**1136. Parallel Courses**

Hard

304FavoriteShare

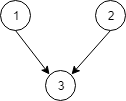
There are N courses, labelled from 1 to N.

We are given relations[i] = [X, Y], representing a prerequisite relationship between course X and course Y: course X has to be studied before course Y.

In one semester you can study any number of courses as long as you have studied all the prerequisites for the course you are studying.

Return the minimum number of semesters needed to study all courses.  If there is no way to study all the courses, return -1.

**Example 1:**

****

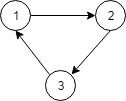
**Input:** N = 3, relations = [[1,3],[2,3]]

**Output:** 2

**Explanation:**

In the first semester, courses 1 and 2 are studied. In the second semester, course 3 is studied.

**Example 2:**

****

**Input:** N = 3, relations = [[1,2],[2,3],[3,1]]

**Output:** -1

**Explanation:**

No course can be studied because they depend on each other.

**Note:**

1. 1 <= N <= 5000
2. 1 <= relations.length <= 5000
3. relations[i][0] != relations[i][1]
4. There are no repeated relations in the input.

class Solution:

def minimumSemesters(self, N: int, relations: List[List[int]]) -> int:

child = collections.defaultdict(list)

parents = [0]\*N

for x,y in relations:

child[x].append(y)

parents[y-1] += 1

result = 0

can\_take = []

for key,val in enumerate(parents):

if val == 0: can\_take.append(key+1)

taken = 0

while can\_take:

result += 1

temp = []

taken += len(can\_take)

for x in can\_take:

for y in child[x]:

parents[y-1] -= 1

if parents[y-1] == 0: temp.append(y)

can\_take = temp

if taken != N: return -1

else: return result

**1121. Divide Array Into Increasing Sequences**

Hard

2910FavoriteShare

Given a **non-decreasing** array of positive integers nums and an integer K, find out if this array can be divided into one or more **disjoint increasing subsequences of length at least** K.

**Example 1:**

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

**Output:** true

**Explanation:**

The array can be divided into the two subsequences [1,2,3,4] and [2,3,4] with lengths at least 3 each.

**Example 2:**

**Input:** nums = [5,6,6,7,8], K = 3

**Output:** false

**Explanation:**

There is no way to divide the array using the conditions required.

**Note:**

1. 1 <= nums.length <= 10^5
2. 1 <= K <= nums.length
3. 1 <= nums[i] <= 10^5

class Solution:

def canDivideIntoSubsequences(self, nums: List[int], K: int) -> bool:

count = {}

largest = 0

for x in nums:

if x in count:

count[x] += 1

if count[x] > largest: largest = count[x]

else: count[x] = 1

if largest \* K > len(nums): return False

return True

**1088. Confusing Number II**

Hard

175FavoriteShare

We can rotate digits by 180 degrees to form new digits. When 0, 1, 6, 8, 9 are rotated 180 degrees, they become 0, 1, 9, 8, 6 respectively. When 2, 3, 4, 5 and 7 are rotated 180 degrees, they become invalid.

A *confusing number* is a number that when rotated 180 degrees becomes a **different** number with each digit valid.(Note that the rotated number can be greater than the original number.)

Given a positive integer N, return the number of confusing numbers between 1 and N inclusive.

**Example 1:**

**Input:** 20

**Output:** 6

**Explanation:**

The confusing numbers are [6,9,10,16,18,19].

6 converts to 9.

9 converts to 6.

10 converts to 01 which is just 1.

16 converts to 91.

18 converts to 81.

19 converts to 61.

**Example 2:**

**Input:** 100

**Output:** 19

**Explanation:**

The confusing numbers are [6,9,10,16,18,19,60,61,66,68,80,81,86,89,90,91,98,99,100].

**Note:**

1. 1 <= N <= 10^9

class Solution:

def confusingNumberII(self, N: int) -> int:

if N < 6: return 0

if N < 9: return 1

if N < 10: return 2

N = str(N)

n = len(N)

# find all valid number with length < n

result = 2

for i in range(2,n):

result += 4\*(5\*\*(i-1))

if i % 2 == 1:

result -= 3\*4\*(5\*\*(i//2-1))

else:

result -= 4\*(5\*\*(i//2-1))

'''def get\_similar\_num(i):

# get the number of similar numbers with digit i

# can start with 0

if i == 1: return 5

elif i % 2 == 1:

return -= 3\*4\*(5\*\*(i//2-1))

else:

result -= 4\*(5\*\*(i//2-1))'''

