Code Snippet Repository

Anh Tran - LeetCode

July 1, 2025

1 Python Code Snippets

1.1 Counting number of islands

Count number of islands

```python
# ### BFS
from typing import List
from collections import deque

class Solution:
  def numIslands(self, grid: List[List[str]]) -> int:
    if not grid:
      return 0

    m, n = len(grid), len(grid[0])
    count = 0

    def bfs(i, j):
      # Build a list of to-be-visited pixels
      queue = deque()
      queue.append((i, j))
      grid[i][j] = '0'  # mark as visited

      while queue:
        ii, jj = queue.popleft()
        for di, dj in [(-1, 0), (1, 0), (0, -1), (0, 1)]:  # down, up, left, right
          ni, nj = x + di, y + dj
          if 0 <= ni < m and 0 <= nj < n and grid[ni][nj] == '1':
            queue.append((ni, nj))
            grid[ni][nj] = '0'  # mark as visited

    # Loop through the grid
    for i in range(m):
      for j in range(n):
        if grid[i][j] == '1':
          bfs(i, j)
          count += 1  # finished one island

    return count

## DFS
class Solution:
  def numIslands(self, grid: List[List[str]]) -> int:
    if not grid:
      return 0

    m, n = len(grid), len(grid[0])
    count = 0

    def dfs(i, j):
      if i < 0 or i >= m or j < 0 or j >= n or grid[i][j] != '1':
        return
      grid[i][j] = '0'  # mark visited
      dfs(i+1, j)  # down
      dfs(i-1, j)  # up
      dfs(i, j+1)  # right
      dfs(i, j-1)  # left

    for i in range(m):
      for j in range(n):
        if grid[i][j] == '1':
          dfs(i, j)
          count += 1

    return count
```

1.2 Average of levels in binary tree

Average of levels in binary tree

```python
# Definition for a binary tree node.
# class TreeNode:
#   def __init__(self, val=0, left=None, right=None):
#     self.val = val
#     self.left = left
#     self.right = right
from typing import List, Optional
from collections import deque

class Solution:
  def averageOfLevels(self, root: Optional[TreeNode]) -> List[float]:
    if not root:
      return []

    result = []
    queue = deque([root])  # Start with the root node in the queue

    while queue:
      level_sum = 0
      level_count = len(queue)  # Number of nodes at the current level

      # Process all nodes at the current level
      for _ in range(level_count):
        node = queue.popleft()  # Pop the front node from the queue
        level_sum += node.val  # Add its value to the level sum

        # Enqueue left and right children if they exist
        if node.left:
          queue.append(node.left)
        if node.right:
          queue.append(node.right)

      # Compute the average for this level
      result.append(level_sum / level_count)

    return result
```

1.3 Two-sum

Two sum

```python
class Solution:
  def twoSum(self, nums: List[int], target: int) -> List[int]:
    # Dictionary to store the number as the key and its index as the value
    num_to_index = {}

    # Iterate through the list of numbers with their indices
    for i, num in enumerate(nums):
      # Calculate the complement of the current number
      x = target - num

      # Check if the complement is already in the dictionary
      if x in num_to_index:
        # If found, return the indices of the complement and the current number
        return [num_to_index[x], i]

      # If the complement is not found, store the current number and its index in the dictionary
      num_to_index[num] = i

    # If no pair is found, return [-1, -1] indicating failure
    return [-1, -1]
```

1.4 Max Area of Island

Breadth first searchs

```python
from collections import deque

class Solution:
  def maxAreaOfIsland(self, grid: List[List[int]]) -> int:
    m = len(grid)
    n = len(grid[0])
    count = 0
    self.max_area = -float('inf')

    def bfs(i,j):
      queue = deque()
      queue.append((i,j))
      grid[i][j] = 0 # marked as visited
      area = 1

      while queue:
        ii, jj = queue.popleft()
        for di, dj in [(-1, 0), (1, 0), (0, 1), (0, -1)]:
          ni, nj = ii + di, jj + dj
          if 0 <= ni < m and 0 <= nj < n and grid[ni][nj] == 1:
            queue.append((ni, nj))
            grid[ni][nj] = 0 # marked as visited
            area += 1
      self.max_area = max(self.max_area, area)

    for i in range(m):
      for j in range(n):
        if grid[i][j] == 1:
          bfs(i,j)
          count += 1
    return max(self.max_area, 0)
```

