BINARY TREE

CONCEPT

- Tree?
  - No loop/circle
  - Non-directional
  - Connected graph
- Binary tree?
  - Only contains at most 2 children for a given node

PROBLEM SET

- No. 94 Binary Tree Inorder Traversal
- No. 144 Binary Tree Preorder Traversal
- No. 145 Binary Tree Postorder Traversal
- No. 208 Implement Trie (Prefix Tree)
- No. 211 Add and Search Word
- No. 226 Invert Binary Tree
- No. 297 Serialize and Deserialize Binary Tree
- No. 307 Range Sum Query - Mutable
- No. 530 Minimum Absolute Difference in BST

NO. 94

BINARY TREE INORDER TRAVERSAL

PROBLEM DESCRIPTION

- Given a binary tree, return the inorder traversal of its nodes' values.

IDEA

- Similar to No. 230 Kth Smallest Element in a BST
  - Use a stack to maintain the traversal path
  - Go left until the leaf is reached
  - Pop from stack
  - Go right, and then go left till the leaf…etc

SOLUTION

- https://github.com/Brady31027/leetcode/tree/master/94_Binary_Tree_Inorder_Traversal

NO. 144

BINARY TREE PREORDER TRAVERSAL

PROBLEM DESCRIPTION

- Given a binary tree, return the preorder traversal of its nodes' values.

IDEA

- DFS

```
def dfs(root):
    if not root: return
    ans.append(root.val)
    dfs(root.left)
    dfs(root.right)
```

- Try to use **iterative approach** to solve it

SOLUTION

- https://github.com/Brady31027/leetcode/tree/master/144_Binary_Tree_Preorder_Traversal

NO. 145

BINARY TREE POSTORDER TRAVERSAL

PROBLEM DESCRIPTION

- Given a binary tree, return the postorder traversal of its nodes' values.

IDEA

- Recursively go left first
- Then go right
- Cannot go either left or right, dump node value

SOLUTION

- https://github.com/Brady31027/leetcode/tree/master/145_Binary_Tree_Postorder_Traversal

BRIEF SUMMARY

- in-order

```
def in_order_dfs(root):
    if not root: return
    in_order_dfs(root.left)
    ans.append(root.val)
    in_order_dfs(root.right)
```

- pre-order

```
def pre_order_dfs(root):
    if not root: return
    ans.append(root.val)
    pre_order_dfs(root.left)
    pre_order_dfs(root.right)
```

- post-order

```
def post_order_dfs(root):
    if not root: return
    post_order_dfs(root.left)
    post_order_dfs(root.right)
    ans.append(root.val)
```

NO. 208

IMPLEMENT TRIE (PREFIX TREE)