# find all number with length = n, including strobographic number

print(result)

less = [1,2,2,2,2,2,3,3,4,5]

valid = set([0,1,6,8,9])

original\_good = True

for i in range(n):

temp = 1

x = int(N[i])

if x == 0: continue

if i == 0: temp \*= less[x-1]-1

else: temp \*= less[x-1]

temp \*= 5\*\*(n-i-1)

result += temp

if x not in valid:

original\_good = False

break

if original\_good: result += 1

'''for key,val in enumerate(N):

if key == 0: temp \*= less[int(val)]-1

else: temp \*= less[int(val)]

print(temp)

result += temp'''

# subtract the number of strobographic number with length = n

dic = {'6': '9', '9': '6', '8': '8', '0': '0', '1': '1'}

buf = collections.deque()

def is\_valid(buf):

x = "".join(buf)

return x <= N

def make\_strobo(buf,length):

result = 0

if len(buf) == length:

if buf[0] != '0' and is\_valid(buf):

result += 1

return result

elif len(buf) == length-1:

for x in ('0','1','8'):

buf.insert(len(buf)//2,x)

if buf[0] != '0' and is\_valid(buf):

result += 1

del buf[len(buf)//2]

return result

else:

for x in dic:

buf.append(x)

buf.appendleft(dic[x])

result += make\_strobo(buf,length)

buf.pop()

buf.popleft()

# print(result)

return result

print(result)

result -= make\_strobo(buf,n)

return result

**1066. Campus Bikes II**

Medium

1365FavoriteShare

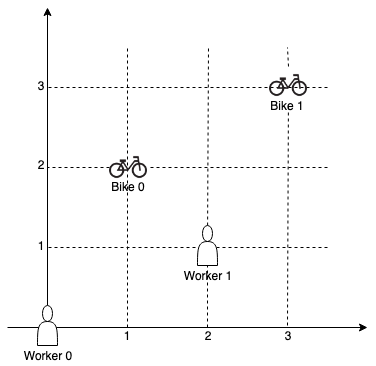
On a campus represented as a 2D grid, there are N workers and M bikes, with N <= M. Each worker and bike is a 2D coordinate on this grid.

We assign one unique bike to each worker so that the sum of the Manhattan distances between each worker and their assigned bike is minimized.

The Manhattan distance between two points p1 and p2 is Manhattan(p1, p2) = |p1.x - p2.x| + |p1.y - p2.y|.

Return the minimum possible sum of Manhattan distances between each worker and their assigned bike.

**Example 1:**



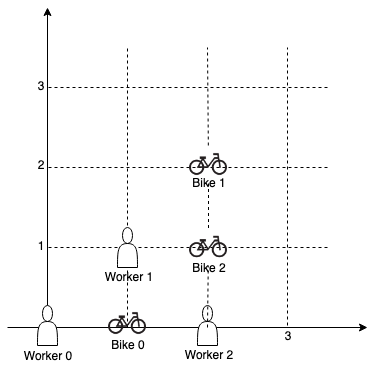
**Input:** workers = [[0,0],[2,1]], bikes = [[1,2],[3,3]]

**Output:** 6

**Explanation:**

We assign bike 0 to worker 0, bike 1 to worker 1. The Manhattan distance of both assignments is 3, so the output is 6.

**Example 2:**



**Input:** workers = [[0,0],[1,1],[2,0]], bikes = [[1,0],[2,2],[2,1]]

**Output:** 4

**Explanation:**

We first assign bike 0 to worker 0, then assign bike 1 to worker 1 or worker 2, bike 2 to worker 2 or worker 1. Both assignments lead to sum of the Manhattan distances as 4.

