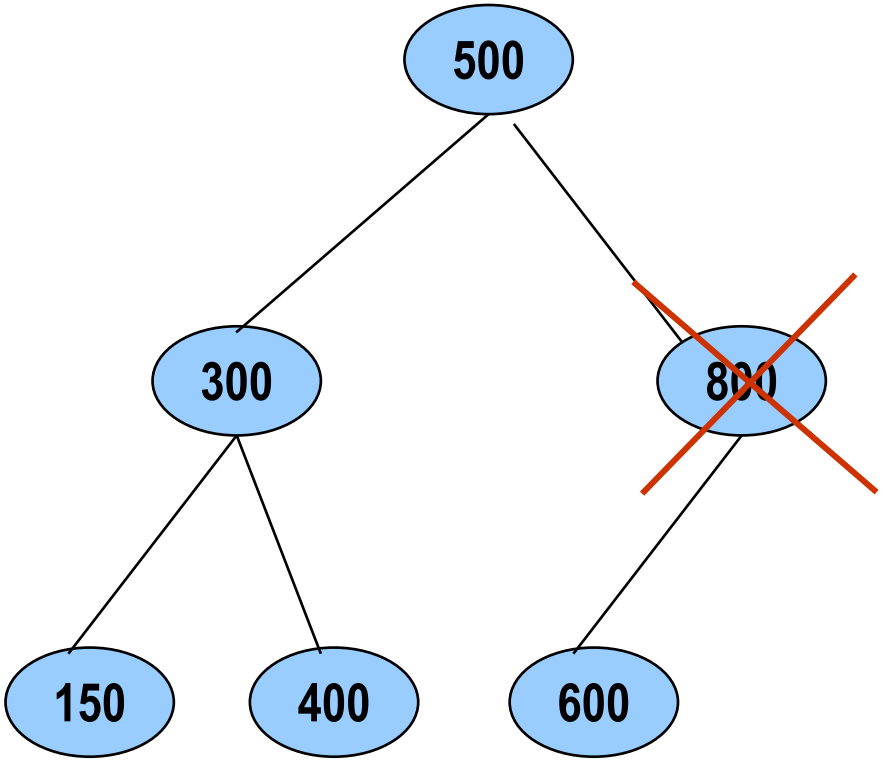
- **O nó é folha**
- O nó tem 1 sub-árvore
- O nó tem 2 subárvores



Exclusão

3 possibilidades:

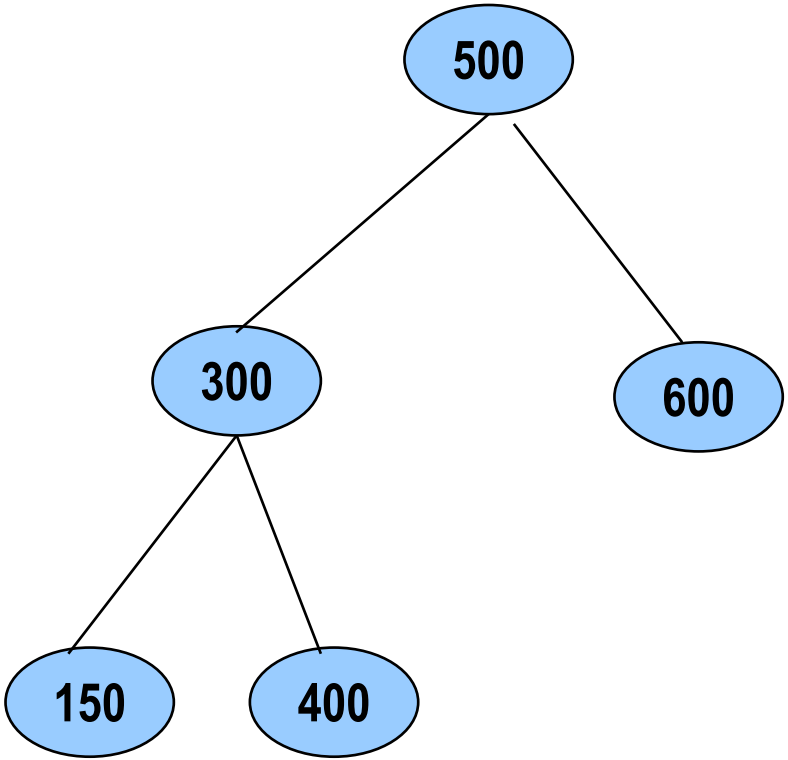
- O nó é folha
- **O nó tem 1 sub-árvore**
- O nó tem 2 subárvores



