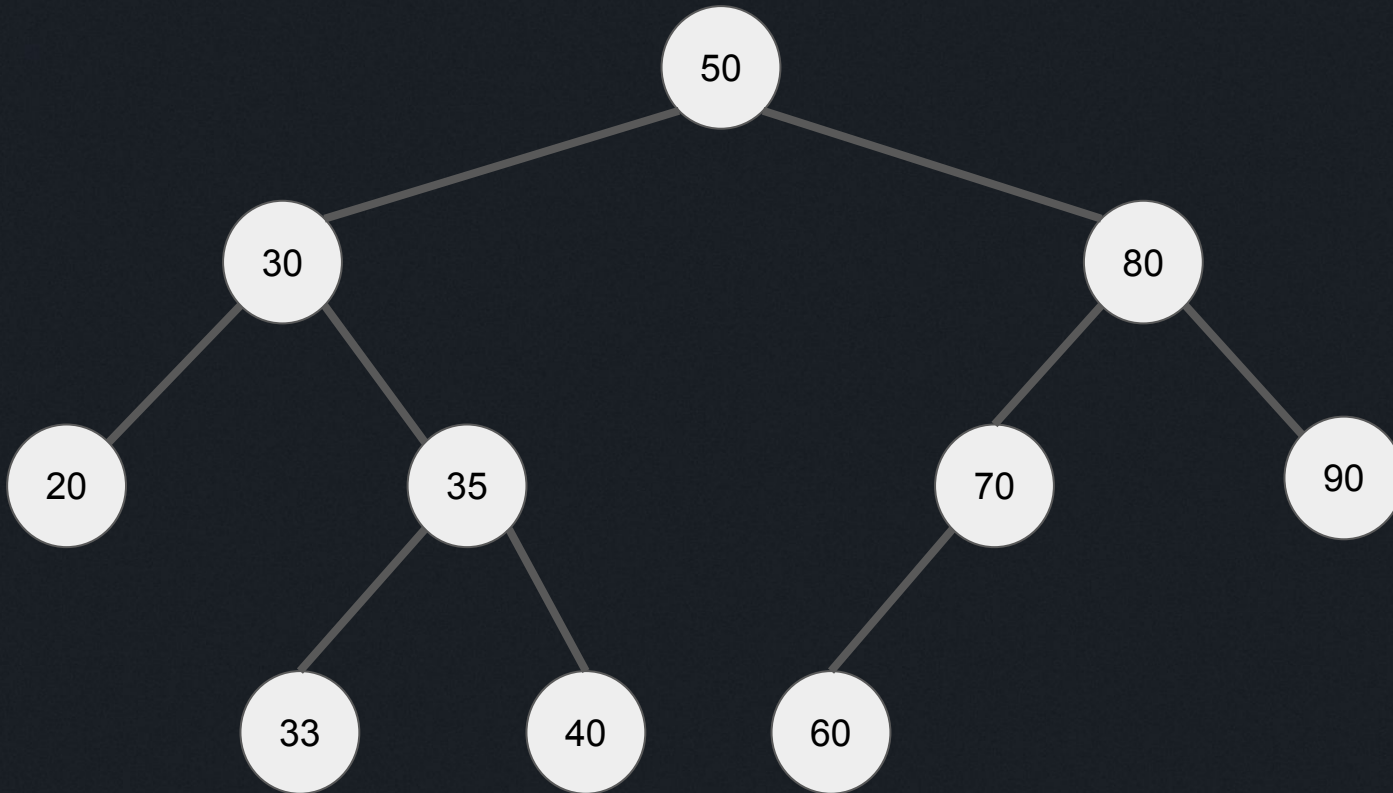




Árvores

Binary Search Tree - Sucessor

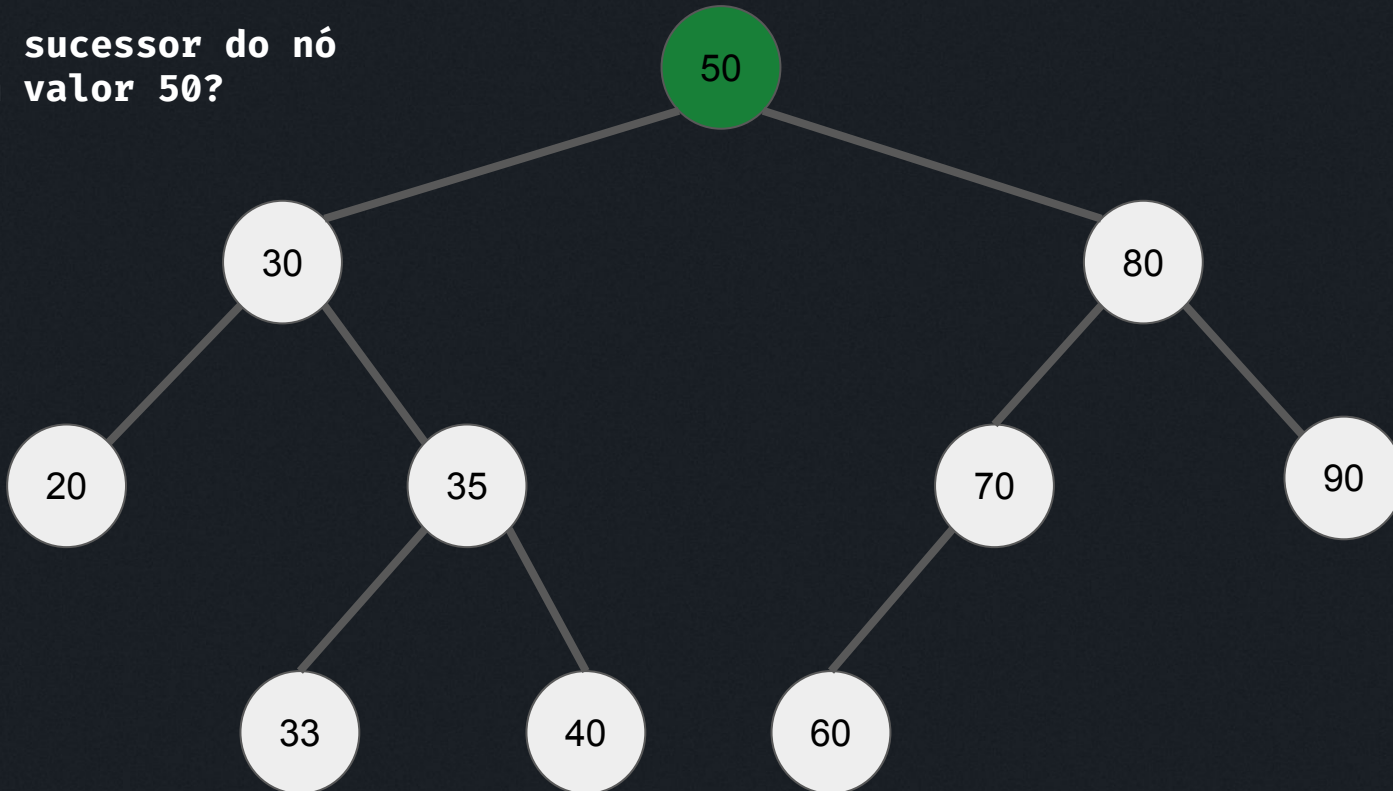
Dado um nó **N** em uma BST, o sucessor de **N** é definido com o nó com menor valor na BST que seja maior que o valor de **N**.



Binary Search Tree - Sucessor

Dado um nó **N** em uma BST, o sucessor de **N** é definido com o nó com menor valor na BST que seja maior que o valor de **N**.

Qual é o sucessor do nó com valor 50?

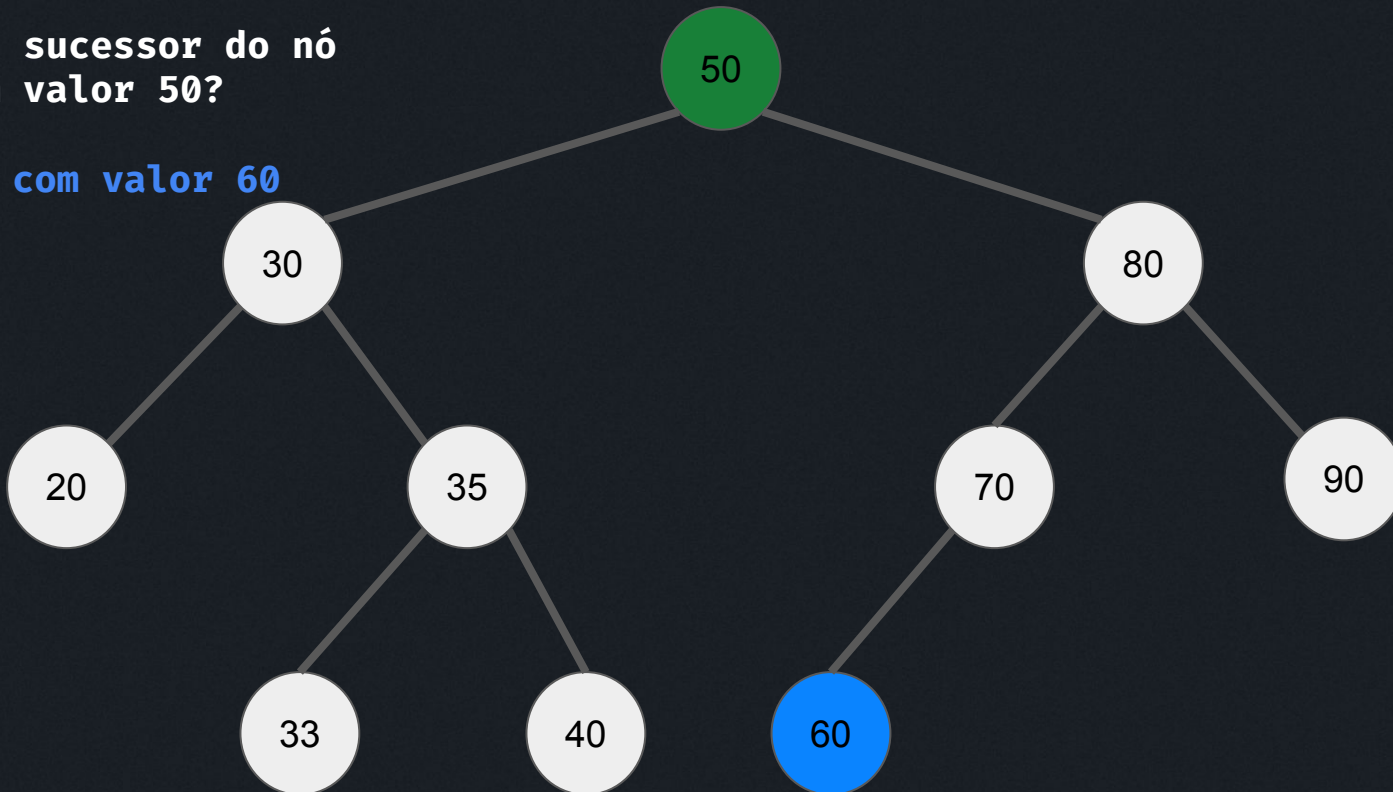


Binary Search Tree - Sucessor

Dado um nó **N** em uma BST, o sucessor de **N** é definido com o nó com menor valor na BST que seja maior que o valor de **N**.

Qual é o sucessor do nó com valor 50?

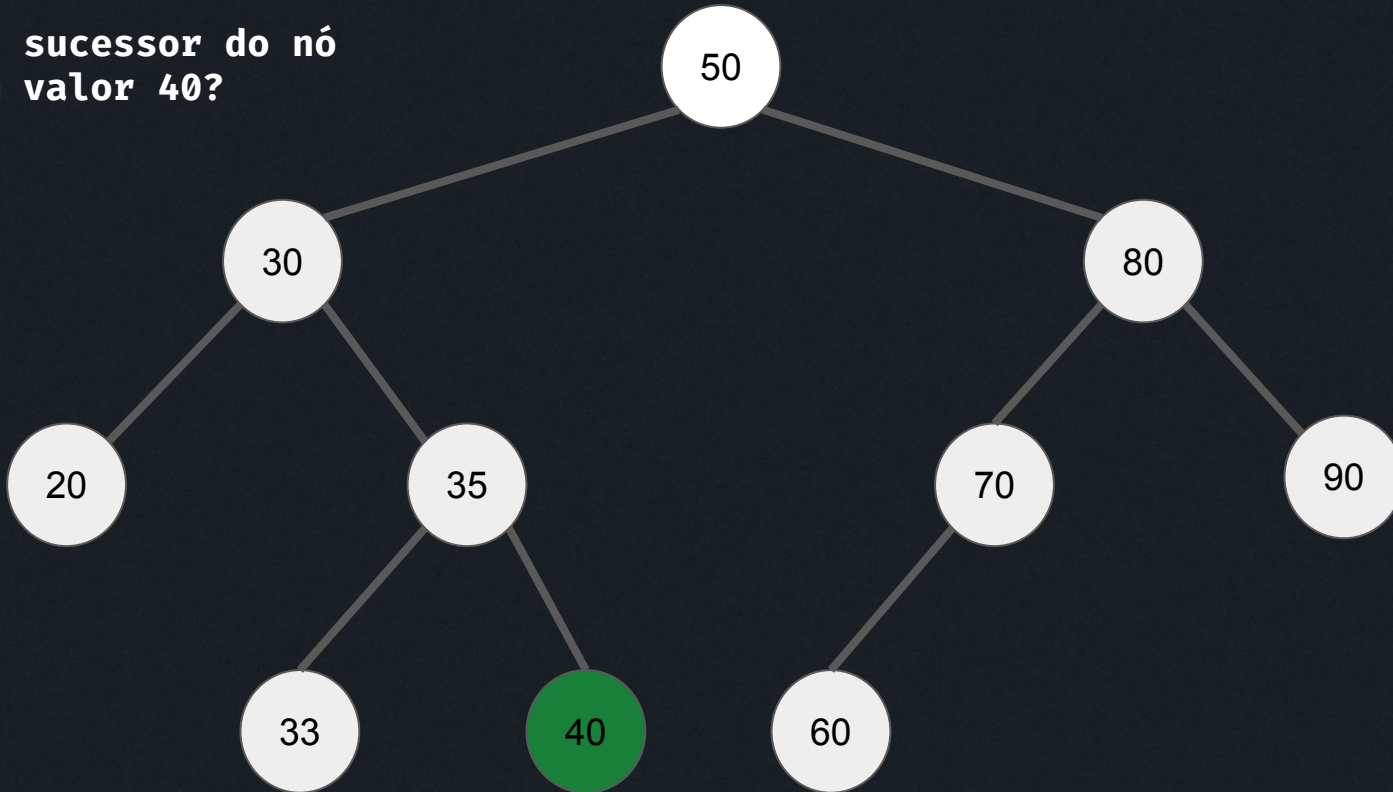
R: Nó com valor 60



Binary Search Tree - Sucessor

Dado um nó **N** em uma BST, o sucessor de **N** é definido com o nó com menor valor na BST que seja maior que o valor de **N**.