1.5 Valid word abbreviation

Valid word abbreviation

```python
class Solution:
  def validWordAbbreviation(self, word: str, abbr: str) -> bool
      ↪ :
    i, j = 0, 0  # i for word, j for abbr

    while i < len(word) and j < len(abbr):
      if abbr[j].isalpha():
        # If the current character in abbr is a letter, it must
            ↪  match the word
        if word[i] != abbr[j]:
          return False
        i += 1
        j += 1
      else:
        # If the current character in abbr is a digit, handle
            ↪ it
        if abbr[j] == '0' and (j == 0 or not abbr[j-1].isdigit
            ↪ ()):  # no leading zeros
          return False
        # Extract the number from abbr (could be more than one
            ↪ digit)
        num = 0
        while j < len(abbr) and abbr[j].isdigit():
          num = num * 10 + int(abbr[j])
          j += 1
        i += num  # skip the corresponding number of characters
            ↪  in word

    # If both pointers have reached the end, it is a valid
        ↪ abbreviation
    return i == len(word) and j == len(abbr)
```

1.6 Check if path exists in graph

check path exists in graph

```python
from collections import defaultdict

class Solution:
  def validPath(self, n: int, edges: List[List[int]], source:
      ↪ int, destination: int) -> bool:
    # Build the adjacency list representation of the graph
    graph = defaultdict(list)
    for u, v in edges:
      graph[u].append(v)
      graph[v].append(u)

    # DFS function to explore the graph
    def dfs(node, visited):
      if node == destination:
        return True
      visited.add(node)
      for neighbor in graph[node]:
        if neighbor not in visited:
          if dfs(neighbor, visited):
            return True
      return False

    # Perform DFS starting from source
    visited = set()
    return dfs(source, visited)
```

1.7 Kadane algorithm - max sub array

Max sub array

```python
class Solution:
  def maxSubArray(self, nums: List[int]) -> int:
    # https://www.geeksforgeeks.org/python/python-program-for-
        ↪ largest-sum-contiguous-subarray/
    max_so_far = float('-inf')
    max_ending_here = 0

    for i in range(0, len(nums)):
      max_ending_here = max_ending_here + nums[i]
      if (max_so_far < max_ending_here):
        max_so_far = max_ending_here
      if max_ending_here < 0:
        max_ending_here = 0
    return max_so_far

class Solution:
  def maxSubArray(self, nums: List[int]) -> int:
    # Kadane algorithm: https://en.wikipedia.org/wiki/
        ↪ Maximum_subarray_problem
    best_sum = -float('inf')  # Initialize to a very small
        ↪ number
    current_sum = 0
    for x in nums:
      # Update the current_sum to be the maximum of the current
          ↪  element alone or adding it to the current_sum
      current_sum = max(x, current_sum + x)
      # # Update the best_sum to store the maximum value
          ↪ encountered so far
      best_sum = max(best_sum, current_sum)

    # Return the best_sum after the loop has finished
        ↪ processing all elements
    return best_sum
```

1.8 Climbing stair

Climbing stairs

```python
class Solution:
  def climbStairs(self, n: int) -> int:
    # Base cases: 1 way to climb 1 step, 2 ways to climb 2 steps
    if n == 1:
      return 1
    if n == 2:
      return 2

    # prev_one: ways to reach the previous step (n-1)
    # prev_two: ways to reach two steps before (n-2)
    prev_two = 1   # ways to reach step 1
    prev_one = 2   # ways to reach step 2

    # Calculate number of ways for each step from 3 to n
    for current_step in range(3, n + 1):
      current_ways = prev_one + prev_two  # total ways to reach current step
      prev_two = prev_one         # update for next iteration
      prev_one = current_ways

    return prev_one  # this is the result for n steps
```