PROBLEM DESCRIPTION

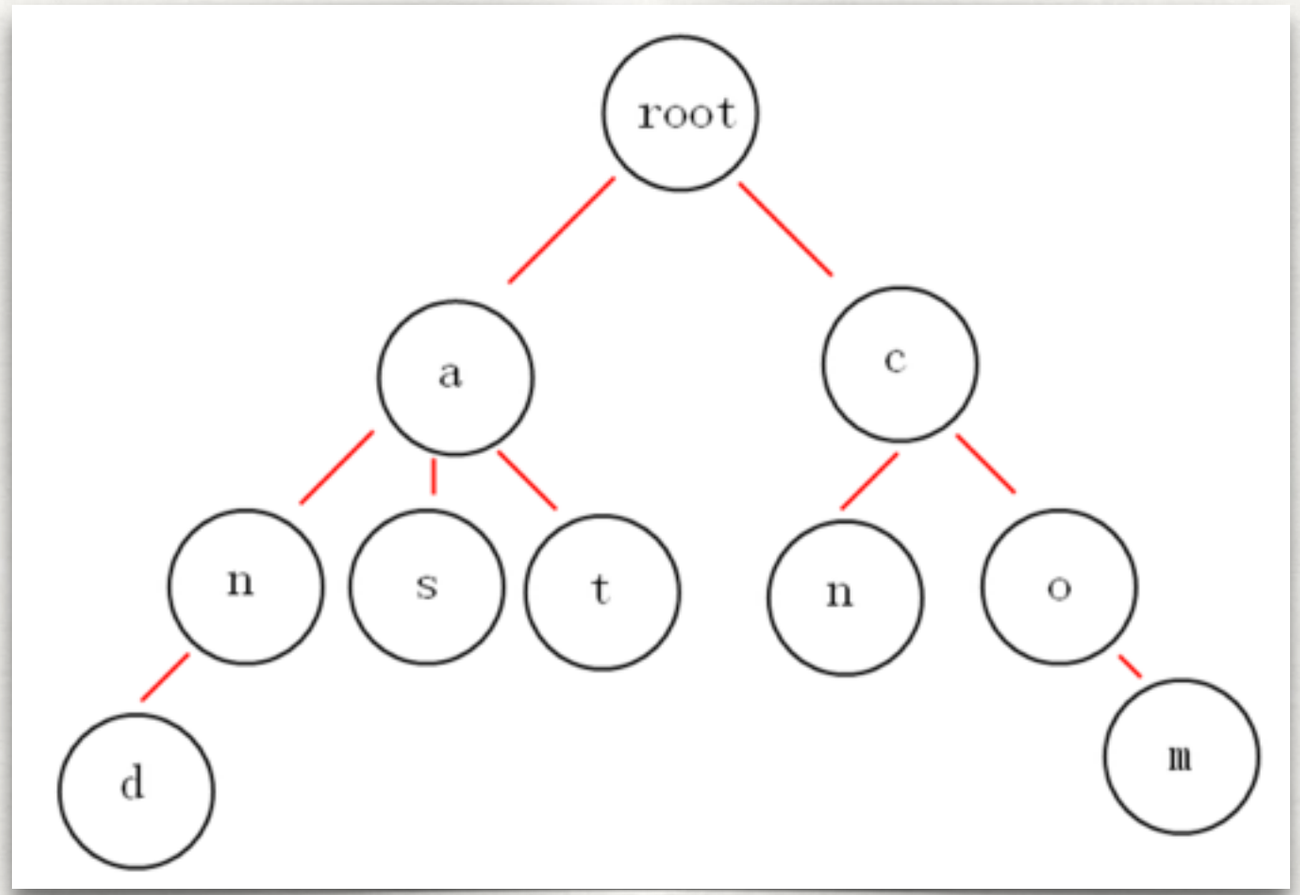
- Implement a trie with insert, search, and startsWith methods.
- Note:
- You may assume that all inputs are consist of lowercase letters a-z.

IDEAS

- In Chinese, we call it “字典樹”
- What is “Trie”? What’s its property? How to build it?
  - http://www.cnblogs.com/huangxincheng/archive/2012/11/25/2788268.html

IDEAS

- Example
  - Given a list of words such as [“and”, “as”, “at”, “cn”, “com”]



IDEAS

- Properties
  - Root doesn’t stand for any character
  - The path from root to any other node can form a string/ token
    - Common prefix of multiple tokens can be extracted as the parent of these tokens
- Why do we use a trie instead of a big hash?
  - Reduce memory usage
  - Reduce search complexity

IDEAS

- Common ops
  - Insertion()
  - Delete()
  - Find()
- Data structure design
  - Use a hash to record its children with the following form : {‘c’ : idx}
  - Use a linked list to record the children
  - Use sibling notation

SOLUTION

- https://github.com/Brady31027/leetcode/tree/master/208_Implement_Trie

NO. 211

ADD AND SEARCH WORD

PROBLEM DESCRIPTION

- Design a data structure that supports the following two operations:
  - void addWord(word)
  - bool search(word)
- Example
  - example:
    - addWord(“bad”); addWord(“dad”); addWord("mad")
    - search("pad") -> false
    - search("bad") -> true
    - search(".ad") -> true
    - search("b..") -> true

IDEA

- Do it after 208. Implement Trie (Prefix Tree)
- Basically the same as No. 208 except
  - Wildcard character (.)
- For search()
  - Use recursion to handle every char one by one
  - if we meet wildcard: use DFS to try every possibility
  - else: search(node.children.get(word[0]), word[1:])

