1. Given an integer array nums, return *an array* answer *such that* answer[i] *is equal to the product of all the elements of* nums *except* nums[i].

   The product of any prefix or suffix of nums is guaranteed to fit in a 32-bit integer.

   You must write an algorithm that runs in O(n) time and without using the division operation.

   Example 1:

   Input: nums = [1,2,3,4]

   Output: [24,12,8,6]

   Example 2:

   Input: nums = [-1,1,0,-3,3]

   Output: [0,0,9,0,0]

   Constraints:

   - 2 <= nums.length <= 10^5
   - -30 <= nums[i] <= 30
   - The product of any prefix or suffix of nums is guaranteed to fit in a 32-bit integer.

2. Write an efficient algorithm that searches for a target value in an m x n integer matrix. This matrix has the following properties:
   - Integers in each row are sorted in ascending order from left to right.
   - Integers in each column are sorted in ascending order from top to bottom.

Example 1:

1	4	7	11	15
2	5	8	12	19
3	6	9	16	22
10	13	14	17	24
18	21	23	26	30

Input: matrix = [[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],[18,21,23,26,30]], target = 5

Output: true

Example 2:

1	4	7	11	15
2	5	8	12	19
3	6	9	16	22
10	13	14	17	24
18	21	23	26	30

Input: matrix = [[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],[18,21,23,26,30]], target = 20

Output: false

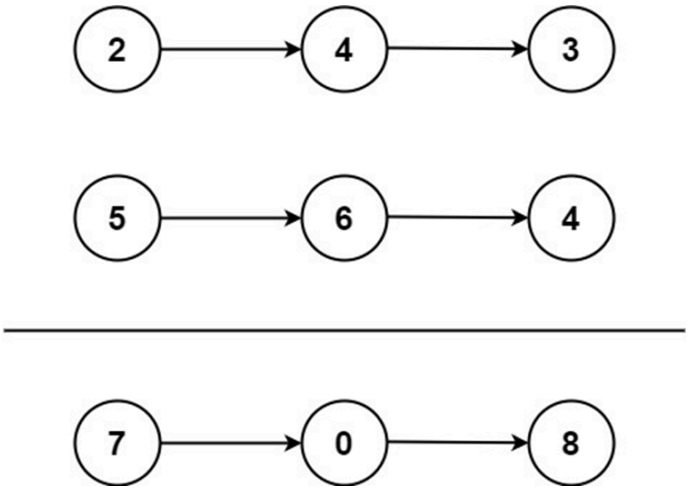
Constraints:

- m == matrix.length
- n == matrix[i].length
- 1<= n, m <= 300
- -10^9 <= matrix[i][j] <= 10^9
- All the integers in each row are sorted in ascending order.
- All the integers in each column are sorted in ascending order.
- -10^9 <= target <= 10^9

3. You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Example 1:



Input: l1 = [2,4,3], l2 = [5,6,4]

Output: [7,0,8]

Explanation: 342 + 465 = 807.

Example 2:

Input: l1 = [0], l2 = [0]

Output: [0]

Example 3:

Input: l1 = [9,9,9,9,9,9,9], l2 = [9,9,9,9]

Output: [8,9,9,9,0,0,0,1]

Constraints:

- The number of nodes in each linked list is in the range [1, 100].
- 0 <= Node.val <= 9
- It is guaranteed that the list represents a number that does not have leading zeros.

4. You are given an array of k linked-lists lists, each linked-list is sorted in ascending order.

Merge all the linked lists into one sorted linked list and return it.

Example 1:

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

Output: [1,1,2,3,4,4,5,6]

Explanation: The linked-lists are:

[

1->4->5,

1->3->4,

2->6

]

merging them into one sorted list:

1->1->2->3->4->4->5->6

Example 2:

Input: lists = []

Output: []

Example 3:

Input: lists = [[]]

Output: []

Constraints:

- k == lists.length
- 0 <= k <= 10^4
- 0 <= lists[i].length <= 500
- -10^4 <= lists[i][j] <= 10^4
- lists[i] is sorted in ascending order.
- The sum of lists[i]. length will not exceed 10^4.

5. Design a data structure that follows the constraints of a Least Recently Used (LRU) cache.

Implement the LRUCache class:

- LRUCache(int capacity) Initialise the LRU cache with a positive size capacity.
- int get(int key) Return the value of the key if the key exists, otherwise return -1.
- void put(int key, int value) Update the value of the key if the key exists. Otherwise, add the key-value pair to the cache. If the number of keys exceeds the capacity of this operation, evict the least recently used key.

The functions get and put must each run in O(1) average time complexity.