1.9 Longest harmonious subsequence

Longest harmonic sequence

```python
from collections import Counter

class Solution:
  def findLHS(self, nums: List[int]) -> int:
    # Count the frequency of each element
    freq = Counter(nums)

    max_len = 0  # Initialize the max length to 0

    # Iterate through the frequency map
    for num in freq:
      # Check if the next consecutive number (num + 1) exists in the map
      if num + 1 in freq:
        # Calculate the length of the harmonious subsequence
        max_len = max(max_len, freq[num] + freq[num + 1])

    return max_len
```

1.10 Rotate string

Rotate string

```python
from collections import Counter, deque

class Solution:
  def rotateString(self, s: str, goal: str) -> bool:
    if Counter(s) != Counter(goal) or len(s) != len(goal):
      return False
    n = len(s)
    queue_g = deque(goal)
    queue_s = deque(s)
    for i in range(n):
      if queue_s == queue_g:
        return True
      else:
        last = queue_g.pop()
        queue_g.appendleft(last)
    return False
```

1.11 Isomorphic string

Isomorphic string

```python
class Solution:
  def isIsomorphic(self, s: str, t: str) -> bool:
    char_to_index_s = {}
    char_to_index_t = {}

    for i in range(len(s)):
      if s[i] not in char_to_index_s:
        char_to_index_s[s[i]] = i

      if t[i] not in char_to_index_t:
        char_to_index_t[t[i]] = i

      if char_to_index_s[s[i]] != char_to_index_t[t[i]]:
        return False
    return True
```

1.12 Happy number

Happy number: sum digits

```python
class Solution:
  # 1**2 + 9**2 = 82
  # 8**2 + 2**2 = 68
  # 6**2 + 8**2 = 100
  # 1**2 + 0**2 + 02 = 1
  def isHappy(self, n: int) -> bool:
    seen = set()  # To track previously seen sums
    while n != 1:
      if n in seen:  # If we've seen this number before, we're in a cycle
        return False
      seen.add(n)
      # Calculate the sum of the squares of the digits of n
      n = sum(int(digit) ** 2 for digit in str(n))
    return True
```

1.13 Search in binary tree

Search in binary tree

```python
class Solution:
  def searchBST(self, root: Optional[TreeNode], val: int) -> Optional[TreeNode]:
    node = root

    while node:
      if node.val == val:
        return node
      if node.val < val:
        node = node.right
      else:
        node = node.left

    return None
```

1.14 Reverse linked list

Reverse linked list

```python
class Solution:
  def reverseList(self, head: Optional[ListNode]) -> Optional[ListNode]:
    # Init 3 pointers: curr, prev, next_node
    prev = None
    curr = head

    while curr: #  is not None:
      next_node = curr.next
      curr.next = prev
      # Advance curr and prev pointers
      prev = curr
      curr = next_node
    return prev # return prev, not curr
```

1.15 Top k frequent

Top k

```python
import heapq
class Solution:
  def topKFrequent(self, nums: List[int], k: int) -> List[int]:
    freq = {}
    for num in nums:
      freq[num] = 1 + freq.get(num, 0)

    heap = [(-v, k) for k, v in freq.items()]
    heapq.heapify(heap)
    return [heapq.heappop(heap)[1] for _ in range(k)]
```

1.16 Reverse integer

Reverse integer

```python
class Solution:
  def reverse(self, x: int) -> int:
    sign = -1 if x<0 else +1
    x = abs(x)
    reversed_x = int( str(x)[::-1] )
    if reversed_x > 2**31-1:
      return 0
    else:
      return sign*reversed_x
```

1.17 Best time to buy and sell stock

Best time buy/sell stock

```python
class Solution:
  def maxProfit(self, prices: List[int]) -> int:
    min_price = float('inf')
    max_profit = 0
    for price in prices:
      if price < min_price:
        min_price = price
      else:
        max_profit = max(max_profit, price - min_price)
    return max_profit
```