Qual é o sucessor do nó com valor 40?

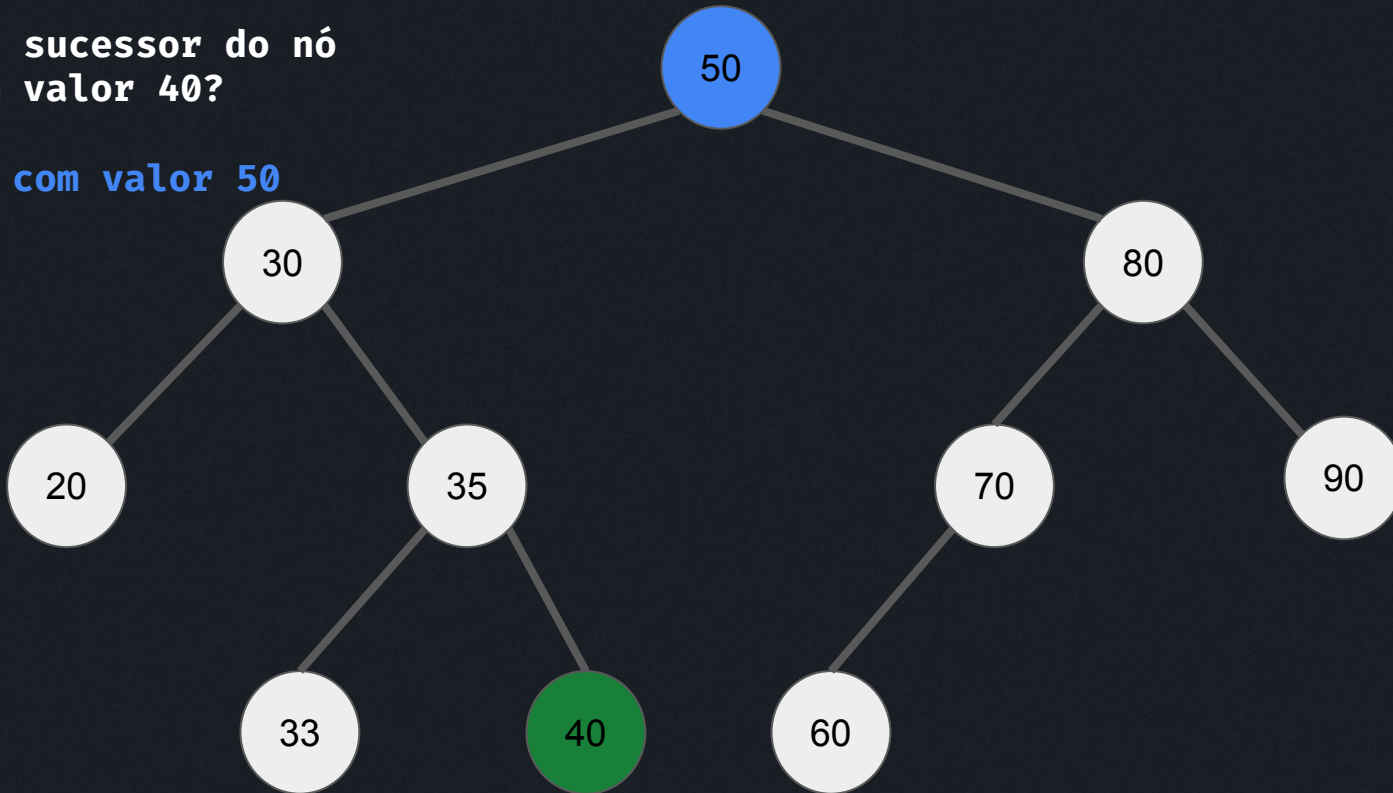


Binary Search Tree - Sucessor

Dado um nó **N** em uma BST, o sucessor de **N** é definido com o nó com menor valor na BST que seja maior que o valor de **N**.

Qual é o sucessor do nó com valor 40?

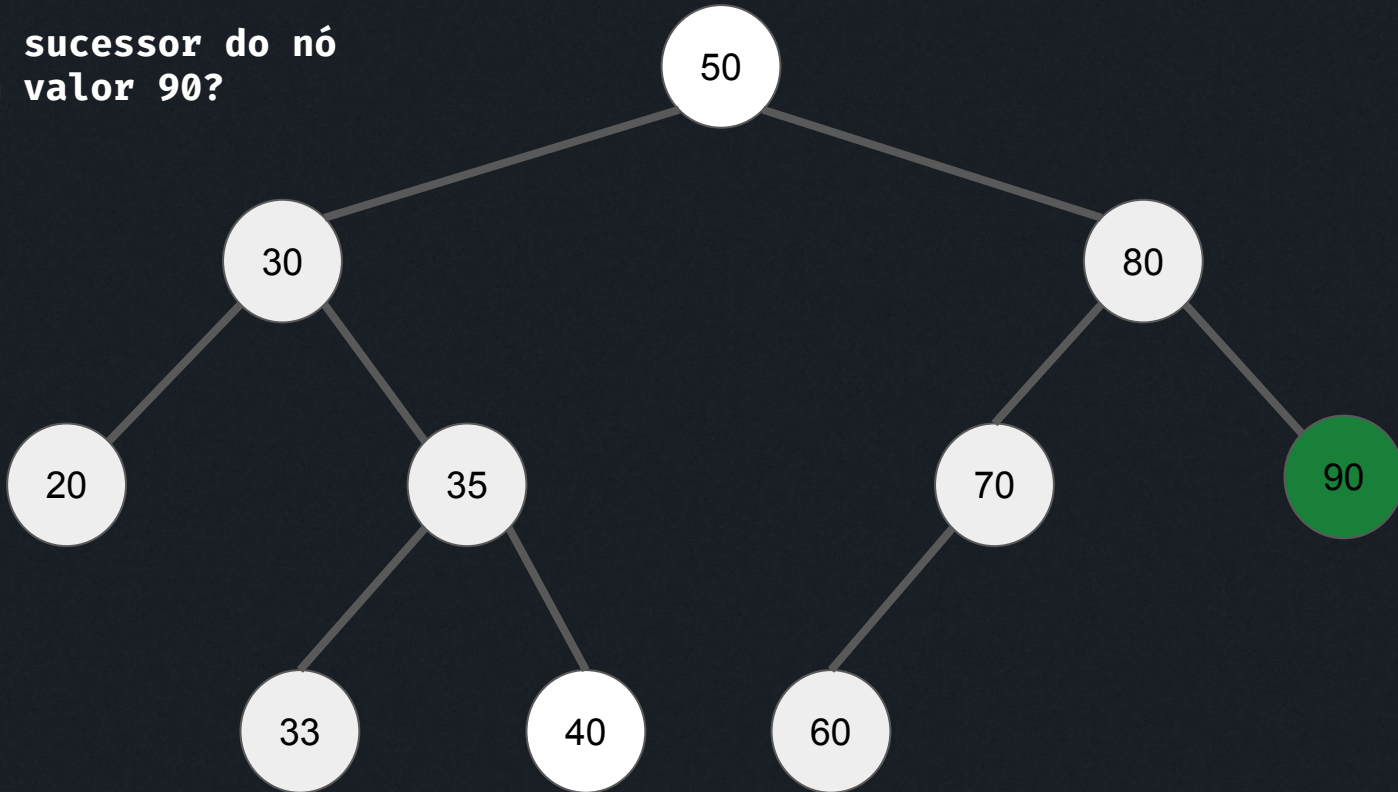
R: Nó com valor 50



Binary Search Tree - Sucessor

Dado um nó **N** em uma BST, o sucessor de **N** é definido com o nó com menor valor na BST que seja maior que o valor de **N**.

Qual é o sucessor do nó com valor 90?

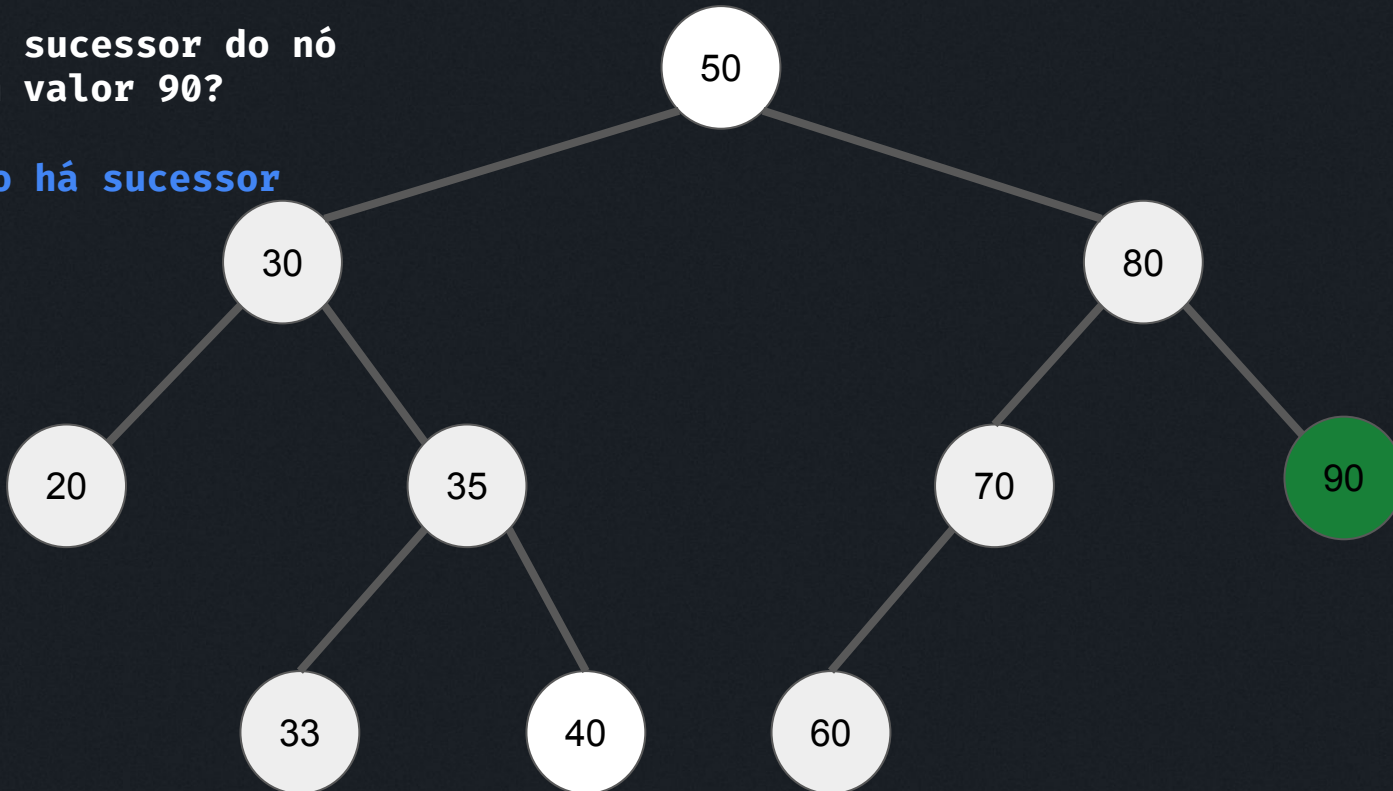


Binary Search Tree - Sucessor

Dado um nó **N** em uma BST, o sucessor de **N** é definido com o nó com menor valor na BST que seja maior que o valor de **N**.

Qual é o sucessor do nó com valor 90?

R: Não há sucessor



Binary Search Tree - Sucessor

<https://www.geeksforgeeks.org/problems/inorder-successor-in-bst/1>

Inorder Successor in BST

Difficulty: Easy

Accuracy: 34.97%

Submissions: 124K+

Points: 2

Given a BST, and a reference to a Node x in the BST. Find the Inorder Successor of the given node in the BST.

Example 1:

Input:

```
    2
   / \
  1   3
```

K(data of x) = 2

Output: 3

Explanation:

Inorder traversal : 1 2 3

Hence, inorder successor of 2 is 3.

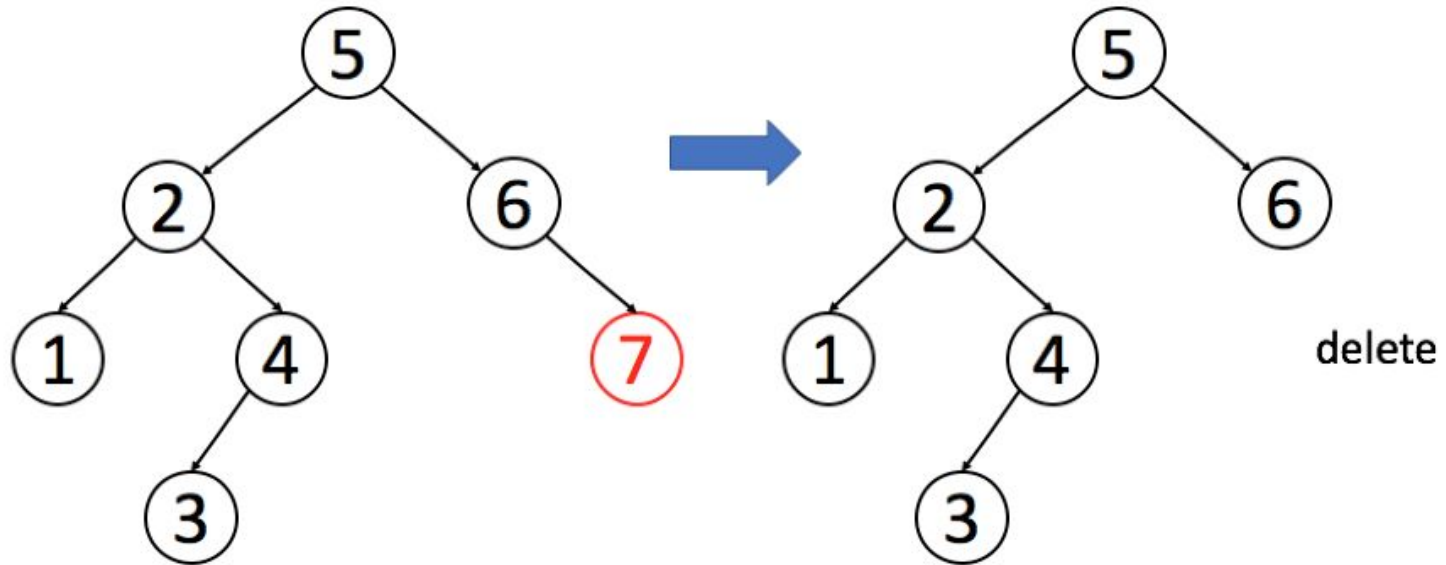
Binary Search Tree - Sucessor

<https://www.geeksforgeeks.org/problems/inorder-successor-in-bst/1>

```
def inorderSuccessor(self, root, x):  
    successor = None  
    while root:  
        if root.data > x.data:  
            successor = root  
            root = root.left  
        else:  
            root = root.right  
  
    return successor
```