**Note:**

1. 0 <= workers[i][0], workers[i][1], bikes[i][0], bikes[i][1] < 1000
2. All worker and bike locations are distinct.
3. 1 <= workers.length <= bikes.length <= 10

class Solution {

public:

int assignBikes(vector<vector<int>>& workers, vector<vector<int>>& bikes) {

int n = bikes.size();

vector<vector<int>> dp(workers.size()+1,vector<int>(1<<n,INT\_MAX));

dp[0][0] = 0;

for (int w = 0; w< workers.size();w++){

for (int bike\_mask = 0; bike\_mask < 1<<n; bike\_mask ++){

for (int b = 0; b < bikes.size();b++){

if (bike\_mask & (1<<b)) continue;

if (dp[w][bike\_mask] == INT\_MAX) continue;

int new\_bike\_mask = bike\_mask | (1<<b);

dp[w+1][new\_bike\_mask] = min(dp[w+1][new\_bike\_mask], dp[w][bike\_mask] + abs(workers[w][0] - bikes[b][0])+abs(workers[w][1] - bikes[b][1]));

}

}

}

// for (int i = 0; i < workers.size()+1;i++){

// for (int j = 0; j < 1<<n;j++){

// cout<<dp[i][j]<<",";

// }

// cout<<endl;

// }

return \*min\_element(dp[workers.size()].begin(), dp[workers.size()].end());

}

};

**1057. Campus Bikes**

Medium

12822FavoriteShare

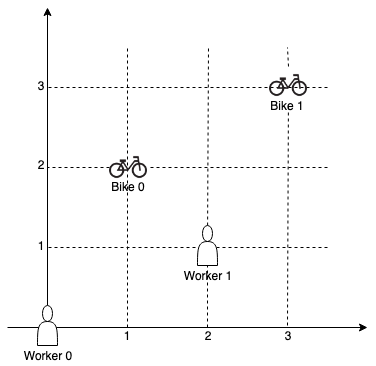
On a campus represented as a 2D grid, there are N workers and M bikes, with N <= M. Each worker and bike is a 2D coordinate on this grid.

Our goal is to assign a bike to each worker. Among the available bikes and workers, we choose the (worker, bike) pair with the shortest Manhattan distance between each other, and assign the bike to that worker. (If there are multiple (worker, bike) pairs with the same shortest Manhattan distance, we choose the pair with the smallest worker index; if there are multiple ways to do that, we choose the pair with the smallest bike index). We repeat this process until there are no available workers.

The Manhattan distance between two points p1 and p2 is Manhattan(p1, p2) = |p1.x - p2.x| + |p1.y - p2.y|.

Return a vector ans of length N, where ans[i] is the index (0-indexed) of the bike that the i-th worker is assigned to.

**Example 1:**



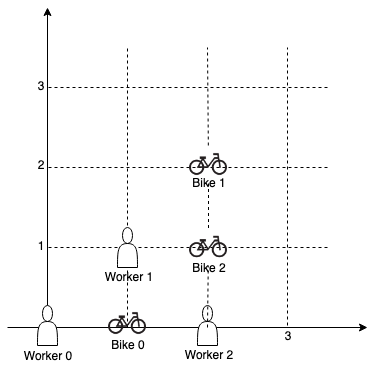
**Input:** workers = [[0,0],[2,1]], bikes = [[1,2],[3,3]]

**Output:** [1,0]

**Explanation:**

Worker 1 grabs Bike 0 as they are closest (without ties), and Worker 0 is assigned Bike 1. So the output is [1, 0].

**Example 2:**



**Input:** workers = [[0,0],[1,1],[2,0]], bikes = [[1,0],[2,2],[2,1]]

**Output:** [0,2,1]

**Explanation:**

Worker 0 grabs Bike 0 at first. Worker 1 and Worker 2 share the same distance to Bike 2, thus Worker 1 is assigned to Bike 2, and Worker 2 will take Bike 1. So the output is [0,2,1].

**Note:**

1. 0 <= workers[i][j], bikes[i][j] < 1000
2. All worker and bike locations are distinct.
3. 1 <= workers.length <= bikes.length <= 1000

class Solution:

def assignBikes(self, workers: List[List[int]], bikes: List[List[int]]) -> List[int]:

def dis(x,y):

return abs(x[0]-y[0])+abs(x[1]-y[1])

'''result = [-1] \* len(workers)

used = [0] \* len(bikes)

for w in range(len(workers)):

minimum = 2\*\*31

min\_index = -1

for b in range(len(bikes)):

if used[b]: continue

ass = dis(workers[w],bikes[b])

if ass < minimum:

minimum = ass

min\_index = b

result[w] = min\_index

used[min\_index] = 1

return result'''

data = []

for w in range(len(workers)):

for b in range(len(bikes)):

data.append([dis(workers[w],bikes[b]),w,b])

data.sort()

used\_worker = [0]\*len(workers)

used\_bike = [0]\*len(bikes)

result = [-1]\*len(workers)

for x,y,z in data:

if used\_worker[y] == 0 and used\_bike[z] == 0:

result[y]=z

used\_worker[y] = 1

used\_bike[z] = 1

return result

**1055. Shortest Way to Form String**

Medium

814FavoriteShare

From any string, we can form a *subsequence* of that string by deleting some number of characters (possibly no deletions).