1.18 Roman to integer

Roman to integer

```python
class Solution:
  def romanToInt(self, s: str) -> int:
    assert 1 <= len(s) <= 15, "invalid input"
    roman_int_dict = {'I': 1, "V": 5, "X": 10, 'L': 50,'C':100,
        ↪ 'D': 500,'M':1000}
    num = 0
    i = 0
    while i < len(s):
      if i < len(s) - 1 and roman_int_dict[s[i]] <
          ↪ roman_int_dict[s[i+1]]:
        num += (roman_int_dict[s[i+1]] - roman_int_dict[s[i]])
        i += 2
      else:
        num += roman_int_dict[s[i]]
        i += 1
    return num
```

1.19 Add two numbers

Add two numbers

```python
class Solution:
  def addTwoNumbers(self, l1: Optional[ListNode], l2: Optional[
      ↪ ListNode]) -> Optional[ListNode]:
    dummy_head = ListNode()
    current = dummy_head
    carry = 0

    while l1 or l2 or carry:
      val1 = l1.val if l1 else 0
      val2 = l2.val if l2 else 0

      total = val1 + val2 + carry
      carry = total // 10
      digit = total % 10

      current.next = ListNode(digit)
      current = current.next

      if l1: l1 = l1.next
      if l2: l2 = l2.next

    return dummy_head.next
```

1.20 Tree path sum

Tree path sum

```python
class Solution:
  def hasPathSum(self, root: Optional[TreeNode], targetSum: int
      ↪ ) -> bool:
    # Base case: if the root is None, no path exists
    if not root:
      return False

    # Subtract the current node's value from the target sum
    targetSum -= root.val

    # If we reached a leaf node (both left and right children
        ↪ are None)
    if not root.left and not root.right:
      return targetSum == 0  # If the target sum becomes 0,
          ↪ return True

    # Recursively check both left and right subtrees
    return (self.hasPathSum(root.left, targetSum) or
        self.hasPathSum(root.right, targetSum))
```

1.21 k distant indices

Add two numbers

```python
class Solution:
  def findKDistantIndices(self, nums: List[int], key: int, k:
      ↪ int) -> List[int]:
    result = set()
    key_indices = []

    for i, num in enumerate(nums):
      if num == key:
        key_indices.append(i)

    for key_idx in key_indices:
      for i in range(max(0, key_idx - k), min(len(nums),
          ↪ key_idx + k + 1)):
        result.add(i)

    return sorted(result)
```

1.22 Max difference

Max difference

```python
class Solution:
  def maximumDifference(self, nums: List[int]) -> int:
    max_diff = -1  # Initialize max_diff to -1, as per problem statement
    i = 0

    # Iterate through the array to find the maximum difference
    for j in range(1, len(nums)):
      if nums[j] > nums[i]:
        max_diff = max(max_diff, nums[j] - nums[i])
      else:
        i = j  # Move 'i' to 'j' if we find a smaller value, so that 'nums[j]' can be larger than 'nums[i]'

    return max_diff
```

1.23 Binary search

Binary Search

```python
class Solution:
  def search(self, nums: List[int], target: int) -> int:
    left = 0
    right = len(nums) - 1
    while left <= right:
      mid = (left + right) // 2
      if target == nums[mid]:
        return mid
      elif target > nums[mid]:
        left = mid + 1
      else:
        right = mid - 1
    return -1
```

1.24 Koko eating bananas

Koko eating bananas (binary search)

```python
import math
class Solution:
  def minEatingSpeed(self, piles: List[int], h: int) -> int:
    left, right = 1, max(piles)
    while left < right:
      mid = (left + right) // 2
      # Calculate round up ceil() number of hours
      # hours = sum((pile + mid - 1) // mid for pile in piles)
      hours = sum(math.ceil(pile / mid) for pile in piles)
      if hours > h:
        left = mid + 1
      else:
        right = mid
    return left
```

