22-06-2024 AN session

1.Height balanced binary search tree

class TreeNode:

def \_init\_(self, val=0, left=None, right=None):

self.val = val

self.left = left

self.right = right

def sortedArrayToBST(nums):

if not nums:

return None

mid = len(nums) //

root = TreeNode(nums[mid])

root.left = sortedArrayToBST(nums[:mid])

root.right = sortedArrayToBST(nums[mid+1:])

return root

2.Substring

def stringMatching(words):

return [word for word in words if any(other\_word.find(word) != -1 for other\_word in words if word != other\_word)]

# Example

words = ["mass", "as", "hero", "superhero"]

output = stringMatching(words)

print(output)

3.Find the median of two sorted arrays

def findMedianSortedArrays(nums1, nums2):

nums = sorted(nums1 + nums2)

n = len(nums)

if n % 2 == 0:

return (nums[n // 2 - 1] + nums[n // 2]) / 2

else:

return nums[n // 2]

nums1 = [1, 3]

nums2 = [2]

print(findMedianSortedArrays(nums1, nums2))

4.M\*N Binary matrix

from queue import PriorityQueue

class ListNode:

def \_init\_(self, val=0, next=None):

self.val = val

self.next = next

def mergeKLists(lists):

dummy = ListNode(0)

curr = dummy

q = PriorityQueue()

for l in lists:

if l:

q.put((l.val, l))

while not q.empty():

val, node = q.get()

curr.next = ListNode(val)

curr = curr.next

node = node.next

if node:

q.put((node.val, node))

return dummy.next

# Example

lists = [[1,4,5],[1,3,4],[2,6]]

merged\_list = mergeKLists(lists)

result = []

while merged\_list:

result.append(merged\_list.val)

merged\_list = merged\_list.next

print(result)

5.Priority queue

from queue import PriorityQueue

class ListNode:

def \_init\_(self, val=0, next=None):

self.val = val

self.next = next

def mergeKLists(lists):

dummy = ListNode(0)

curr = dummy

q = PriorityQueue()

for l in lists:

if l:

q.put((l.val, l))

while not q.empty():

val, node = q.get()

curr.next = ListNode(val)

curr = curr.next

node = node.next

if node:

q.put((node.val, node))

return dummy.next

# Example

lists = [[1,4,5],[1,3,4],[2,6]]

merged\_list = mergeKLists(lists)

result = []

while merged\_list:

result.append(merged\_list.val)

merged\_list = merged\_list.next

print(result)