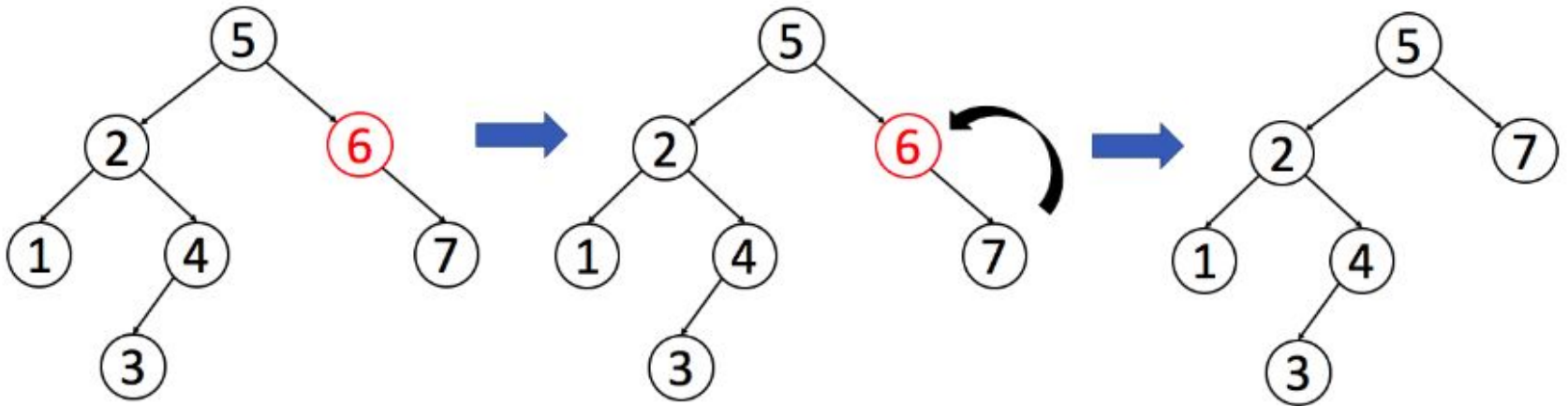
Binary Search Tree - Remove

Case 1: No Child



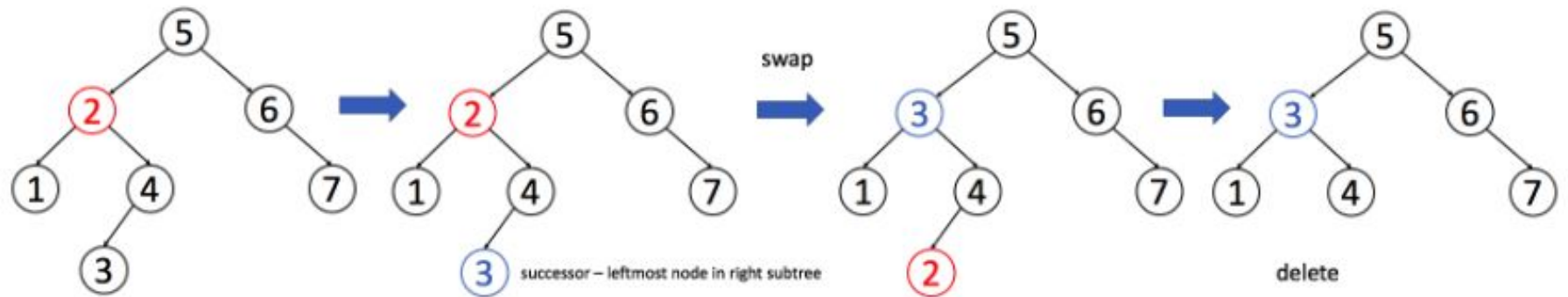
Binary Search Tree - Remove

Case 2: One Child



Binary Search Tree - Remove

Case 3: Two Children





Binary Search Tree - Remove

<https://leetcode.com/problems/delete-node-in-a-bst/description/>

450. Delete Node in a BST

Solved 

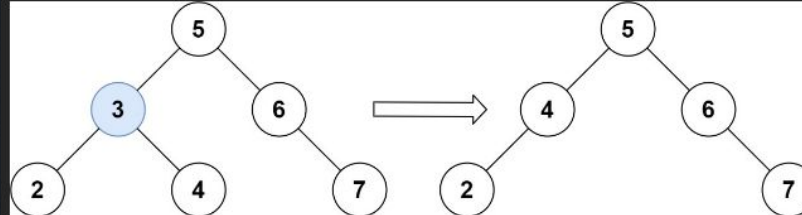
Medium  Topics  Companies

Given a root node reference of a BST and a key, delete the node with the given key in the BST. Return the **root node reference** (possibly updated) of the BST.

Basically, the deletion can be divided into two stages:

1. Search for a node to remove.
2. If the node is found, delete the node.

Example 1:



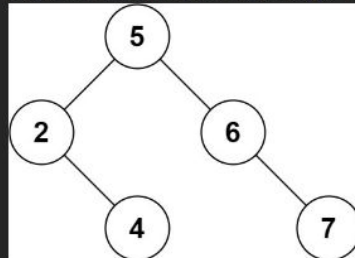
Input: root = [5,3,6,2,4,null,7], key = 3

Output: [5,4,6,2,null,null,7]

Explanation: Given key to delete is 3. So we find the node with value 3 and delete it.

One valid answer is [5,4,6,2,null,null,7], shown in the above BST.

Please notice that another valid answer is [5,2,6,null,4,null,7] and it's also accepted.



Binary Search Tree - Remove

<https://leetcode.com/problems/delete-node-in-a-bst/description/>

```
def deleteNode(self, root, key):
    if not root:
        return None

    if root.val > key:
        root.left = self.deleteNode(root.left, key)
    elif root.val < key:
        root.right = self.deleteNode(root.right, key)
    else: # root.val == key
        if not root.left:
            return root.right
        if not root.right:
            return root.left

        it = root.right
        while it.left:
            it = it.left

        root.val, it.val = it.val, root.val
        root.right = self.deleteNode(root.right, it.val)

    return root
```

Binary Search Tree - Remove

<https://leetcode.com/problems/delete-node-in-a-bst/description/>

```
if (root == null) return null;

if (key < root.val)
    root.left = deleteNode(root.left, key);
else if (root.val < key)
    root.right = deleteNode(root.right, key);
else { // root.val == key
    if (root.left == null)
        return root.right;
    if (root.right == null)
        return root.left;

    TreeNode it = root.right;
    while (it.left != null)
        it = it.left;

    root.val = it.val;
    root.right = deleteNode(root.right, it.val);
}
return root;
```


Binary Search Tree - Balanceamento

<https://leetcode.com/problems/balance-a-binary-search-tree/description/>

1382. Balance a Binary Search Tree

Solved 

Medium

Topics

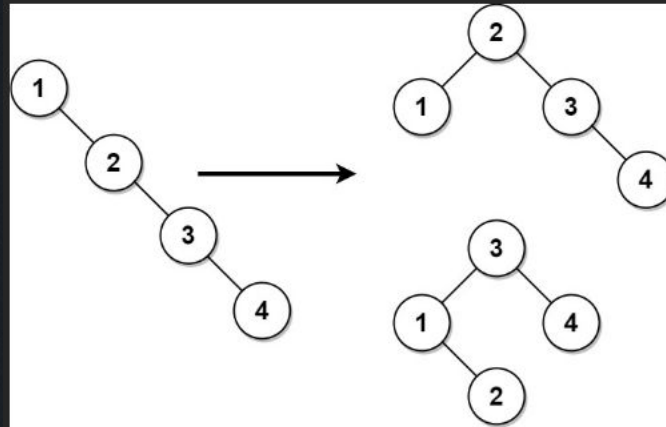
Companies

Hint

Given the `root` of a binary search tree, return a **balanced** binary search tree with the same node values. If there is more than one answer, return **any of them**.

A binary search tree is **balanced** if the depth of the two subtrees of every node never differs by more than 1.

Example 1:



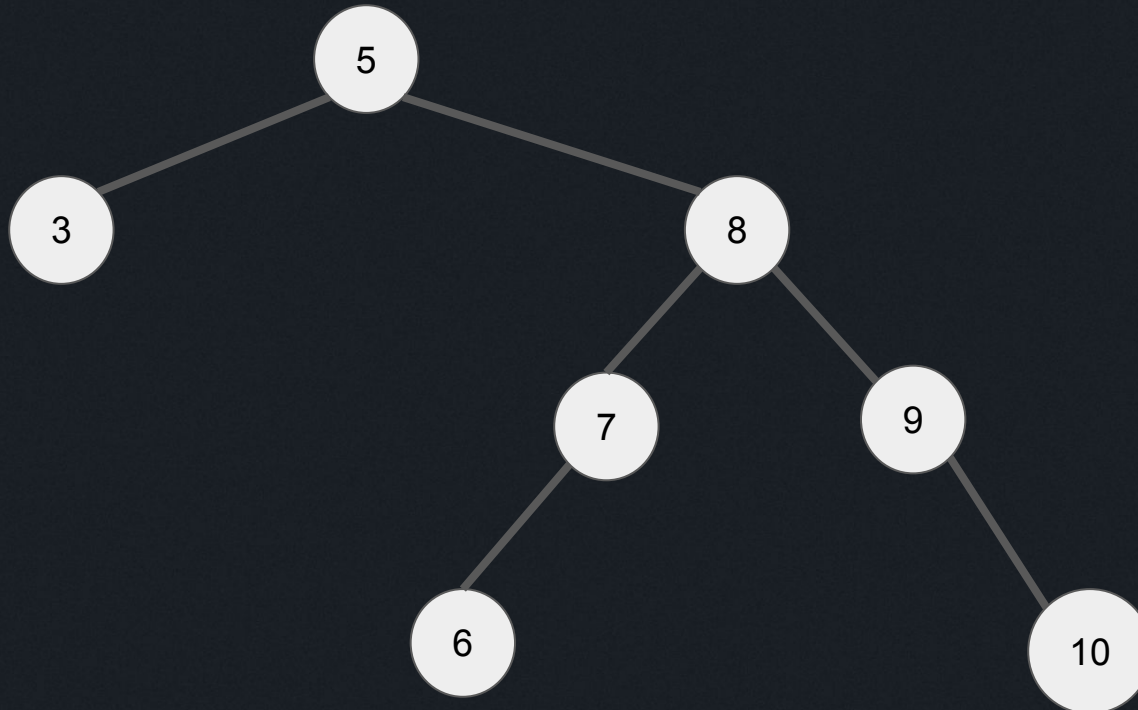
Input: `root = [1,null,2,null,3,null,4,null,null]`

Output: `[2,1,3,null,null,null,4]`

Explanation: This is not the only correct answer, `[3,1,4,null,2]` is also correct.

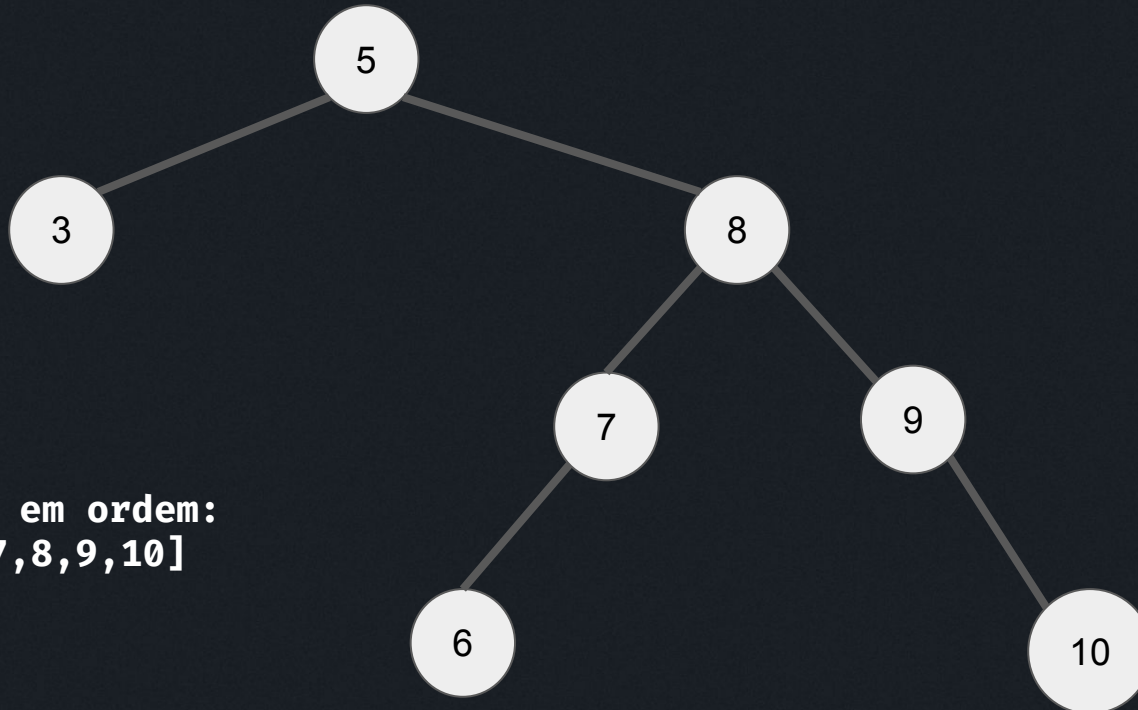
Binary Search Tree - Balanceamento

Dado a raiz de uma BST, se fizermos uma travessia em ordem armazenando os valores em um array, ao final da travessia o array terá todos os elementos da BST **ordenados**



Binary Search Tree - Balanceamento

Dado a raiz de uma BST, se fizermos uma travessia em ordem armazenando os valores em um array, ao final da travessia o array terá todos os elementos da BST **ordenados**



Travessia em ordem:
[3,5,6,7,8,9,10]

Binary Search Tree - Balanceamento

Dado o array ordenado, podemos construir uma BST balanceada de forma recursiva

Travessia em ordem:
[3,5,6,7,8,9,10]

Binary Search Tree - Balanceamento

Dado o array ordenado, podemos construir uma BST balanceada de forma recursiva

Travessia em ordem:
[3,5,6,7,8,9,10]



