Lecture Notes for LeetCode

Emulie Chhor

May 9, 2021

Introduction

Le but de ce document est de compiler les stratégie utilisées pour résoudre les questions de LeetCode. Les stratégies sont séparées de la façon suivante:

1. Arrays
2. Linked List
3. String
4. Hash Table
5. Sorting
6. Recursion
7. Trees
8. Dynamic Programming
9. Graph Problems

Il est à noter que les problèmes les plus durs peuvent toujours être séparés en problème plus simple. De plus, certains problèmes peuvent être considérés comme des problèmes récurrents, c-à-d qu'ils sont souvent utilisées non pas comme une fin en soit, mais comme un stratégie intermédiaire pour résoudre un plus gros problème.

1 Arrays

Overview

1.1 Two Pointers

1.2 Sliding Window Technique

1.2.1 In-place Sliding Window

1.2.2 Dynamic Sliding Window

1.3 Kadane's Algorithm

2 Linked List

2.1 Fast-and-Slow Pointers

3 Trees

3.1 Tree Traversal

Motivation

Malgré qu'on nous demande pas explicitement de traverser un arbre, certains problèmes utilisent la structure des traversals. Il existe 4 façons de traverser les arbres:

1. preorder (depth)
2. in-order(depth)
3. post-order(depth)
4. level-order (breadth)

Les traversals depth-first peuvent être implémentées récursivement ou itérativement avec un stack/queue.

3.1.1 Pre-order

Implémentation Récursive

1. visit node
2. traverse left
3. traverse right

Implémentation Itérative

3.1.2 In-order

Implémentation Récursive

1. traverse left
2. visit node
3. traverse right

Implémentation Itérative

3.1.3 Post-Order

Implémentation Récursive

1. traverse left
2. traverse right
3. visit node

Implémentation Itérative

3.1.4 Level-Order

Implémentation Itérative

3.2 Find Smallest/Biggest Element

Motivation

Un sous-problème qui revient souvent est de trouver le plus gros/plus petit élément dans un subtree. Malgré que l'algorithme est simple, il nous permet de trouver le successeur/prédécesseur lors de la suppression dans un arbre.

Intuition

Puisque le BST possède le BST invariant, qui nous dit que $node.left \leq node \leq node.right$, on trouve le plus petit enfant en traversant la gauche jusqu'à ce qu'on trouve un noeud null. De la même façon, pour trouver le noeud le plus grand, on doit traverser le BST en choisissant toujours le noeud de droite, jusqu'à avoir un null.

3.3 Find Depth/Height of Tree

4 Graph Problems

4.1 Shortest Path

4.2 Topological Sort

4.3 Conectivity

4.4 Minimum Spanning Tree

4.5 Cycle Detection

Ressources

1. Currated Top 75 LeetCode Questions: https://www.teamblind.com/post/New-Year-Gift---Curated-List-of-Top-100-LeetCode-Questions-to-Save-Your-Time-OaM