1.25 Check valid parentheses

Check valid parentheses

```python
class Solution:
  def isValid(self, s: str) -> bool:
    stack = []
    # Dictionary to match opening and closing brackets
    bracket_map = {')': '(', '}': '{', ']': '['}

    for char in s:
      if char in bracket_map.values():  # If it's an opening bracket
        stack.append(char)
      elif char in bracket_map.keys():  # If it's a closing bracket
        # If the stack is empty or the top of the stack is not the matching opening bracket
        if not stack or stack.pop() != bracket_map[char]:
          return False
    # If the stack is empty, all the brackets were properly matched
    return not stack
```

1.26 k distant indices

k distant indices

```python
class Solution:
  def findKDistantIndices(self, nums: List[int], key: int, k: int) -> List[int]:
    result = set()
    key_indices = []

    for i, num in enumerate(nums):
      if num == key:
        key_indices.append(i)

    for key_idx in key_indices:
      for i in range(max(0, key_idx - k), min(len(nums), key_idx + k + 1)):
        result.add(i)

    return sorted(result)
```

1.27 Move zeroes

Move zeroes

```python
class Solution:
  def moveZeroes(self, nums: List[int]) -> None:
    """
    Do not return anything, modify nums in-place instead.
    """
    ptr = 0
    for num in nums:
      if num != 0:
        nums[ptr] = num
        ptr += 1
    for i in range(ptr, len(nums)):
      nums[i] = 0
```

1.28 Plus one with numbers as lists

Plus one

```python
class Solution:
  def plusOne(self, digits: List[int]) -> List[int]:
    # Traverse the digits from right to left
    for i in range(len(digits) - 1, -1, -1):
      if digits[i] < 9:
        digits[i] += 1
        return digits  # Return once hit any number < 9
      digits[i] = 0  # Set current digit to 0 if there's a carry

    # If all digits are 9, we need to add a 1 at the beginning
    return [1] + digits
```

1.29 Implement stack using queues

Stack using queues

```python
class MyStack:
  def __init__(self):
    self.stack = []
  def push(self, x: int) -> None:
    self.stack.append(x)
  def pop(self) -> int:
    return self.stack.pop() if not self.empty() else -1
  def top(self) -> int:
    return self.stack[-1] if not self.empty() else -1
  def empty(self) -> bool:
    return len(self.stack) == 0
```

1.30 Contains duplicate I

Contain duplicate I

```python
class Solution:
  def containsDuplicate(self, nums: List[int]) -> bool:
    seen = set()
    for i in nums:
      if i in seen:
        return True
      else:
        seen.add(i)
    return False
```

1.31 Contains duplicate II

Contain duplicate II

```python
class Solution:
  def containsNearbyDuplicate(self, nums: List[int], k: int) ->
      ↪  bool:
    seen_num2idx = {}
    for i in range(len(nums)):
      if nums[i] in seen_num2idx and i - seen_num2idx[nums[i]]
          ↪ <= k:
        return True
      seen_num2idx[nums[i]] = i
    return False
```

1.32 Middle linked list

Middle linked list

```python
class Solution:
  def middleNode(self, head: Optional[ListNode]) -> Optional[
      ↪ ListNode]:
    slow = head
    fast = head
    while fast and fast.next:
      slow = slow.next
      fast = fast.next.next
    return slow
```

1.33 Find numbers with even number of digits

Find numbers with even number of digits

```python
class Solution:
  def findNumbers(self, nums: List[int]) -> int:
    count = 0
    for num in nums:
      if len(str(num)) % 2 == 0:
        count += 1
    return count
```

1.34 Find subsequence of length k with largest sum

Find subsequence of length k with largest sum

```python
class Solution:
  def maxSubsequence(self, nums: List[int], k: int) -> List[int
      ↪ ]:
    indexed_nums = [(num, i) for i, num in enumerate(nums)]
    sorted_nums = sorted(indexed_nums, key=lambda x: x[0],
        ↪ reverse=True)[:k]
    indices = [index for num, index in sorted_nums]
    indices.sort()
    return [nums[i] for i in indices]
```