SOLUTION

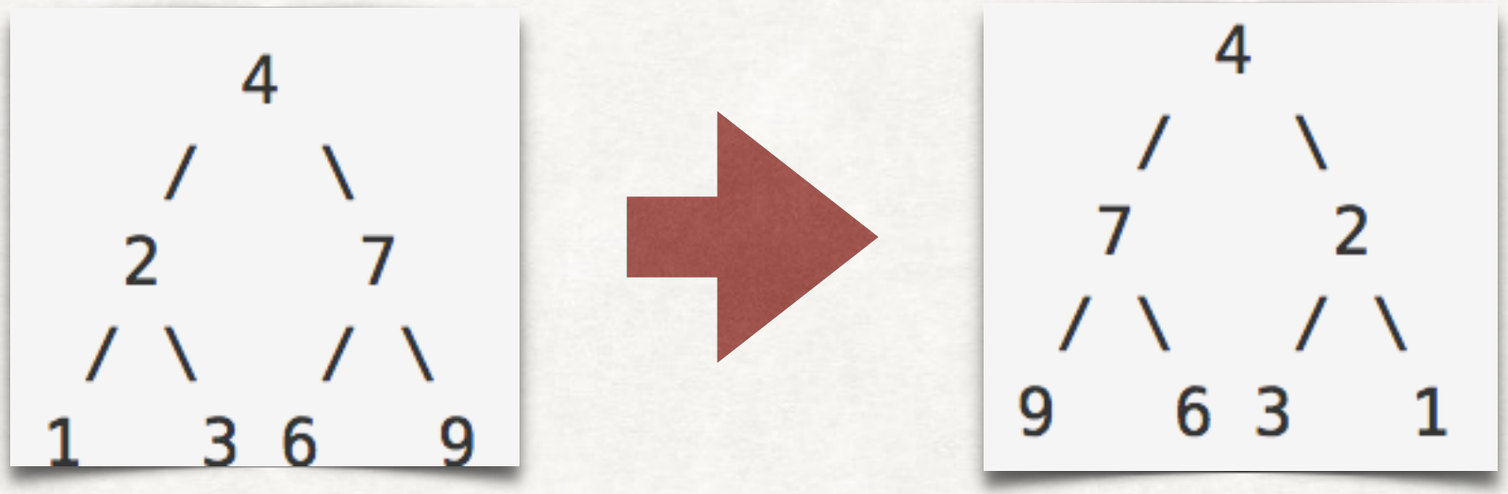
- https://github.com/Brady31027/leetcode/tree/master/211_Add_and_Search_Word_Data_structure_design

NO. 226

INVERT BINARY TREE

PROBLEM DESCRIPTION

- Invert a binary tree



90% of our engineers use the software you wrote (Homebrew), but you can’t invert a binary tree on a whiteboard so fuck off. — Google to M. Howell

IDEA

- BSF
  - pop a node from the tail of the queue
    - swap left and right child nodes
  - if child node is valid, push to queue

SOLUTION

- https://github.com/Brady31027/leetcode/tree/master/226_Invert_Binary_Tree

NO. 297

SERIALIZE AND DESERIALIZE BINARY TREE

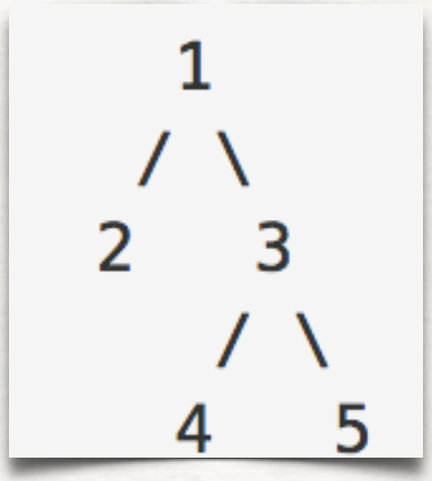
PROBLEM DESCRIPTION

- Serialization is the process of converting a data structure or object into a sequence of bits so that it can be stored in a file or memory buffer, or transmitted across a network connection link to be reconstructed later in the same or another computer environment.