Exclusão

3 possibilidades:

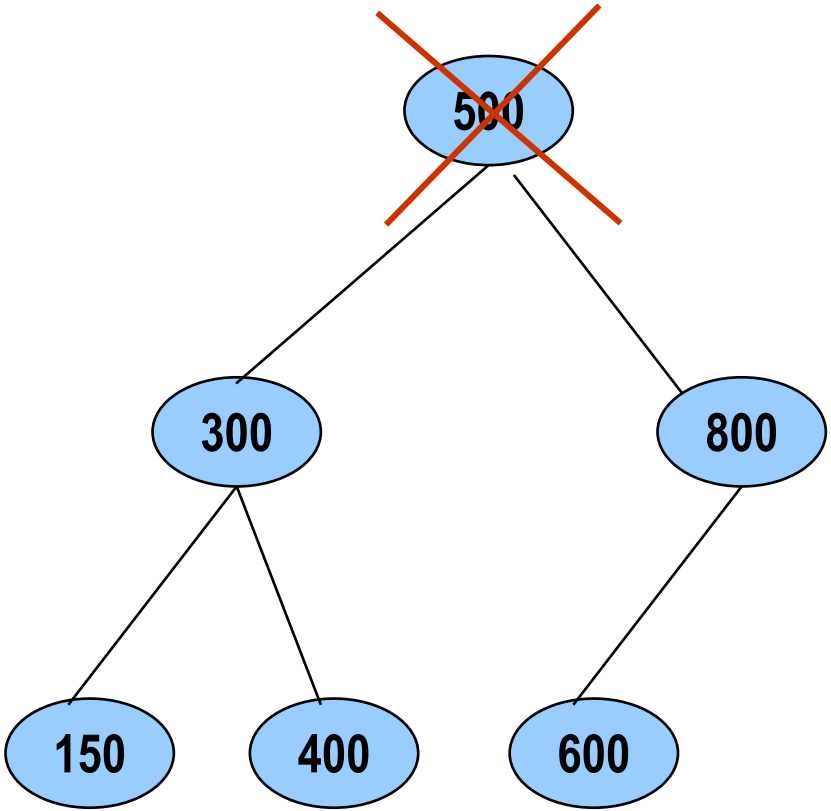
- O nó é folha
- **O nó tem 1 sub-árvore**
- O nó tem 2 subárvores



Exclusão

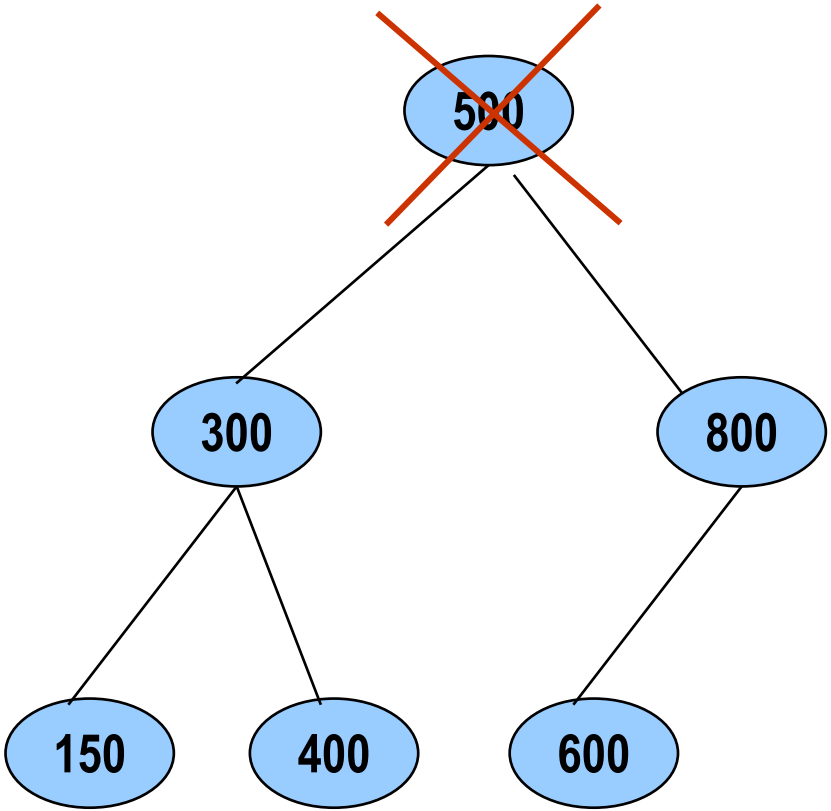
3 possibilidades:

- O nó é folha
- O nó tem 1 sub-árvore
- **O nó tem 2 subárvores**

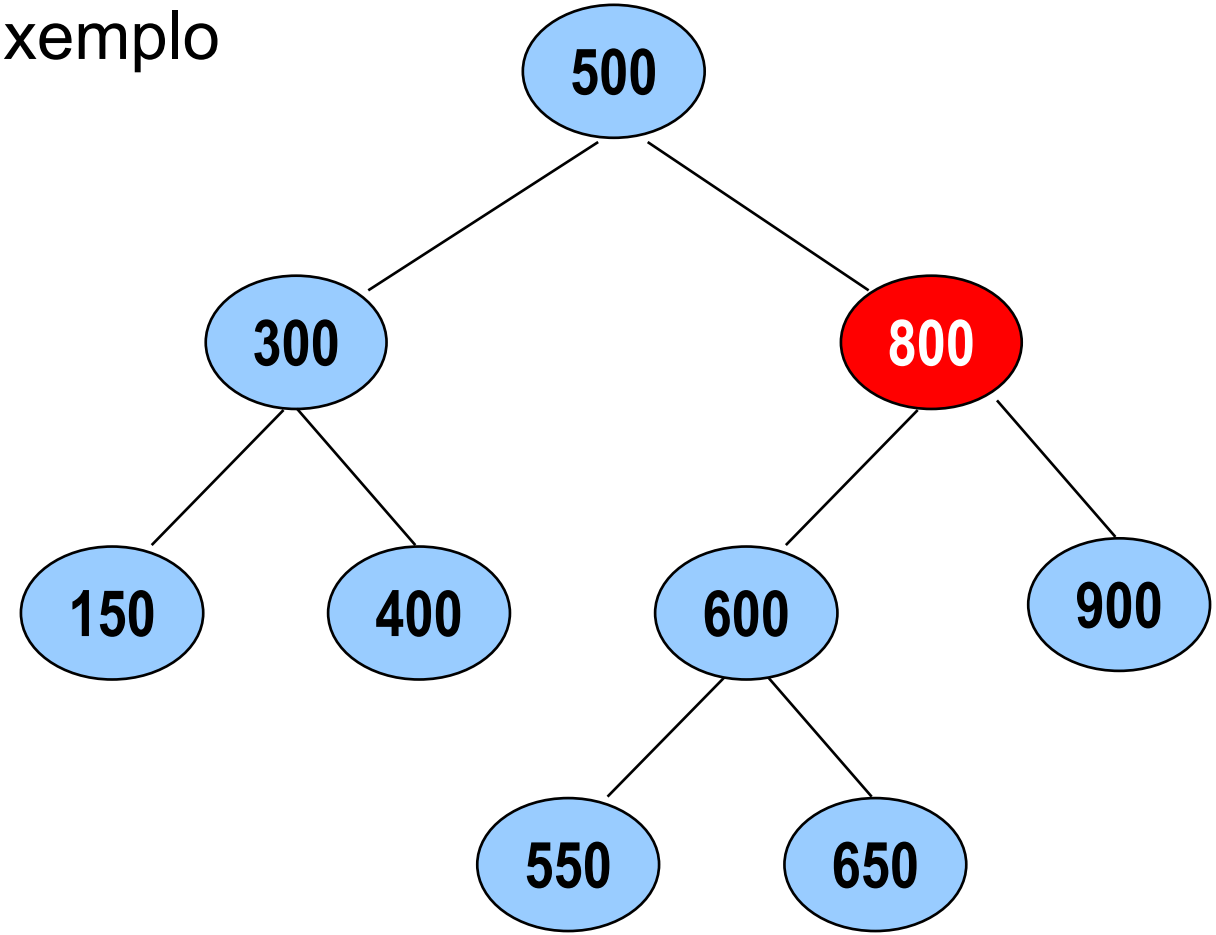


Exclusão

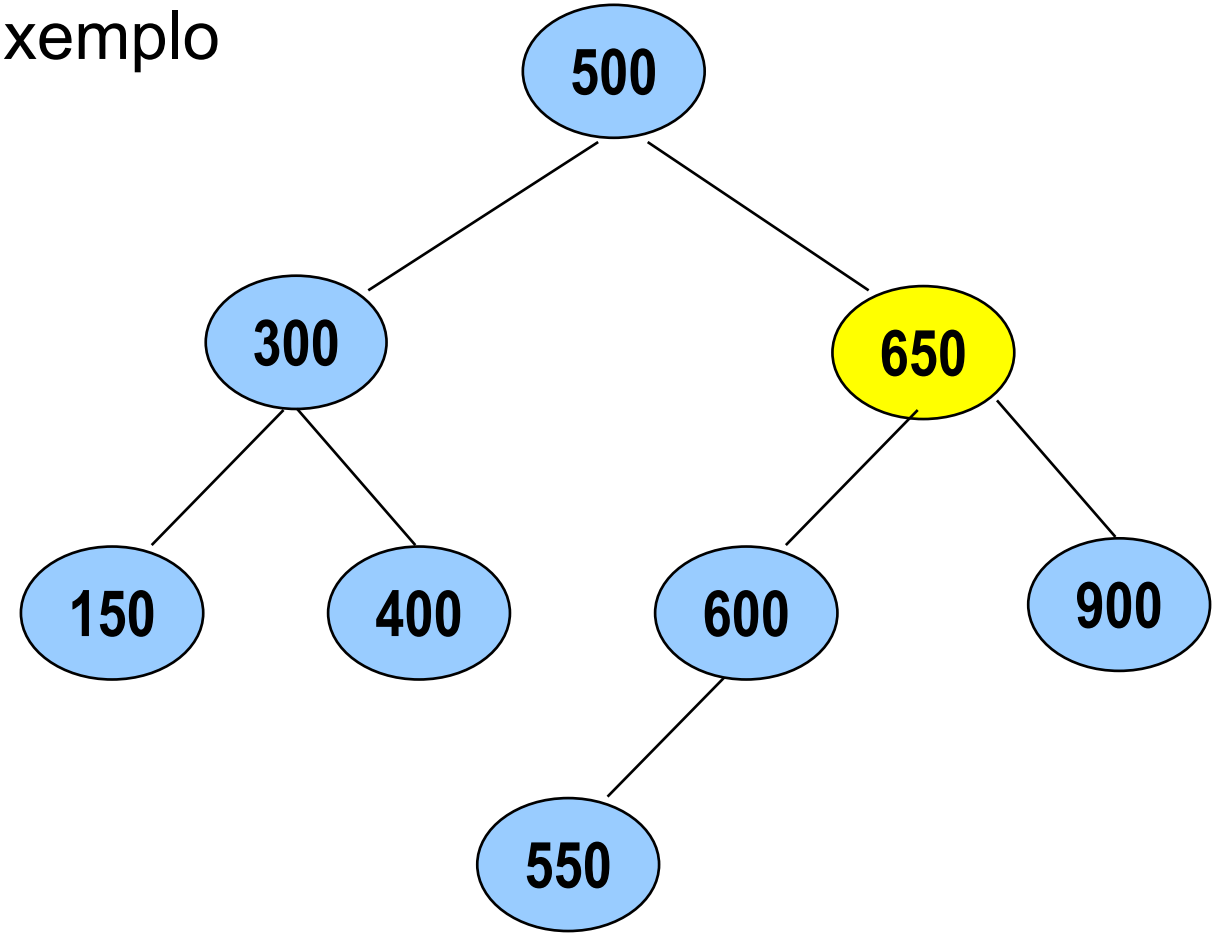
- Substituir o nó:
  - Maior chave da esquerda
  - Menor chave da direita
- Vamos adotar substituir a maior da esquerda