1.35 Is palindrome (number)

Is palindrome (number)

```python
class Solution:
  def isPalindrome(self, x: int) -> bool:
    if x < 0:
      return False
    elif x == 0:
      return True
    else:
      s = str(x)
      n = len(s)
      for i in range(n // 2 + 1):
        if s[i] != s[n - 1 - i]:
          return False
      else:
        return True
```

1.36 Last stone weight

Last stone weight

```python
import heapq
class Solution:
  def lastStoneWeight(self, stones: List[int]) -> int:
    heap = [-stone for stone in stones]
    heapq.heapify(heap)

    while len(heap) > 1:
      first = - heapq.heappop(heap)
      second = - heapq.heappop(heap)

      if first > second:
        heapq.heappush(heap, -(first - second))
    return -heap[0] if heap else 0
```

1.37 Is palindrome (string)

Is palindrome (string)

```python
class Solution:
  def isPalindrome(self, s: str) -> bool:
    filtered = [char.lower() for char in s if char.isalnum()]
    return filtered == filtered[::-1]
```

1.38 Is palindrome (linked list)

Is palindrome (linked list)

```python
class Solution:
  def isPalindrome(self, head: Optional[ListNode]) -> bool:
    res = []
    curr = head
    while(curr):
      res.append(curr.val)
      curr = curr.next
    return res == res[::-1]
```

1.39 Merge sorted array

Merge sorted array

```
class Solution:
  def merge(self, nums1: List[int], m: int, nums2: List[int], n
      ↪ : int) -> None:
    i = m - 1  # pointer for nums1
    j = n - 1  # pointer for nums2
    k = m + n - 1  # pointer for placement in nums1

    # Merge in reverse order
    while i >= 0 and j >= 0:
      if nums1[i] > nums2[j]:
        nums1[k] = nums1[i]
        i -= 1
      elif nums1[i] <= nums2[j]:
        nums1[k] = nums2[j]
        j -= 1
      k -= 1

    # If any remaining in nums2, copy them over
    while j >= 0:
      nums1[k] = nums2[j]
      j -= 1
      k -= 1
```

1.40 First unique char

First unique char

```
from collections import Counter
class Solution:
  def firstUniqChar(self, s: str) -> int:
    freq = Counter(s)
    for i, char in enumerate(s):
      if freq[char] == 1:
        return i
    return -1
```

1.41 Invert binary tree

Invert binary tree

```
class Solution:
  def invertTree(self, root: Optional[TreeNode]) -> Optional[
      ↪ TreeNode]:
    if root:
      root.left, root.right = root.right, root.left
      self.invertTree(root.left)
      self.invertTree(root.right)
    return root
```

1.42 Missing number

Missing unique number

```
class Solution:
  def missingNumber(self, nums: List[int]) -> int:
    nums.sort()
    for i in range(0,len(nums)):
      if nums[i] != i:
        return i
    return len(nums)
```

1.43 Length of last word

Length of last word

```
class Solution:
    def lengthOfLastWord(self, s: str) -> int:
        return len(s.split()[-1])
```

1.44 Valid anagram

Valid anagram

```
from collections import Counter
class Solution:
  def isAnagram(self, s: str, t: str) -> bool:
    return True if Counter(s)==Counter(t) else False
```

1.45 Sum of unique elements

Sum of unique elements

```
from collections import Counter
class Solution:
  def sumOfUnique(self, nums: List[int]) -> int:
    tmp = []
    for k, v in Counter(nums).items():
      if v == 1:
        tmp.append(k)
    return sum(tmp)
```

1.46 Maximum difference between adjacent elements in a circular array

Maximum diff between adjacent elements

```
class Solution:
  def maxAdjacentDistance(self, nums: List[int]) -> int:
    max_diff = 0
    n = len(nums)
    for i in range(n):
      max_diff = max(max_diff, abs(nums[(i+1)%n] - nums[i]))
    return max_diff
```