Binary Search Tree - Balanceamento

Dado o array ordenado, podemos construir uma BST balanceada de forma recursiva

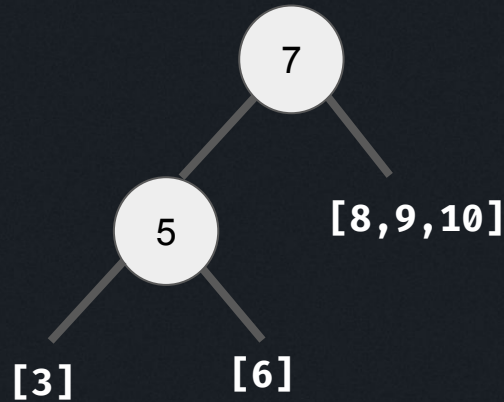
Travessia em ordem:
[3,5,6,7,8,9,10]



Binary Search Tree - Balanceamento

Dado o array ordenado, podemos construir uma BST balanceada de forma recursiva

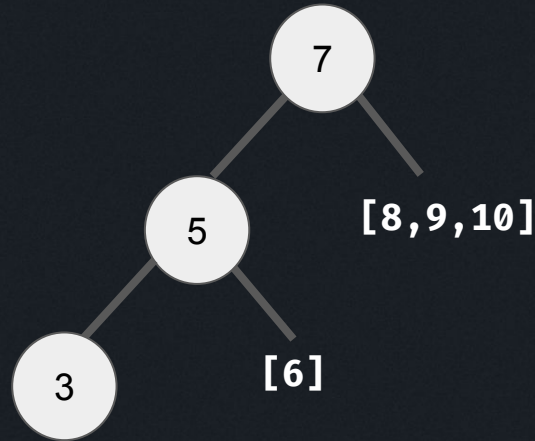
Travessia em ordem:
[3,5,6,7,8,9,10]



Binary Search Tree - Balanceamento

Dado o array ordenado, podemos construir uma BST balanceada de forma recursiva

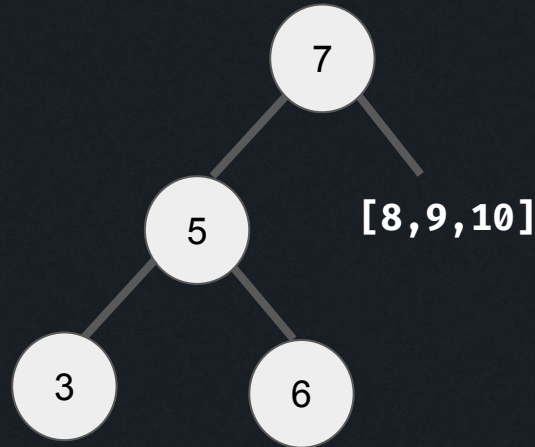
Travessia em ordem:
[3,5,6,7,8,9,10]



Binary Search Tree - Balanceamento

Dado o array ordenado, podemos construir uma BST balanceada de forma recursiva

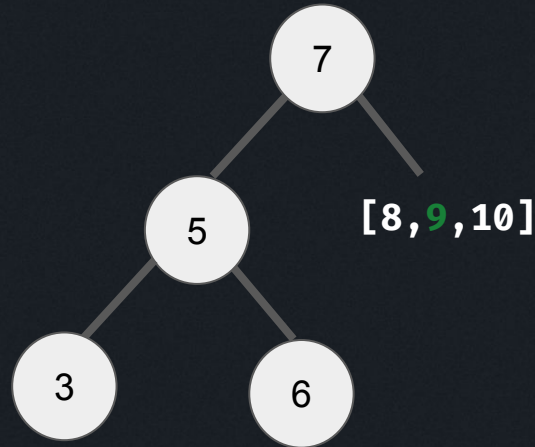
Travessia em ordem:
[3,5,6,7,8,9,10]



Binary Search Tree - Balanceamento

Dado o array ordenado, podemos construir uma BST balanceada de forma recursiva

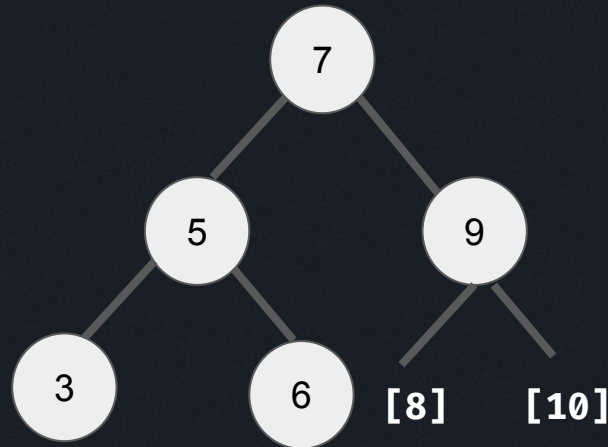
Travessia em ordem:
[3,5,6,7,8,9,10]



Binary Search Tree - Balanceamento

Dado o array ordenado, podemos construir uma BST balanceada de forma recursiva

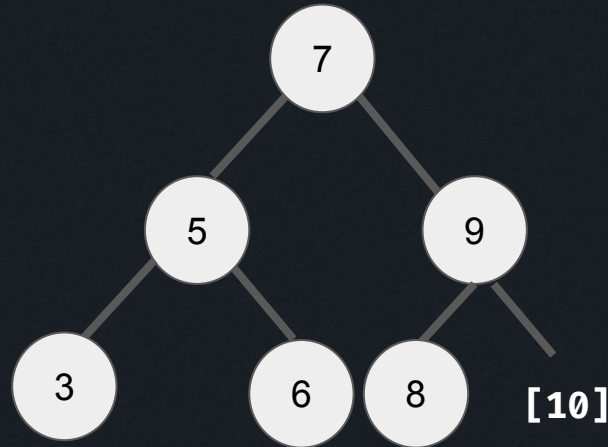
Travessia em ordem:
[3,5,6,7,8,9,10]



Binary Search Tree - Balanceamento

Dado o array ordenado, podemos construir uma BST balanceada de forma recursiva

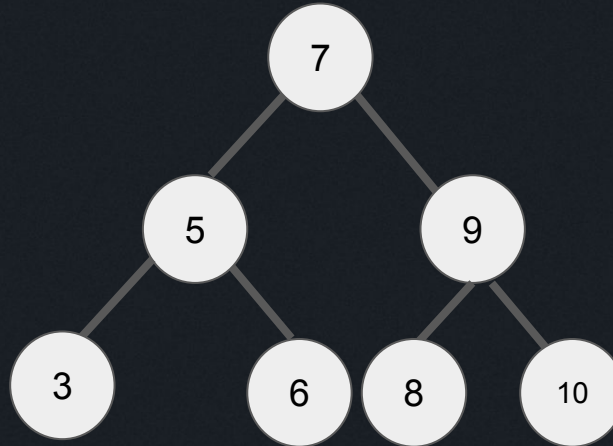
Travessia em ordem:
[3,5,6,7,8,9,10]



Binary Search Tree - Balanceamento

Dado o array ordenado, podemos construir uma BST balanceada de forma recursiva

Travessia em ordem:
[3,5,6,7,8,9,10]



Binary Search Tree - Balanceamento

<https://leetcode.com/problems/balance-a-binary-search-tree/description/>

1382. Balance a Binary Search Tree

Solved 

Medium

Topics

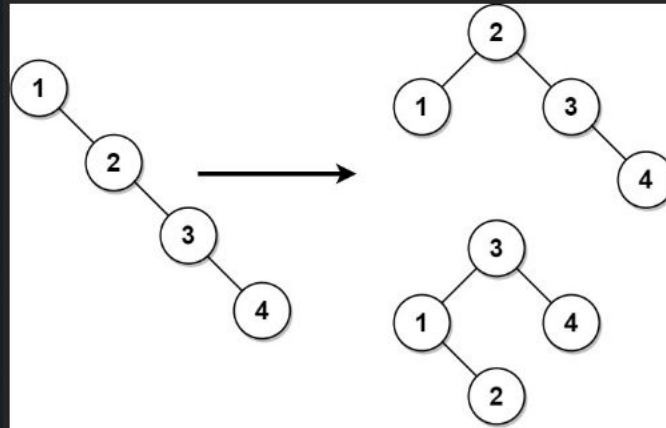
Companies

Hint

Given the `root` of a binary search tree, return a **balanced** binary search tree with the same node values. If there is more than one answer, return **any of them**.

A binary search tree is **balanced** if the depth of the two subtrees of every node never differs by more than 1.

Example 1:



Input: `root = [1,null,2,null,3,null,4,null,null]`

Output: `[2,1,3,null,null,null,4]`

Explanation: This is not the only correct answer, `[3,1,4,null,2]` is also correct.

Binary Search Tree - Balanceamento

<https://leetcode.com/problems/balance-a-binary-search-tree/description/>

```
def balanceBST(self, root: TreeNode) → TreeNode:
    arr = []
    self.inOrder(root, arr)
    return self.balance(arr, 0, len(arr) - 1)

def inOrder(self, root, arr):
    if not root: return

    self.inOrder(root.left, arr)
    arr.append(root.val)
    self.inOrder(root.right, arr)

def balance(self, arr, left, right):
    if left > right: return None

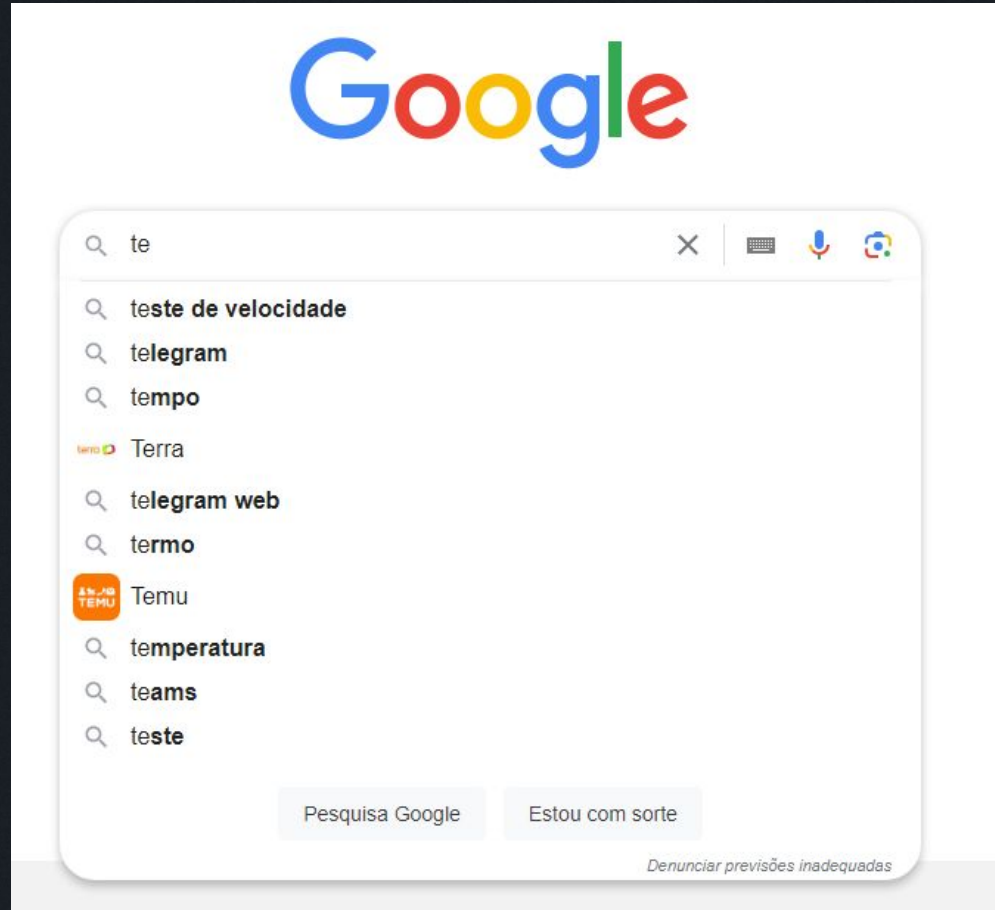
    mid = (left + right) // 2
    root = TreeNode(arr[mid])
    root.left = self.balance(arr, left, mid - 1)
    root.right = self.balance(arr, mid + 1, right)
    return root
```




Tries

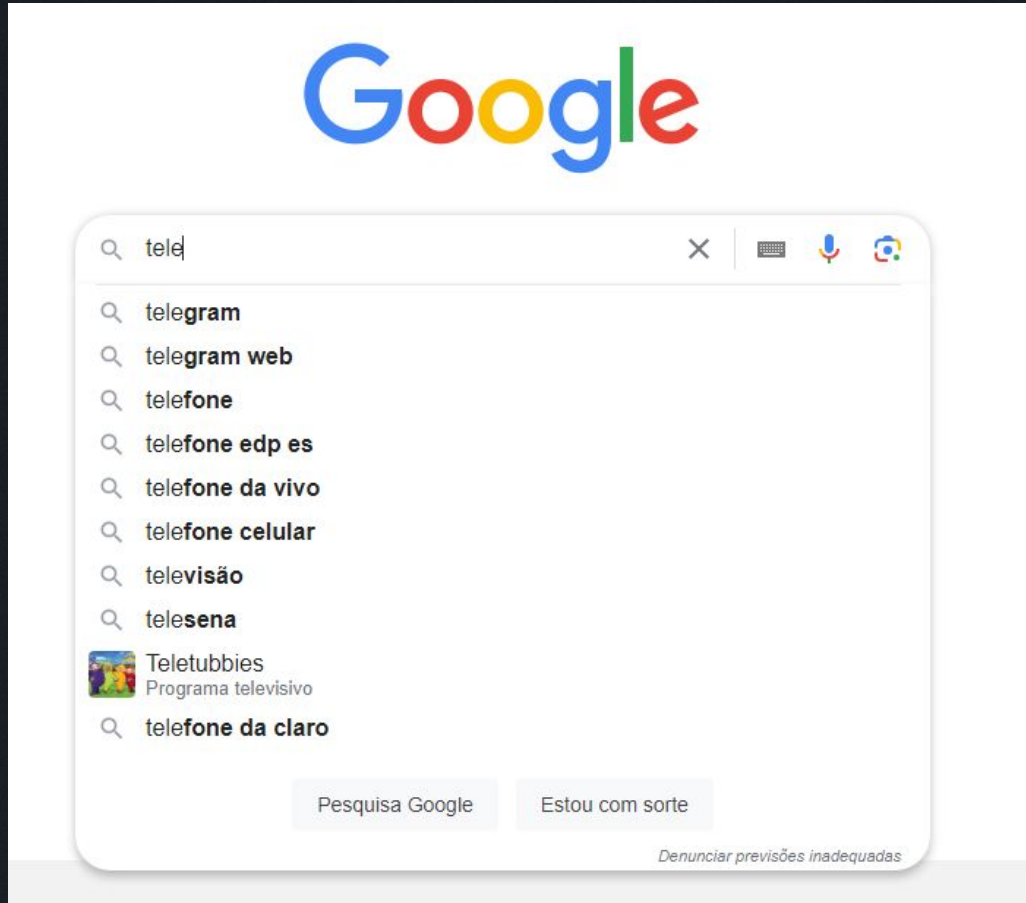
AutoComplete de uma máquina de busca

Como o Google faz isso?



