Code Snippet Repository

Anh Tran - LeetCode

June 30, 2025

1 Python Code Snippets

1.1 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
        ↪ )]
```

1.2 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
```

1.3 Reverse linked list

Roman to integer

```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
```

1.4 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
```

1.5 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
```

1.6 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
```

1.7 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]
```

1.8 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
```

1.9 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
```

1.10 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
```

1.11 Contains duplicate

Two sum

```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
```

1.12 Plus one

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
```

1.13 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
```

1.14 Is palindrome (string)

Merge sorted array

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

1.15 Is palindrome (linked list)

Merge sorted array

```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]
```

1.16 Merge sorted array

Merge sorted array

```python
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
```

1.17 First unique char

First unique char

```python
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
```

1.18 Max Area of Island

Breadth first searchs

```
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)
```

1.19 Counting number of islands

Count number of islands

```
# ### 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
```