- Design an algorithm to serialize and deserialize a binary tree. There is no restriction on how your serialization/deserialization algorithm should work. You just need to ensure that a binary tree can be serialized to a string and this string can be deserialized to the original tree structure.

IDEA

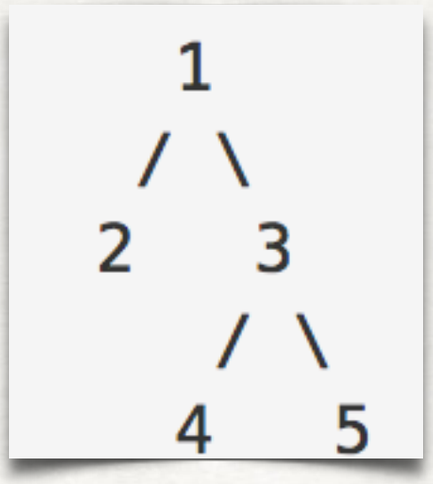
- Serialize by using BFS
  - Special case: leaves
  - E.g. [1,2,3,null,null,4,5]



TLE!

  - Serialized list: [1, 2, 3, '#', '#', 4, 5]
    - Use list instead of string to handle negative node value, i.e **-1**
      - [-1] v.s. [-,1]
- Deserialize by maintaining a list to track the TreeNodes
  - For node *i*, left child index is (*i**2+1), right child index is (*i**2+2)

IDEA



- Serialize by using pre-order traversal
  - Given [1,2,3,null,null,4,5], return “1 2 # # 3 4 # # 5 # #”
    - Concate a list to a string: serialized_string = ' ‘.join(list)
- Deserialize by using pre-order traversal as well
  - Convert input string back to list or iterator
  - Use next(iterator) to fetch every node
  - Recursion orders of serialization and deserialization are the same
    - Go left, then go right

```
def split(input, sep):
    sep_size = len(sep)
    start = 0
    while True:
        length = input.find(sep, start)
        if length == -1:
            yield input[start:]
            return
        yield input[start:length]
        start = length + sep_size
iterators = iter(split(data, ' '))
```

SOLUTION

- https://github.com/Brady31027/leetcode/tree/master/297_Serialize_and_Deserialize_Binary_Tree

NO. 307

RANGE SUM QUERY - MUTABLE

PROBLEM DESCRIPTION

- Given an integer array nums, find the sum of the elements between indices i and j (i ≤ j), inclusive. The update(i, val) function modifies nums by updating the element at index i to val.
- Example
- Given nums = [1, 3, 5]
- sumRange(0, 2) -> 9
- update(1, 2)
- sumRange(0, 2) -> 8

IDEA

- Following up question from 303. Range Sum Query - Immutable
  - https://github.com/Brady31027/leetcode/tree/master/303_Range_Sum_Query_Immutable
- The most straight-forward solution
  - Linear scan

```
def update(self, i, val):
    self.numberArray[i] = val
```

TLE !

```
def sumRange(self, i, j):
    self.returnSum = 0
    while i <= j:
        self.returnSum += self.numberArray[i]
        i += 1
    return self.returnSum
```

IDEA

- Binary indexed tree
  - When to use?
    - Calculate sum of a range within a list/array
      - list/array is updatable except
        - No add/remove nodes
- https://github.com/Brady31027/leetcode/wiki/%5BDS%5D-Binary-Indexed-Tree

SOLUTION

- https://github.com/Brady31027/leetcode/tree/master/307_Range_Sum_Query_Mutable

NO. 530

MINIMUM ABSOLUTE DIFFERENCE IN BST

PROBLEM DESCRIPTION

- Given a binary search tree with non-negative values, find the minimum absolute difference between values of any two nodes.

IDEA

- Traverse this tree and remember nodes' value
  - Sort the value and calculate the distance

Should have a better solution

SOLUTION

- https://github.com/Brady31027/leetcode/tree/master/530_Minimum_Absolute_Difference_in_BST