AutoComplete de uma máquina de busca

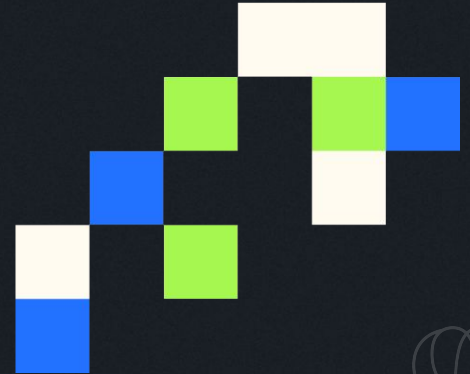
Como o Google faz isso?



01. Problema

Implementando o AutoComplete

Sugestões de palavras à medida que você começa a digitá-las.



AutoComplete de uma máquina de busca

Imagine que você começou a trabalhar em uma empresa que possui uma máquina de busca online que permite você encontrar qualquer página na web. Essa máquina de busca aceita palavras (queries). Após digitar uma letra, a máquina de busca automaticamente sugere palavras que começam com a letra digitada. Ao digitar a segunda letra, palavras que começam com prefixo digitado são sugeridas e assim por diante. Se você digitar uma palavra que não existe, a lista de sugestões é vazia. Além disso, se o usuário selecionar uma das sugestões, a palavra (query) é enviada ao buscador para obtenção das páginas mais relevantes de acordo com a seleção do usuário.

Para a aula de hoje vamos focar na funcionalidade de sugestões de palavras à medida que você começar a digitá-las (AutoComplete).

Implemente uma solução que contenha 3 métodos:

1) Inserir palavra

Método para adicionar palavras à máquina de busca.

2) Buscar palavra

Método que verifica se uma palavra já foi adicionada na máquina de busca.

3) Buscar prefixo

Método que retorna um conjunto de palavras que começam com o prefixo informado.

Interface do AutoComplete

Para implementar o AutoComplete, precisaremos de três métodos principais:

```
package Ada;

import java.util.*;

public interface IAutoComplete {
    void insert(String word);
    boolean search(String word);
    Set<String> startsWith(String word);
}
```

Vamos implementar?

```
package Ada.ToDo;

import Ada.IAutoComplete;

import java.util.*;

public class AutoCompleteSolution1 implements IAutoComplete {

    @Override
    public void insert(String word) {

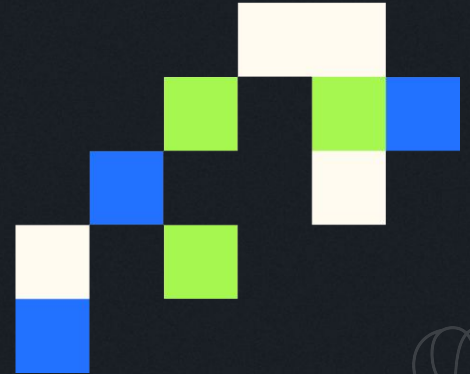
    }

    @Override
    public boolean search(String word) {
        return false;
    }

    @Override
    public Set<String> startsWith(String word) {
        return null;
    }
}
```

02. Soluções iniciais

Discussão de estratégias para resolução



Estratégia 1

- 1 - Inicializar uma lista vazia para armazenar as palavras.
- 2 - Ao executar o método inserir palavra, podemos adicionar a palavra na lista.
- 3 - Ao executar o método buscar palavra, podemos percorrer a lista de palavras verificando se a palavra informada existe na lista.
- 4 - Ao executar o método buscar prefixo, podemos iniciar um conjunto para receber as possíveis sugestões e então percorrer a lista de palavras verificando se existe alguma palavra com o prefixo informado, caso positivo adicionaremos a palavra ao conjunto de sugestões.

Primeira implementação

```
package Ada;

import java.util.*;

public class AutoCompleteSolution1 implements IAutoComplete {
    List<String> cacheList;

    public AutoCompleteSolution1() {
        cacheList = new ArrayList<>();
    }

    @Override
    public void insert(String word) {
        cacheList.add(word);
    }

    @Override
    public boolean search(String word) {
        for (int index = 0; index < cacheList.size(); index++) {
            if (cacheList.get(index).equals(word)) {
                return true;
            }
        }
        return false;
    }
}
```

```
    @Override
    public Set<String> startsWith(String word) {
        Set<String> result = new HashSet<>();

        for (int index = 0; index < cacheList.size(); index++) {
            if (cacheList.get(index).startsWith(word)) {
                result.add(cacheList.get(index));
            }
        }

        return result;
    }
}
```

Primeira implementação

```
package Ada;

import java.util.*;

public class AutoCompleteSolution1 implements IAutoComplete {
    List<String> cacheList;

    public AutoCompleteSolution1() {
        cacheList = new ArrayList<>();
    }

    @Override
    public void insert(String word) {
        cacheList.add(word);
    }

    @Override
    public boolean search(String word) {
        for (int index = 0; index < cacheList.size(); index++) {
            if (cacheList.get(index).equals(word)) {
                return true;
            }
        }
        return false;
    }
}
```

```
@Override
public Set<String> startsWith(String word) {
    Set<String> result = new HashSet<>();

    for (int index = 0; index < cacheList.size(); index++) {
        if (cacheList.get(index).startsWith(word)) {
            result.add(cacheList.get(index));
        }
    }

    return result;
}
```

Complexidade de tempo:

Inserção - ?

Busca - ?

Busca do prefixo - ?

Complexidade de espaço:

?

Primeira implementação

```
package Ada;

import java.util.*;

public class AutoCompleteSolution1 implements IAutoComplete {
    List<String> cacheList;

    public AutoCompleteSolution1() {
        cacheList = new ArrayList<>();
    }

    @Override
    public void insert(String word) {
        cacheList.add(word);
    }

    @Override
    public boolean search(String word) {
        for (int index = 0; index < cacheList.size(); index++) {
            if (cacheList.get(index).equals(word)) {
                return true;
            }
        }
        return false;
    }
}
```

```
@Override
public Set<String> startsWith(String word) {
    Set<String> result = new HashSet<>();

    for (int index = 0; index < cacheList.size(); index++) {
        if (cacheList.get(index).startsWith(word)) {
            result.add(cacheList.get(index));
        }
    }

    return result;
}
```

Complexidade de tempo:

Inserção - $O(1)$

Busca - $O(N*L) \rightarrow O(N)$

Busca do prefixo - $O(N*P) \rightarrow O(N)$

Complexidade de espaço:

$O(N*L) \rightarrow O(N)$

Estratégia 2 – Redução da complexidade de tempo do método Buscar palavra

Complexidade de tempo:

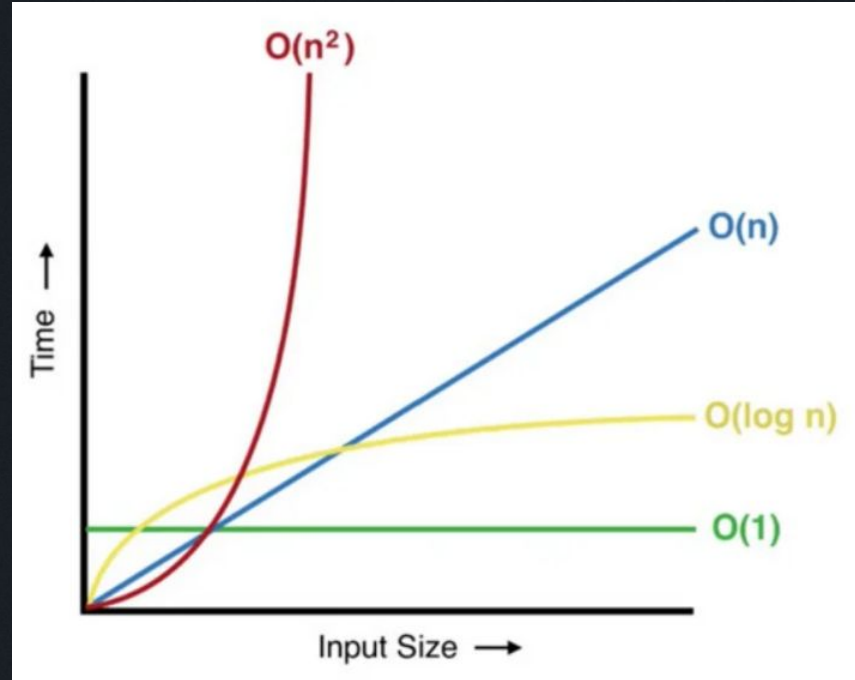
Inserção - $O(1)$

Busca - $O(N*L) \rightarrow O(N)$

Busca do prefixo - $O(N*P) \rightarrow O(N)$

Complexidade de espaço:

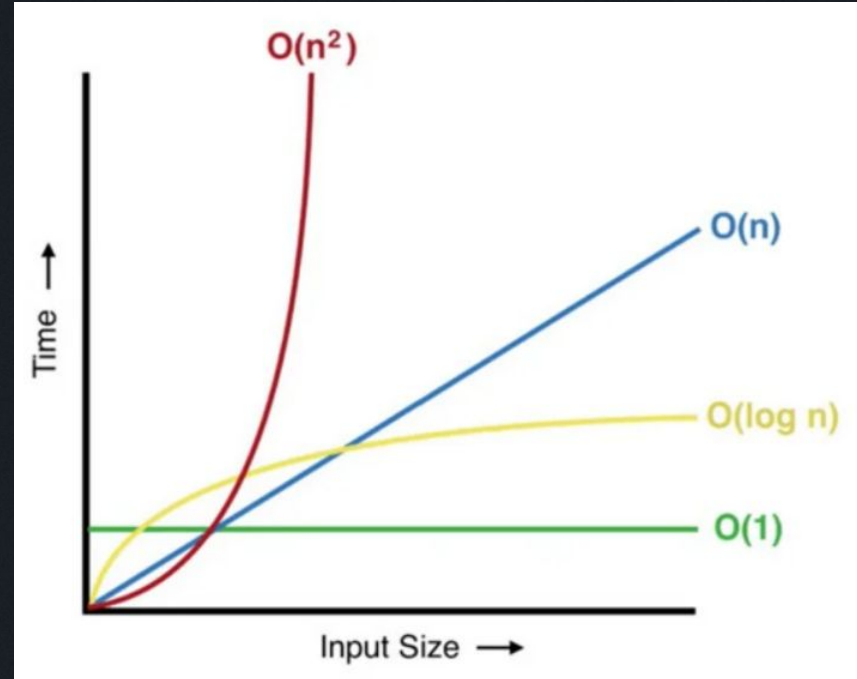
$O(N*L) \rightarrow O(N)$



Estratégia 2 – Redução da complexidade de tempo do método Buscar palavra

HashTable (HashSet)

- Inserção - $O(1)$
- Busca - $O(1)$



Segunda implementação

```
package Ada;

import java.util.*;

public class AutoCompleteSolution2 implements IAutoComplete {
    Set<String> cacheSet;

    public AutoCompleteSolution2() {
        cacheSet = new HashSet<>();
    }

    @Override
    public void insert(String word) {
        cacheSet.add(word);
    }

    @Override
    public boolean search(String word) {
        return cacheSet.contains(word);
    }
}
```

```
@Override
public Set<String> startsWith(String prefix) {
    Set<String> result = new HashSet<>();

    for(String item : cacheSet) {
        if (item.startsWith(prefix)) {
            result.add(item);
        }
    }

    return result;
}
```

Complexidade de tempo:

Inserção - ?

Busca - ?

Busca do prefixo - ?

Complexidade de espaço:

?

Segunda implementação

```
package Ada;

import java.util.*;

public class AutoCompleteSolution2 implements IAutoComplete {
    Set<String> cacheSet;

    public AutoCompleteSolution2() {
        cacheSet = new HashSet<>();
    }

    @Override
    public void insert(String word) {
        cacheSet.add(word);
    }

    @Override
    public boolean search(String word) {
        return cacheSet.contains(word);
    }
}
```

```
@Override
public Set<String> startsWith(String prefix) {
    Set<String> result = new HashSet<>();

    for(String item : cacheSet) {
        if (item.startsWith(prefix)) {
            result.add(item);
        }
    }

    return result;
}
```

Complexidade de tempo:

Inserção – $O(1)$

Busca – $O(1)$

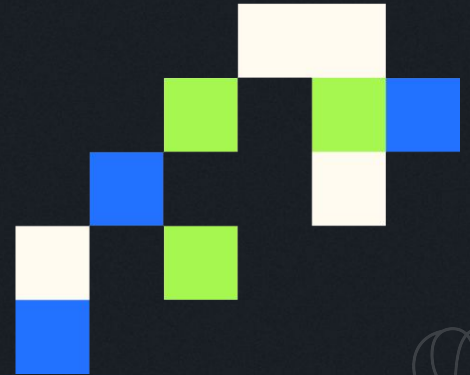
Busca do prefixo – $O(N \cdot P) \rightarrow O(N)$

Complexidade de espaço:

$O(N \cdot L) \rightarrow O(N)$

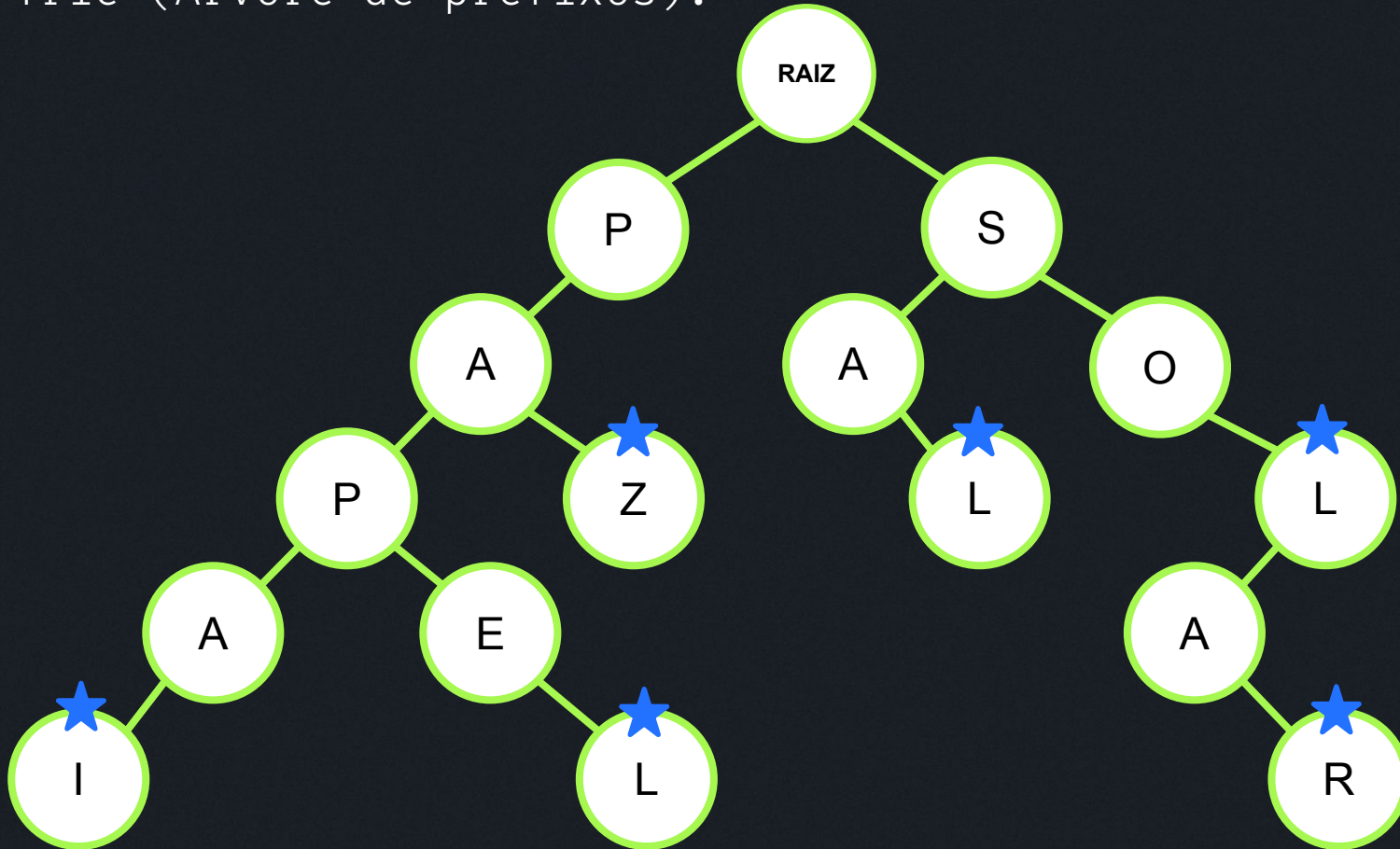
03. Otimização

Apresentação da Trie.



Qual seria uma implementação mais eficiente?

Vamos conhecer um pouco mais sobre a Trie (Árvore de prefixos).



Trie

- Retrieval tree, árvore de prefixos, árvore digital.
- A Trie é uma árvore N-ária onde cada nó representa um caractere de uma string.
- O número de filhos de cada nó é o tamanho do alfabeto (26 letras) – Ordem da árvore.
- Cada caminho da raiz até uma folha forma uma string completa.
- Aplicações comuns: dicionários, sistemas de autocomplete, corretores ortográficos, compressão de texto e busca de padrões.

Inserção de palavras

PAPAI

SOL

PAPEL

SAL

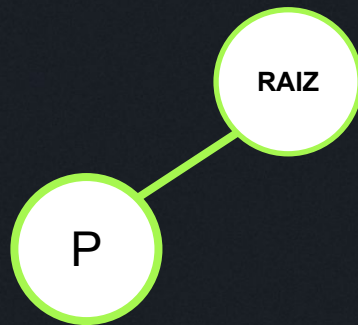
PAZ

SOLAR



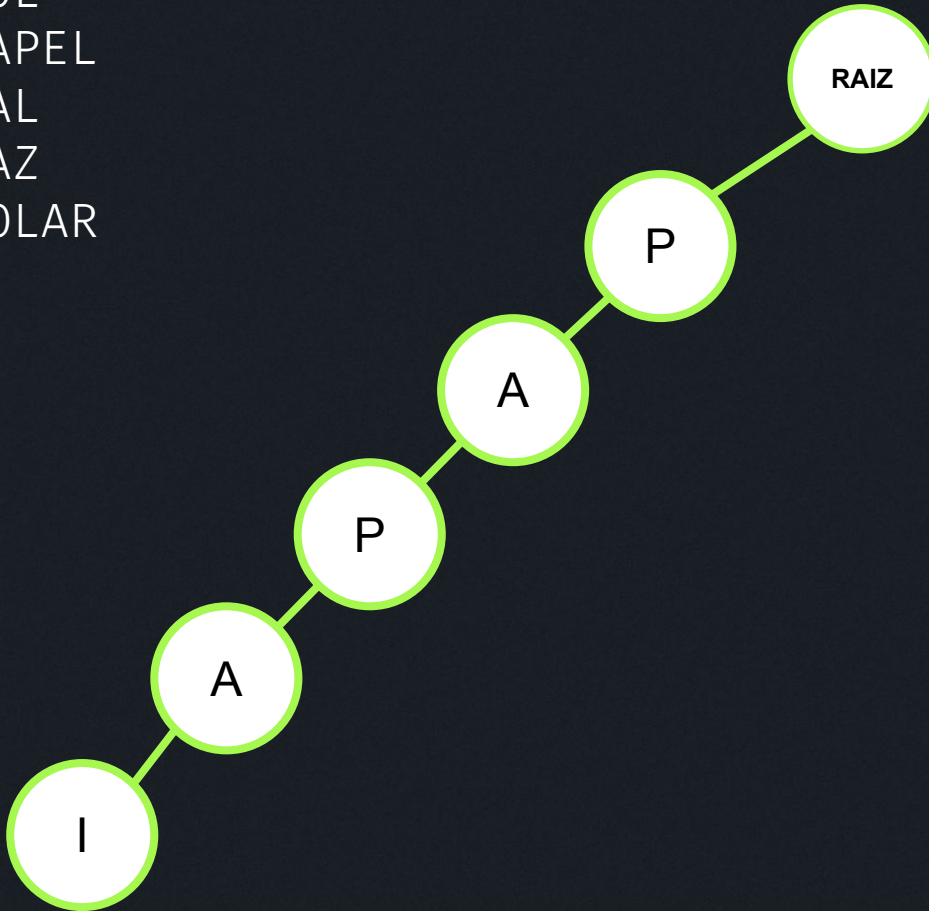
Inserção de palavras

- PAPAI
- SOL
- PAPEL
- SAL
- PAZ
- SOLAR



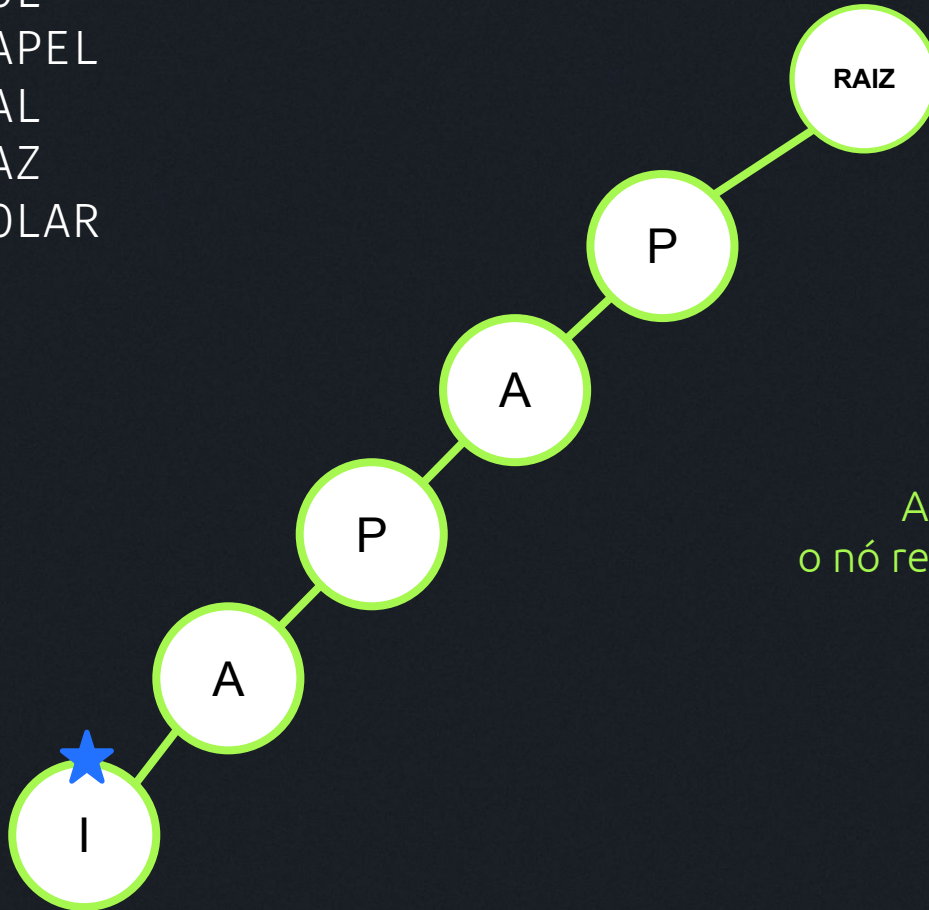
Inserção de palavras

● PAPAI
SOL
PAPEL
SAL
PAZ
SOLAR



Inserção de palavras

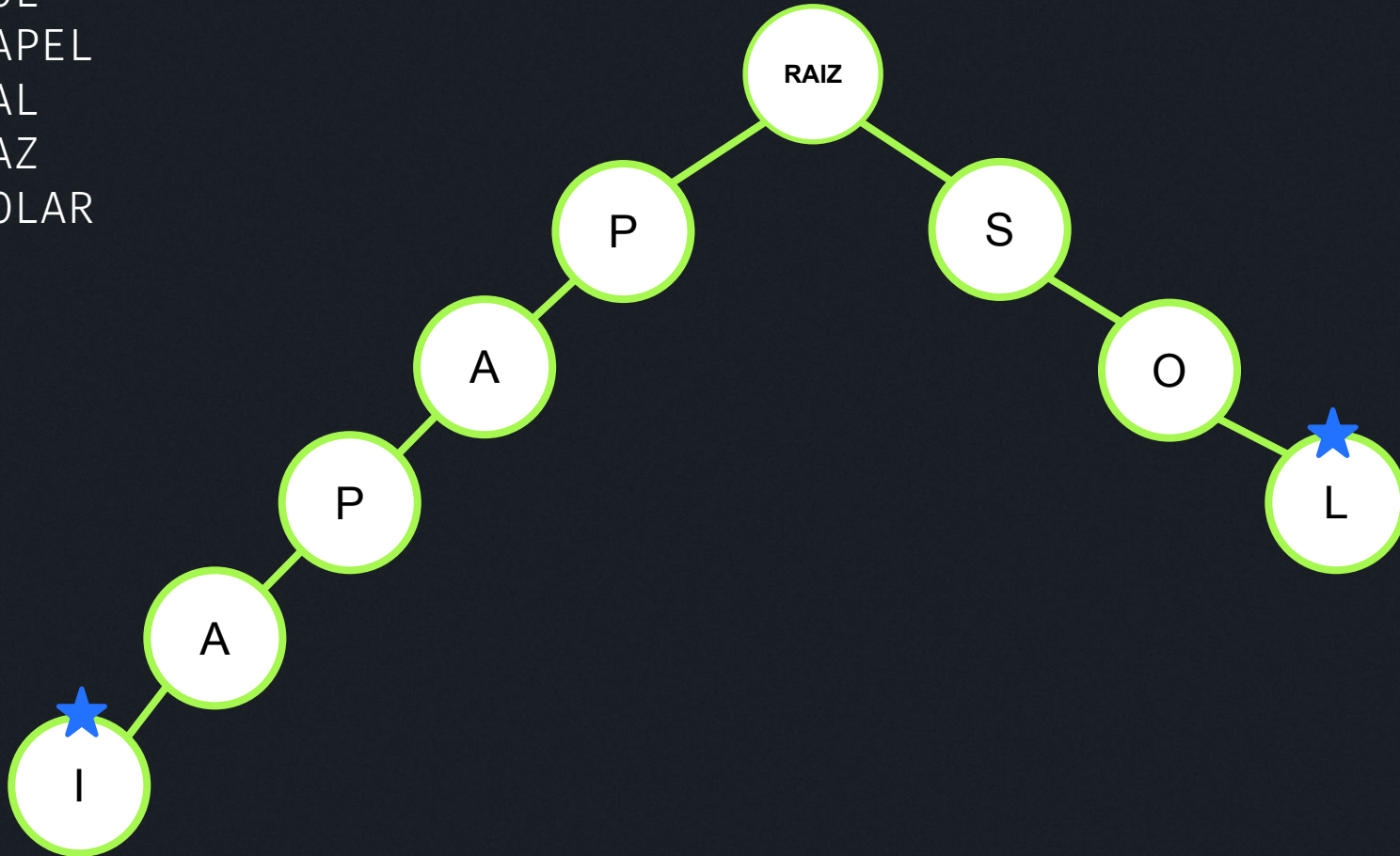
- PAPAI
- SOL
- PAPEL
- SAL
- PAZ
- SOLAR



A estrela é uma flag que indica que o nó representa a última letra de uma palavra.