Exclusão - Exemplo



Exclusão - Exemplo

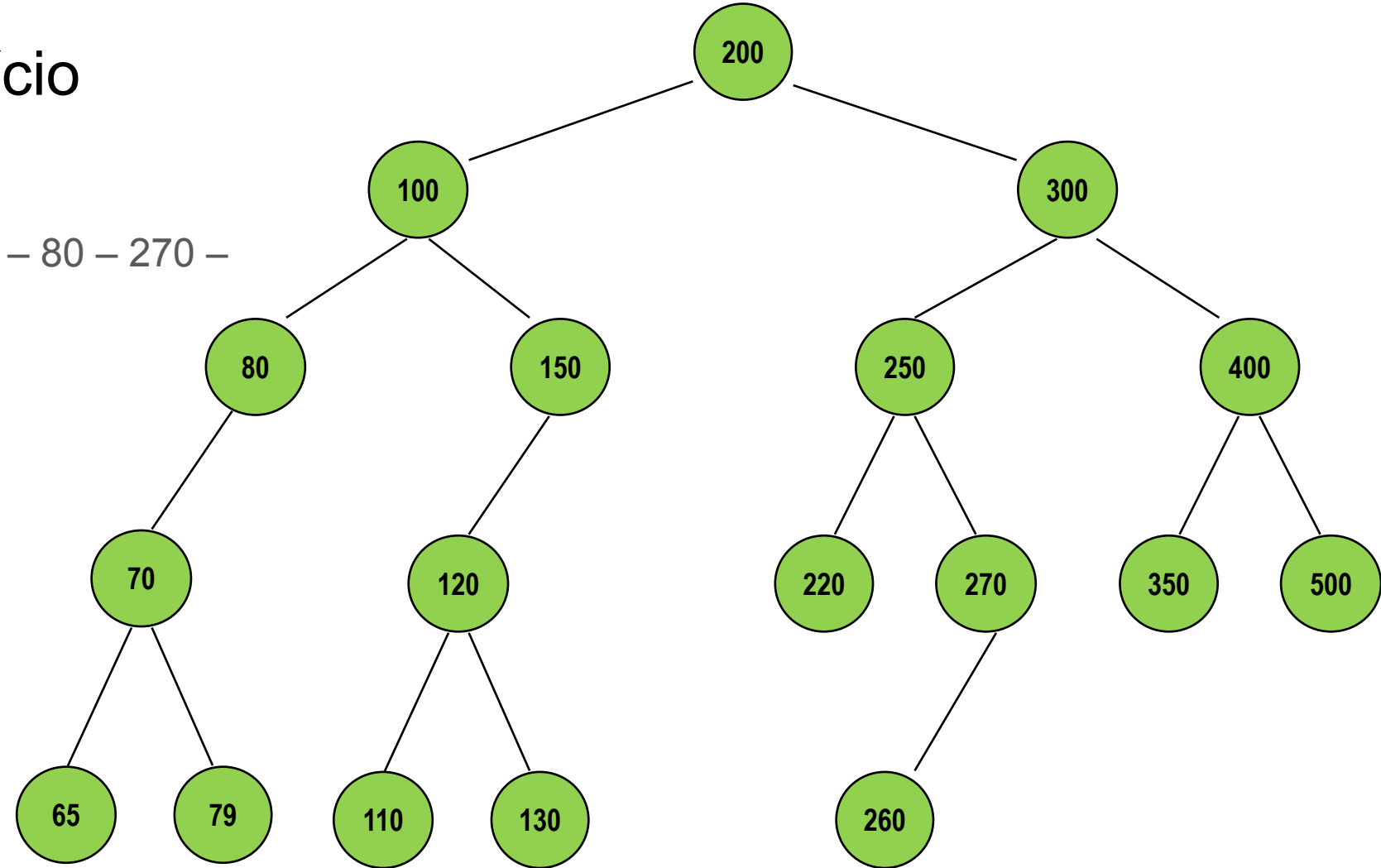


Exercício

Exercício

Excluir

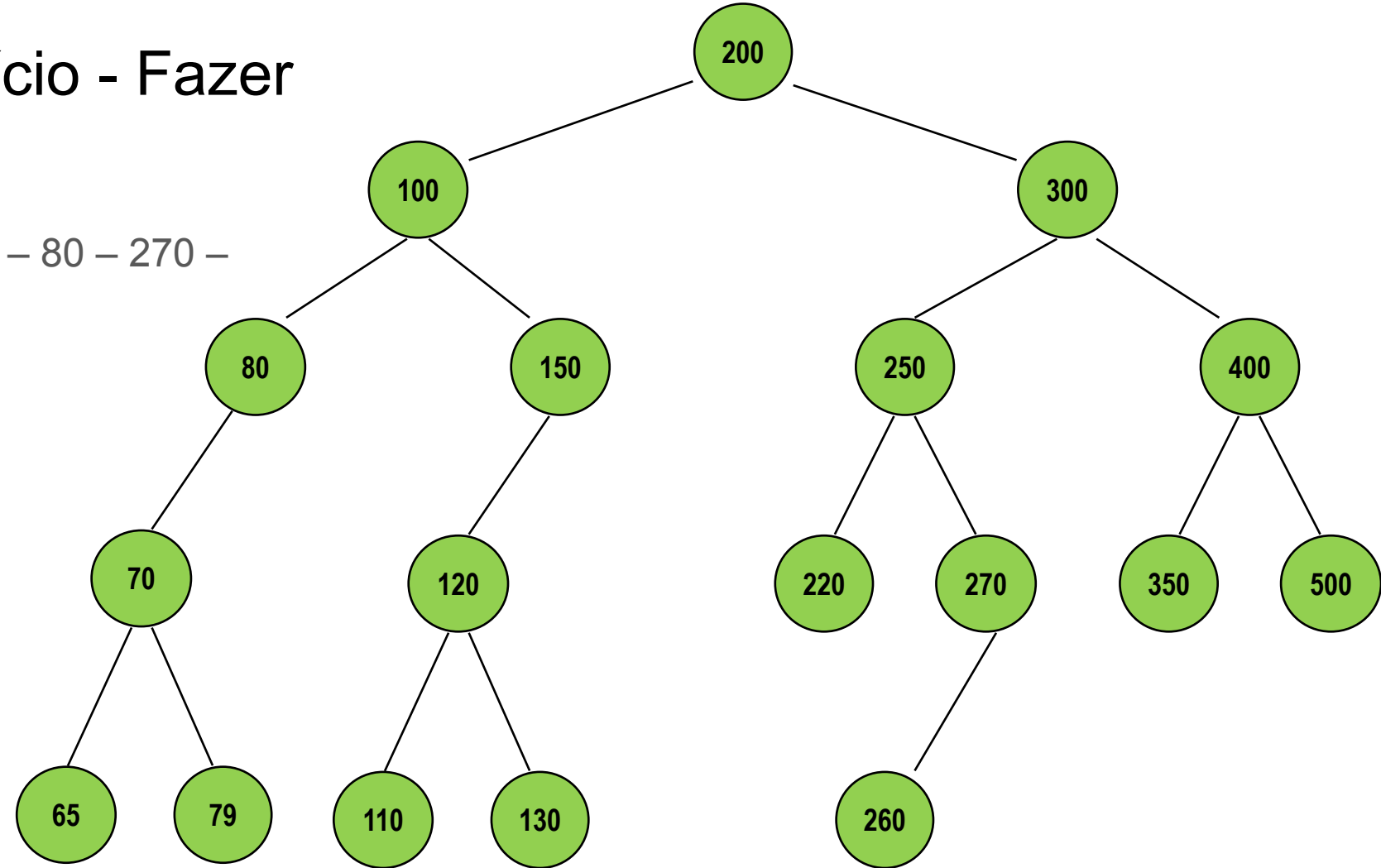
100 – 150 – 80 – 270 – 400 – 200



Exercício - Fazer

Excluir

100 – 150 – 80 – 270 – 400 – 200



Exclui - Pseudo-código

Assumir que a raiz é o nó a ser excluído neste ponto do código

Se só tem subárvore direita:

    Remove o nó raiz

    Retorna um ponteiro para o filho direito da raiz

Se só tem subárvore da esquerda:

    Remove o nó raiz

    Retorna um ponteiro para o filho esquerdo da raiz

Exclui - Pseudo-código

Se o nó tem 2 filhos:

temp aponta o maior nó da esquerda

Copia a chave do temp para a raiz

Recursivamente, manda excluir o nó temp na sub-árvore esquerda

Atualiza o filho esquerdo da raiz com a nova raiz da sub-árvore esquerda

Retorna a própria raiz

Provinha

Provinha 02: Horário Limite 17:20

Implementar uma função para excluir um nó que possui uma chave determinada em uma árvore binária de busca. A função deve retornar um ponteiro para a raiz da árvore.

Usar o código exclui_esqueleto.c que está disponível no google classroom.

O código deve imprimir a árvore antes e após a exclusão do nó.