Example 1:
Input
["LRUCache", "put", "put", "get", "put", "get", "put", "get", "get", "get"]

[[2], [1, 1], [2, 2], [1], [3, 3], [2], [4, 4], [1], [3], [4]]

Output

[null, null, null, 1, null, -1, null, -1, 3, 4]

Explanation

LRUCache lRUCache = new LRUCache(2);

lRUCache.put(1, 1); // cache is {1=1}

lRUCache.put(2, 2); // cache is {1=1, 2=2}

lRUCache.get(1);   // return 1

lRUCache.put(3, 3); // LRU key was 2, evicts key 2, cache is {1=1, 3=3}

lRUCache.get(2);   // returns -1 (not found)

lRUCache.put(4, 4); // LRU key was 1, evicts key 1, cache is {4=4, 3=3}

lRUCache.get(1);   // return -1 (not found)

lRUCache.get(3);   // return 3

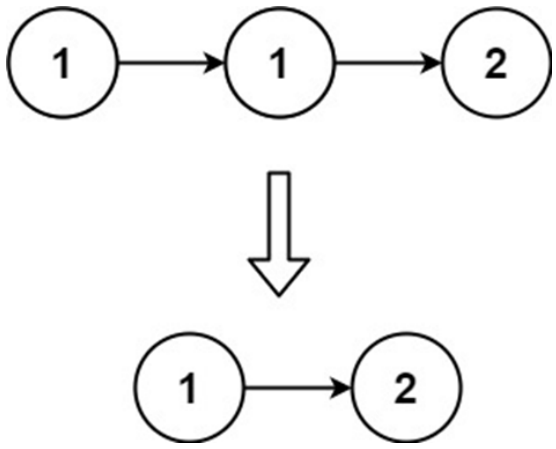
lRUCache.get(4);   // return 4

Constraints:

- 1 <= capacity <= 3000
- 0 <= key <= 10^4
- 0 <= value <= 10^5
- At most 2 * 10^5 calls will be made to get and put.

6. Given the head of a sorted linked list, *delete all duplicates such that each element appears only once.* Return *the linked list sorted as well.*

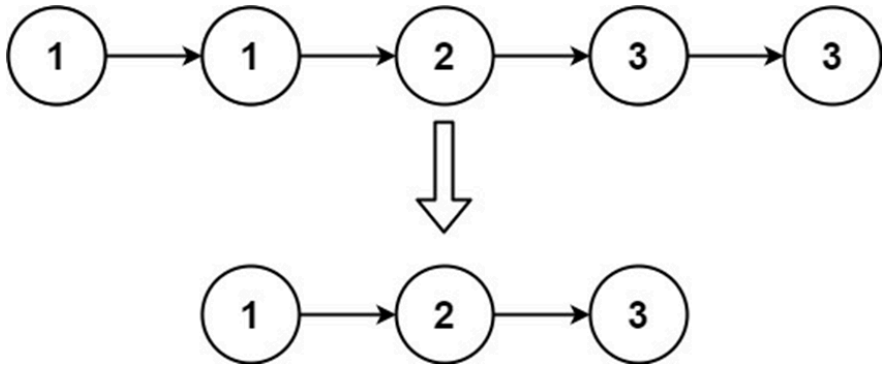
Example 1:



Input: head = [1,1,2]

Output: [1,2]

Example 2:



Input: head = [1,1,2,3,3]

Output: [1,2,3]

Constraints:

- The number of nodes in the list is in the range [0, 300].
- -100 <= Node.val <= 100
- The list is guaranteed to be sorted in ascending order.

7. You are given a string allowed consisting of distinct characters and an array of strings words. A string is consistent if all characters in the string appear in the string allowed.
Return *the number of consistent strings in the array* words.

Example 1:

Input: allowed = "ab", words = ["ad","bd","aaab","baa","badab"]

Output: 2

Explanation: Strings "aaab" and "baa" are consistent since they only contain characters 'a' and 'b'.

Example 2:
Input: allowed = "abc", words = ["a","b","c","ab","ac","bc","abc"]
Output: 7
Explanation: All strings are consistent.

Example 3:
Input: allowed = "cad", words = ["cc","acd","b","ba","bac","bad","ac","d"]
Output: 4
Explanation: Strings "cc", "acd", "ac", and "d" are consistent.

Constraints:

- 1 <= words.length <= 10^4
- 1 <= allowed.length <= 26
- 1 <= words[i].length <= 10
- The characters in allowed are distinct.
- words[i] and allowed contain only lowercase English letters.

8. You are given a string consisting of lowercase English letters. A duplicate removal consists of choosing two adjacent and equal letters and removing them. We repeatedly make duplicate removals on s until we can no longer. Return the final string after all such duplicate removals have been made. It can be proven that the answer is unique.

Example 1:

Input: "abbaca"
Output: "ca"

Explanation: In "abbaca" we could remove "bb" since the letters are adjacent and equal, and this is the only possible move. The result of this move is that the string is "aaca", of which only "aa" is possible, so the final string is "ca".

Example 2:

Input: "azxxzy"
Output: "ay"