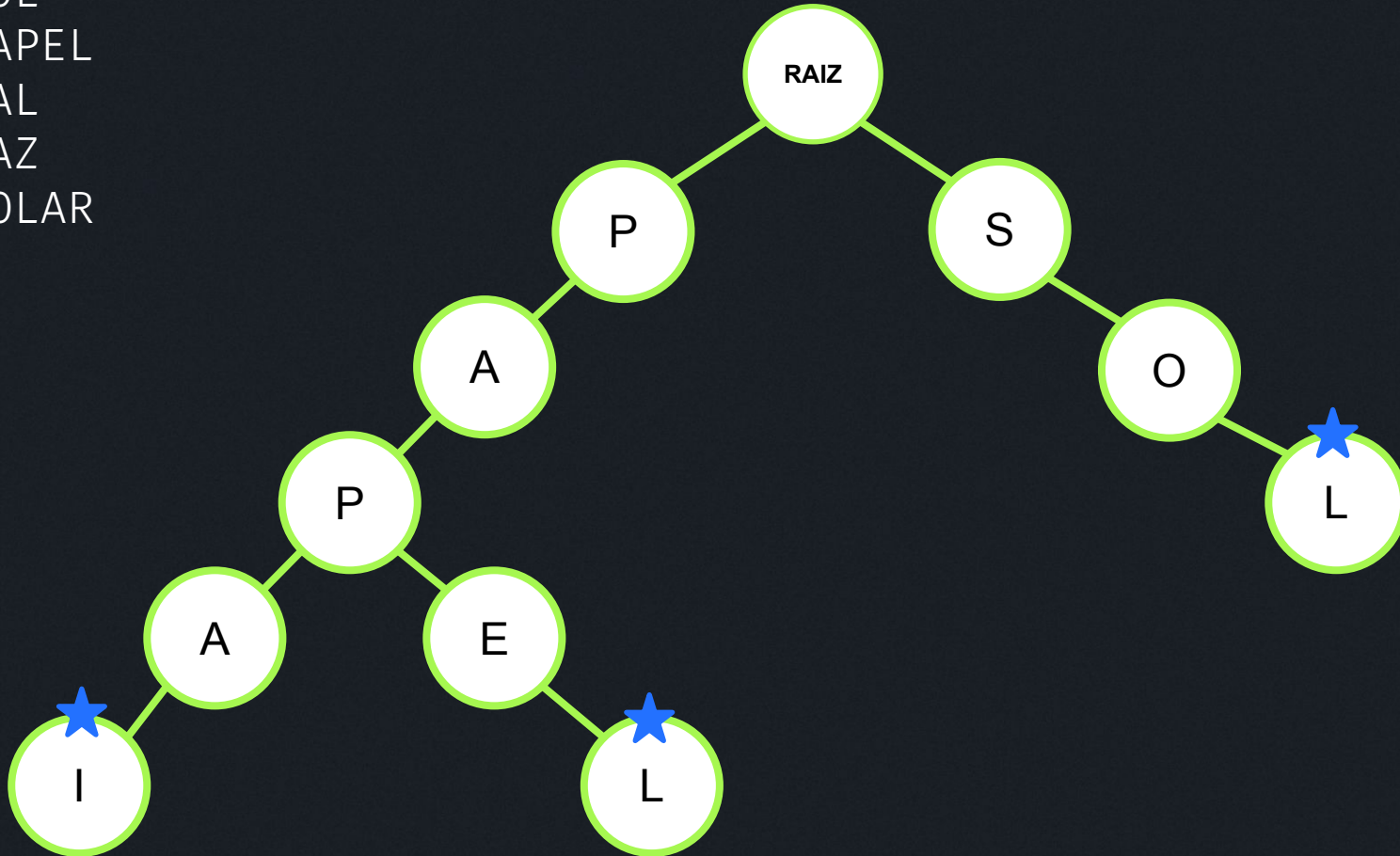
Inserção de palavras

● PAPAI
● SOL
PAPEL
SAL
PAZ
SOLAR



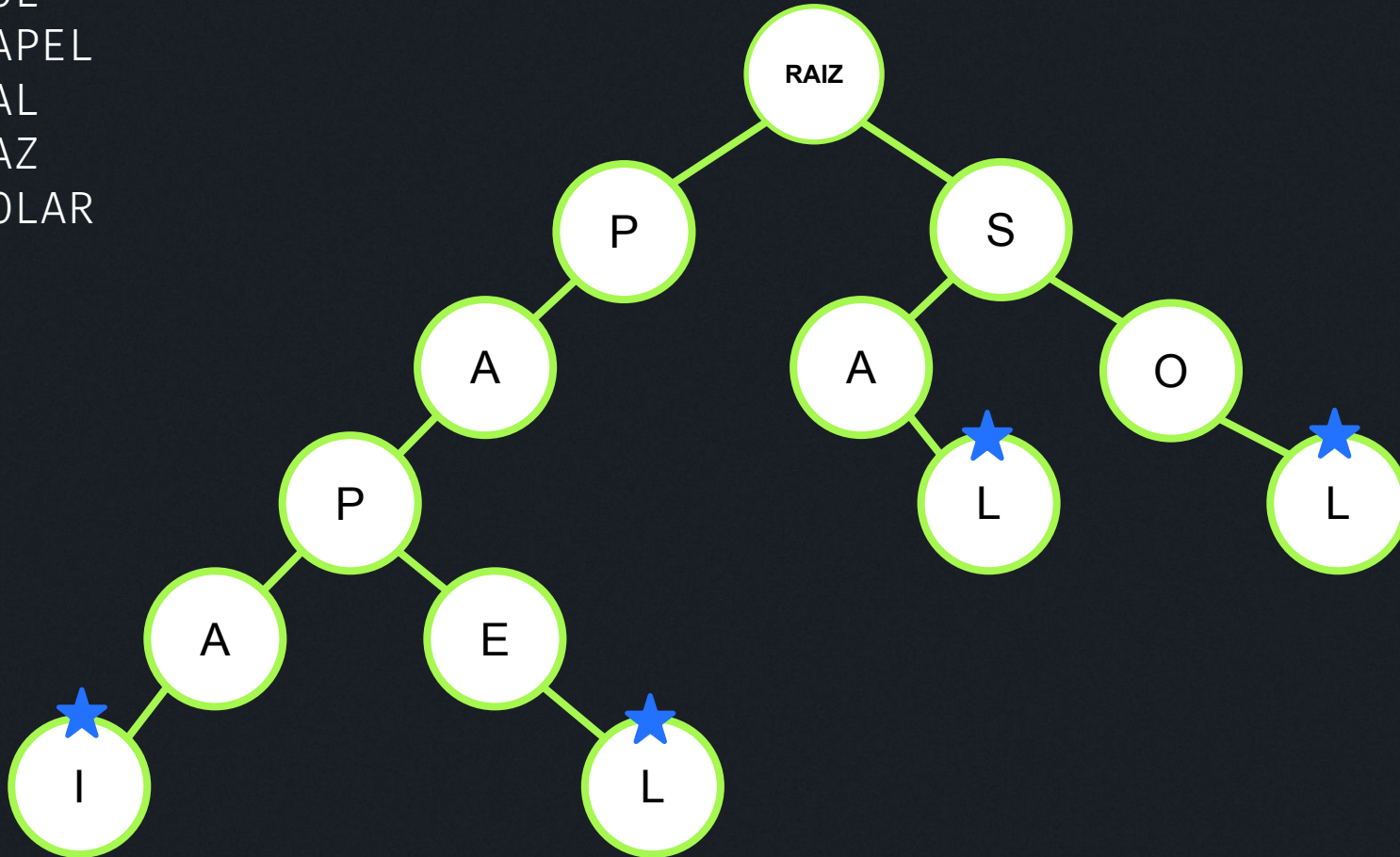
Inserção de palavras

- PAPAI
- SOL
- PAPEL
- SAL
- PAZ
- SOLAR



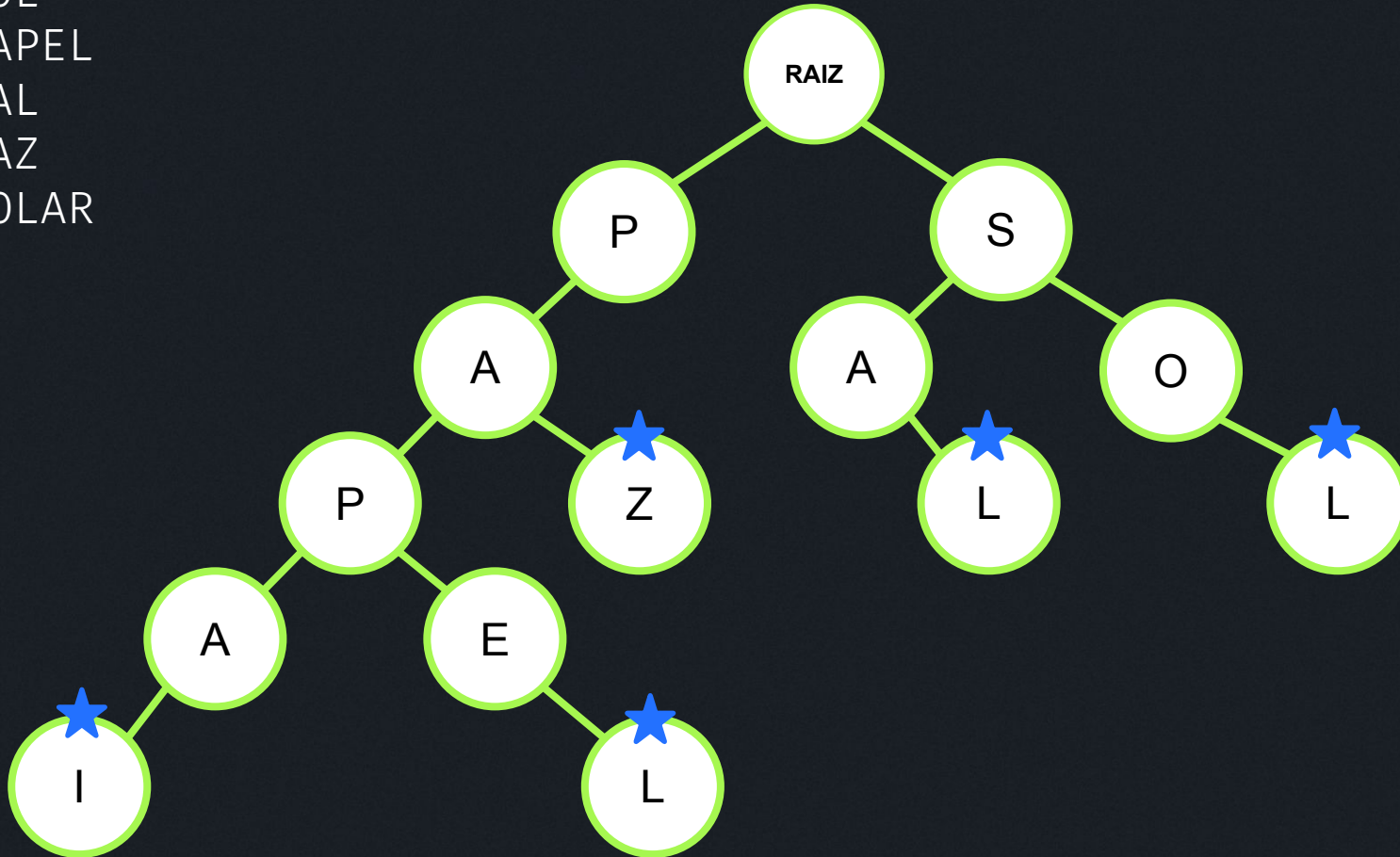
Inserção de palavras

- PAPAI
- SOL
- PAPEL
- SAL
- PAZ
- SOLAR



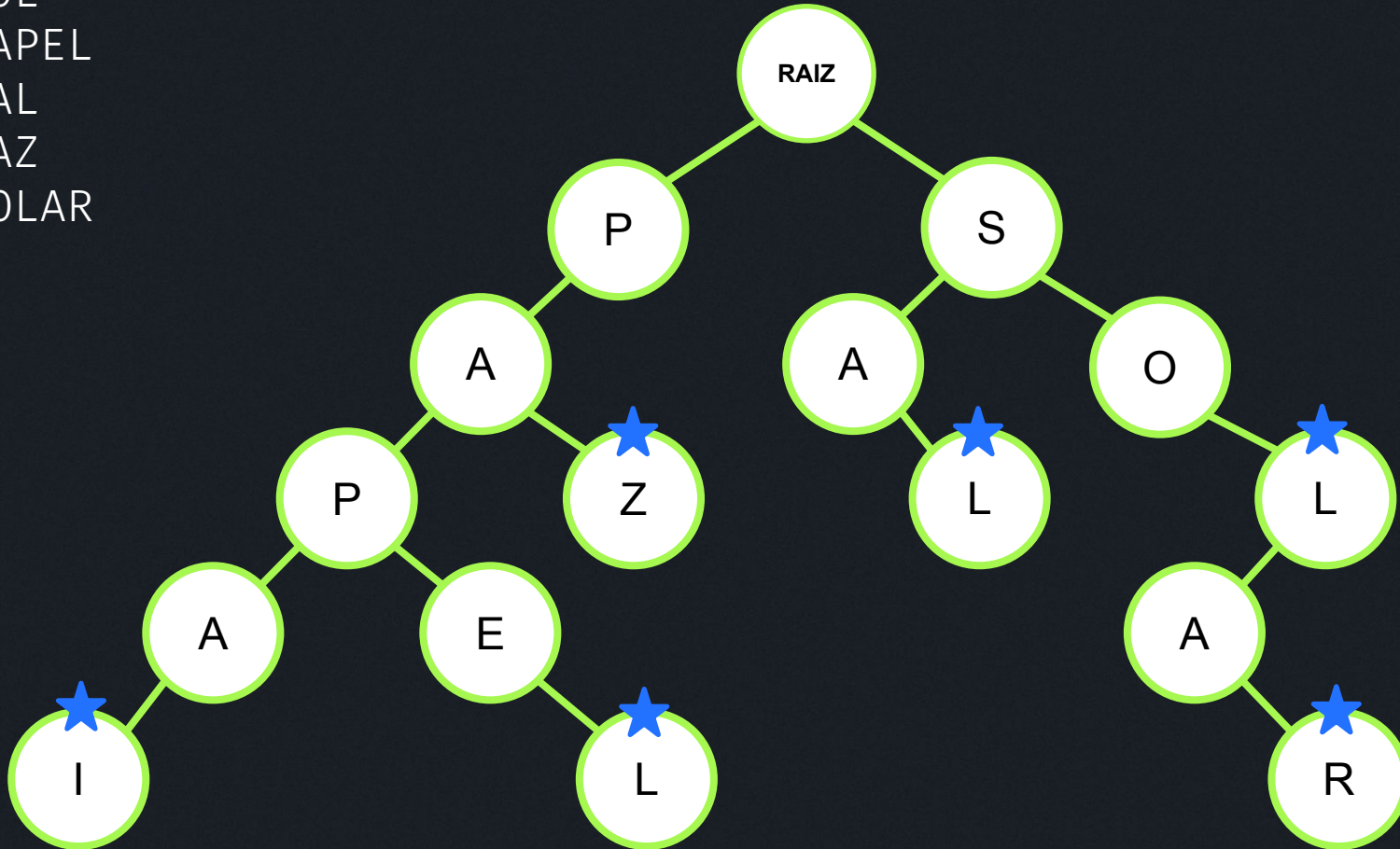
Inserção de palavras

- PAPAI
- SOL
- PAPEL
- SAL
- PAZ
- SOLAR



Inserção de palavras

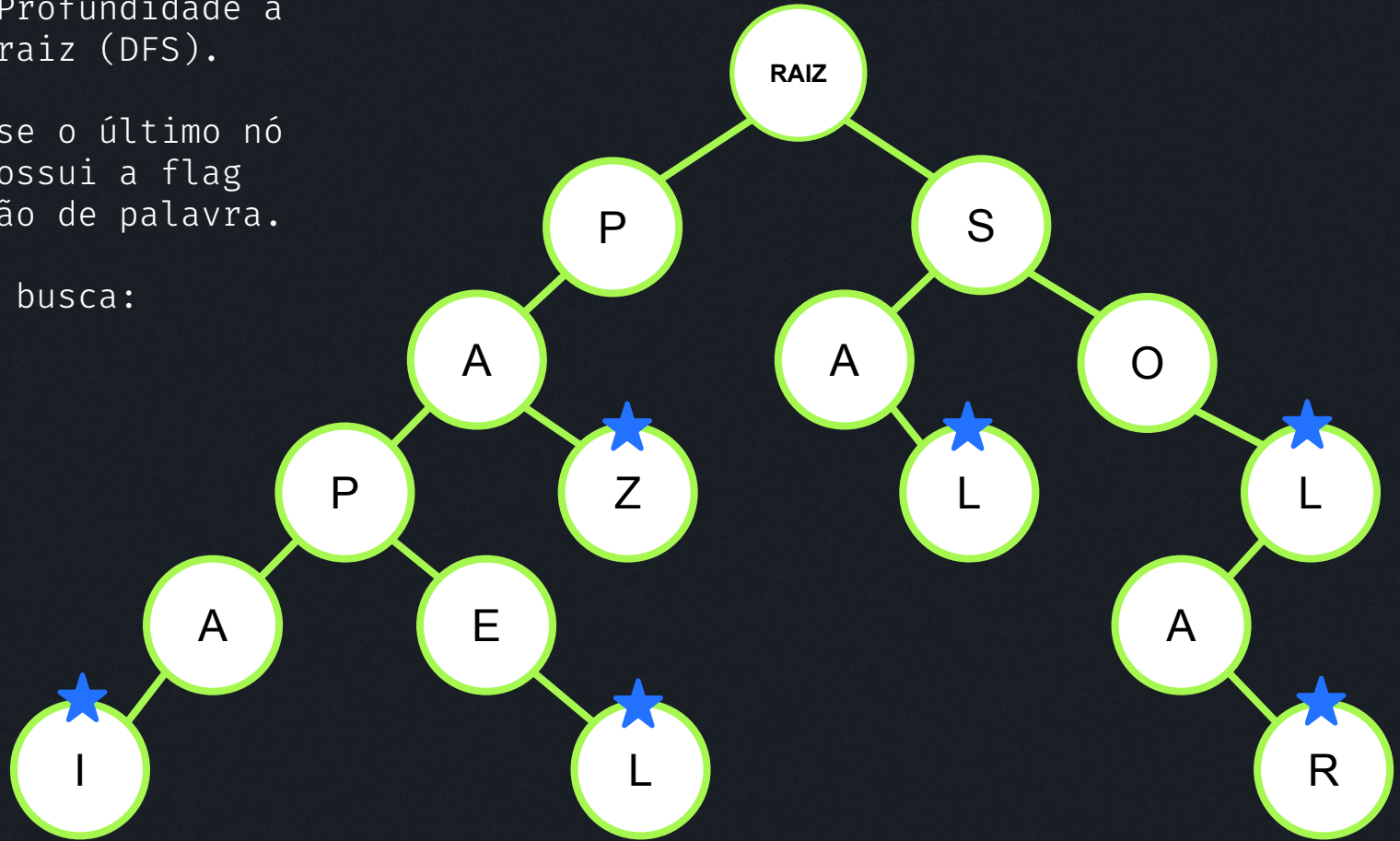
- PAPAI
- SOL
- PAPEL
- SAL
- PAZ
- SOLAR



Como verificar se a Trie contém uma palavra?

- Buscar em Profundidade a partir da raiz (DFS).
- Verificar se o último nó visitado possui a flag de indicação de palavra.
- Exemplo de busca:

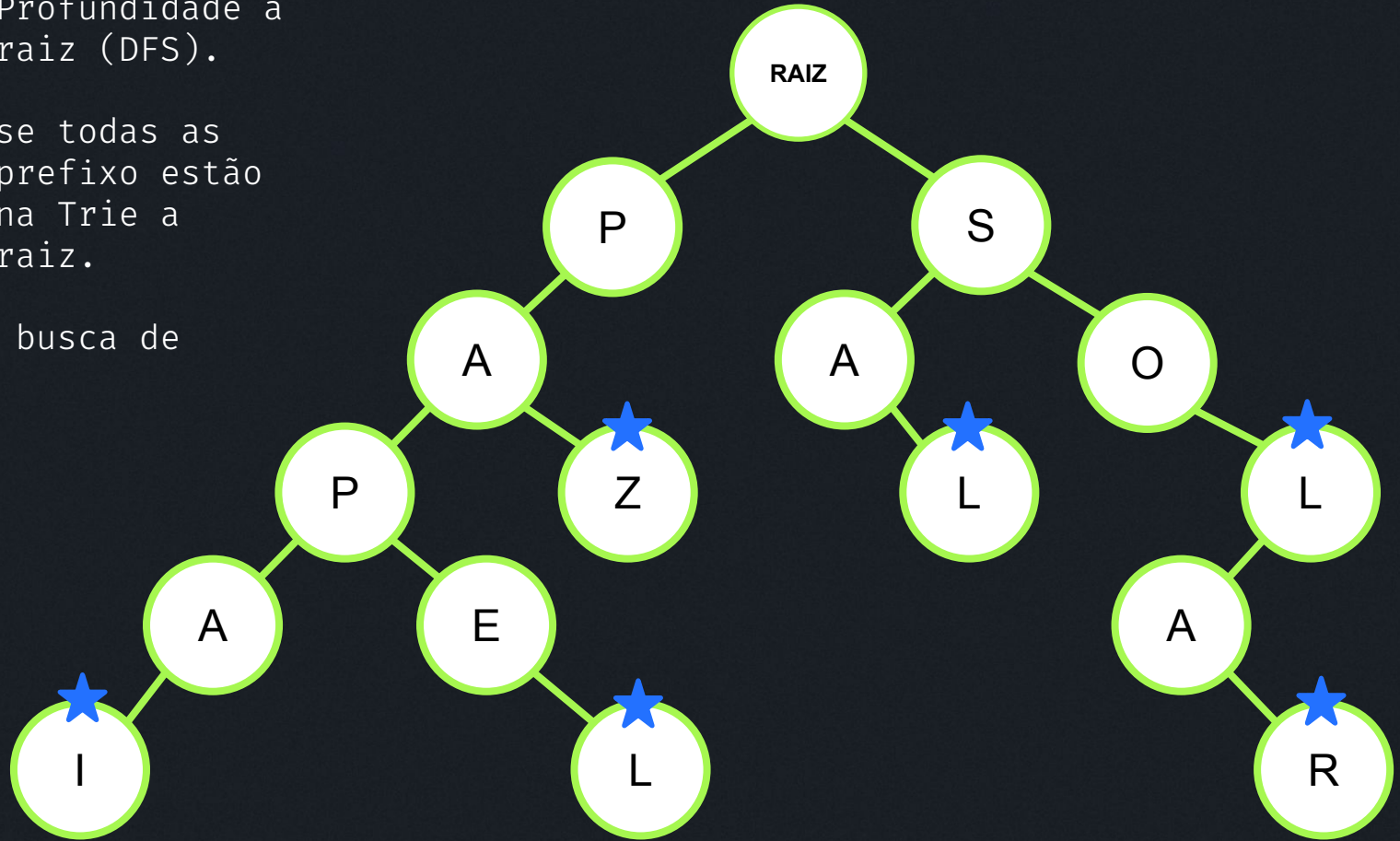
PAZ
PAPELARIA
SOLA



Como verificar se a Trie contém um prefixo?

- Buscar em Profundidade a partir da raiz (DFS).
- Verificar se todas as letras do prefixo estão presentes na Trie a partir da raiz.
- Exemplo de busca de prefixo:

PAP
PAZ
SOLA



Interface da Trie

Métodos básicos de uma Trie

```
package Ada;  
  
public interface Trie {  
    void insert(String word);  
    boolean search(String word);  
    boolean startsWith(String prefix);  
}
```

Como implementar uma Trie?

```
package Ada;

class EmptyTrie implements Trie {
    @Override
    public void insert(String word) {

    }

    @Override
    public boolean search(String word) {
        return false;
    }

    @Override
    public boolean startsWith(String prefix) {
        return false;
    }
}
```

Implementação da Trie

```
package Ada;

class Trie {
    Trie[] letters;
    boolean isWord;

    public Trie() {
        letters = new Trie[26];
    }

    public void insert(String word) {
        Trie currentNode = this;

        for(char letter : word.toCharArray()) {
            if(currentNode.letters[letter - 'a'] == null) {
                currentNode.letters[letter - 'a'] = new Trie();
            }
            currentNode = currentNode.letters[letter - 'a'];
        }

        currentNode.isWord = true;
    }
}
```

```
public boolean search(String word) {
    Trie currentNode = this;

    for(char letter : word.toCharArray()) {
        if(currentNode.letters[letter - 'a'] == null) {
            return false;
        }
        currentNode = currentNode.letters[letter - 'a'];
    }

    return currentNode.isWord;
}

public boolean startsWith(String prefix) {
    Trie currentNode = this;

    for(char letter : prefix.toCharArray()) {
        if(currentNode.letters[letter - 'a'] == null) {
            return false;
        }
        currentNode = currentNode.letters[letter - 'a'];
    }

    return true;
}
```


Sugestão de palavras

Como usar a Trie para sugerir palavras?

Implementação da SuggestionsTrie.

Uso da Trie para sugerir palavras

```
package Ada;

import java.util.*;

class SuggestionsTrie {
    SuggestionsTrie[] letters;
    Set<String> words;

    public SuggestionsTrie() {
        letters = new SuggestionsTrie[26];
        words = new HashSet<>();
    }

    public void insert(String word) {
        SuggestionsTrie currentNode = this;

        for(char letter : word.toCharArray()) {
            if(currentNode.letters[letter - 'a'] == null) {
                currentNode.letters[letter - 'a'] = new SuggestionsTrie();
            }
            currentNode = currentNode.letters[letter - 'a'];
            currentNode.words.add(word);
        }
    }
}
```

```
public boolean search(String word) {
    SuggestionsTrie currentNode = this;

    for(char letter : word.toCharArray()) {
        if(currentNode.letters[letter - 'a'] == null) {
            return false;
        }
        currentNode = currentNode.letters[letter - 'a'];
    }

    return currentNode.words.contains(word);
}

public Set<String> startsWith(String prefix) {
    SuggestionsTrie currentNode = this;

    for(char letter : prefix.toCharArray()) {
        if(currentNode.letters[letter - 'a'] == null) {
            return new HashSet<>();
        }
        currentNode = currentNode.letters[letter - 'a'];
    }

    return currentNode.words;
}
```

AutoComplete usando Trie

Vamos atualizar a classe AutoComplete para usar a Trie.

AutoComplete usando Trie

```
package Ada;

import java.util.*;

public class AutoCompleteWithTrie implements IAutoComplete {
    SuggestionsTrie trie;

    public AutoCompleteWithTrie() {
        trie = new SuggestionsTrie();
    }

    @Override
    public void insert(String word) {
        trie.insert(word);
    }

    @Override
    public boolean search(String word) {
        return trie.search(word);
    }
}
```

```
    @Override
    public Set<String> startsWith(String prefix) {
        Set<String> words = trie.startsWith(prefix);
        return words;
    }
}
```


AutoComplete usando Trie

```
package Ada;

import java.util.*;

public class AutoCompleteWithTrie implements IAutoComplete {
    SuggestionsTrie trie;

    public AutoCompleteWithTrie() {
        trie = new SuggestionsTrie();
    }

    @Override
    public void insert(String word) {
        trie.insert(word);
    }

    @Override
    public boolean search(String word) {
        return trie.search(word);
    }
}
```

```
    @Override
    public Set<String> startsWith(String prefix) {
        Set<String> words = trie.startsWith(prefix);
        return words;
    }
}
```

Complexidade de tempo:

Inserção - ?

Busca - ?

Busca do prefixo - ?

Complexidade de espaço:

?

AutoComplete usando Trie

```
package Ada;

import java.util.*;

public class AutoCompleteWithTrie implements IAutoComplete {
    SuggestionsTrie trie;

    public AutoCompleteWithTrie() {
        trie = new SuggestionsTrie();
    }

    @Override
    public void insert(String word) {
        trie.insert(word);
    }

    @Override
    public boolean search(String word) {
        return trie.search(word);
    }
}
```

```
    @Override
    public Set<String> startsWith(String prefix) {
        Set<String> words = trie.startsWith(prefix);
        return words;
    }
}
```

Complexidade de tempo:

Inserção - $O(L) \rightarrow O(1)$

Busca - $O(L) \rightarrow O(1)$

Busca do prefixo - $O(P) \rightarrow O(1)$

where L is the word length
and P is the prefix length.

Complexidade de espaço:

$O(N * L^2) \rightarrow O(N)$

1) Implementar uma Trie

208. Implement Trie (Prefix Tree)

Medium

Topics

Companies

A **trie** (pronounced as "try") or **prefix tree** is a tree data structure used to efficiently store and retrieve keys in a dataset of strings. There are various applications of this data structure, such as autocomplete and spellchecker.

Implement the Trie class:

- `Trie()` Initializes the trie object.
- `void insert(String word)` Inserts the string `word` into the trie.
- `boolean search(String word)` Returns `true` if the string `word` is in the trie (i.e., was inserted before), and `false` otherwise.
- `boolean startsWith(String prefix)` Returns `true` if there is a previously inserted string `word` that has the prefix `prefix`, and `false` otherwise.

Example 1:

Input

```
["Trie", "insert", "search", "search", "startsWith", "insert", "search"]  
[[], ["apple"], ["apple"], ["app"], ["app"], ["app"], ["app"]]
```

Output

```
[null, null, true, false, true, null, true]
```

Explanation

```
Trie trie = new Trie();  
trie.insert("apple");  
trie.search("apple"); // return True  
trie.search("app");   // return False  
trie.startsWith("app"); // return True  
trie.insert("app");  
trie.search("app");   // return True
```

<https://leetcode.com/problems/implement-trie-prefix-tree/description/>



2) Busca de palavras usando ponto (.) (. pode ser qualquer letra)

Design a data structure that supports adding new words and finding if a string matches any previously added string.

Implement the `WordDictionary` class:

- `WordDictionary()` Initializes the object.
- `void addWord(word)` Adds `word` to the data structure, it can be matched later.
- `bool search(word)` Returns `true` if there is any string in the data structure that matches `word` or `false` otherwise. `word` may contain dots `'.'` where dots can be matched with any letter.

Example:

Input

```
["WordDictionary","addWord","addWord","addWord","search","search","search","search"]  
[[],["bad"],["dad"],["mad"],["pad"],["bad"],[".ad"],["b.."]]
```

Output

```
[null,null,null,null,false,true,true,true]
```

Explanation

```
WordDictionary wordDictionary = new WordDictionary();  
wordDictionary.addWord("bad");  
wordDictionary.addWord("dad");  
wordDictionary.addWord("mad");  
wordDictionary.search("pad"); // return False  
wordDictionary.search("bad"); // return True  
wordDictionary.search(".ad"); // return True  
wordDictionary.search("b.."); // return True
```


Obrigado