Given two strings source and target, return the minimum number of subsequences of source such that their concatenation equals target. If the task is impossible, return -1.

**Example 1:**

**Input:** source = "abc", target = "abcbc"

**Output:** 2

**Explanation:** The target "abcbc" can be formed by "abc" and "bc", which are subsequences of source "abc".

**Example 2:**

**Input:** source = "abc", target = "acdbc"

**Output:** -1

**Explanation:** The target string cannot be constructed from the subsequences of source string due to the character "d" in target string.

**Example 3:**

**Input:** source = "xyz", target = "xzyxz"

**Output:** 3

**Explanation:** The target string can be constructed as follows "xz" + "y" + "xz".

**Constraints:**

* Both the source and target strings consist of only lowercase English letters from "a"-"z".
* The lengths of source and target string are between 1 and 1000.

class Solution:

def shortestWay(self, source: str, target: str) -> int:

#print(set(list(target)),set(list(source)),set(list(target)) in set(list(source)))

if not (set(list(target)).issubset(set(list(source)))): return -1

count = 1

limit = len(source) -1

i = -1

for x in target:

i += 1

if i > limit:

count += 1

i = 0

while source[i] != x:

i += 1

if i > limit:

count += 1

i = 0

return count

**774. Minimize Max Distance to Gas Station**

Hard

24836FavoriteShare

On a horizontal number line, we have gas stations at positions stations[0], stations[1], ..., stations[N-1], where N = stations.length.

Now, we add K more gas stations so that **D**, the maximum distance between adjacent gas stations, is minimized.

Return the smallest possible value of **D**.

**Example:**

**Input:** stations = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], K = 9

**Output:** 0.500000

**Note:**

1. stations.length will be an integer in range [10, 2000].
2. stations[i] will be an integer in range [0, 10^8].
3. K will be an integer in range [1, 10^6].
4. Answers within 10^-6 of the true value will be accepted as correct.

class Solution:

def minmaxGasDist(self, stations: List[int], K: int) -> float:

'''

distance = []

for i in range(1,len(stations)):

distance.append([-(stations[i]-stations[i-1]),1])

heap = distance.copy()

heapq.heapify(heap)

for i in range(K):

head = heapq.heappop(heap)

head[0] = head[0]\*head[1]/(head[1]+1)

head[1] += 1

heapq.heappush(heap, head)

head = heapq.heappop(heap)

return -head[0]

'''

distance = []

for i in range(1,len(stations)):

distance.append(stations[i]-stations[i-1])

l = 0.0

r = float(max(distance))

while (r-l) > (10\*\*-6):

#print(l,r)

mid = (l+r)/2

cnt = 0

result = True

for x in distance:

if x > mid:

cnt += math.ceil(x/mid)-1

if cnt > K:

result = False

break

if result: r = mid

else: l = mid

return (l+r)/2

**772. Basic Calculator III**

Hard

26584FavoriteShare

Implement a basic calculator to evaluate a simple expression string.

The expression string may contain open ( and closing parentheses ), the plus + or minus sign -, **non-negative** integers and empty spaces .

The expression string contains only non-negative integers, +, -, \*, / operators , open ( and closing parentheses ) and empty spaces . The integer division should truncate toward zero.

You may assume that the given expression is always valid. All intermediate results will be in the range of [-2147483648, 2147483647].

Some examples:

"1 + 1" = 2

" 6-4 / 2 " = 4

"2\*(5+5\*2)/3+(6/2+8)" = 21

"(2+6\* 3+5- (3\*14/7+2)\*5)+3"=-12

**Note:** **Do not** use the eval built-in library function.

class Solution:

def calculate(self, s: str) -> int:

def solve\_simple(s):

'''return 1

stack = [""]

for i in range(len(s)):

if s[i] in "+-\*/":

if len(stack) > 1:

if stack[-2][-1] == "\*":

temp = int(stack.pop())

stack[-1] = str(int(stack[-1][0:len(stack[-1])-1]) \* int(temp))

elif stack[-2][-1] == "/":

temp = int(stack.pop())

stack[-1] = str(int(stack[-1][0:len(stack[-1])-1]) // int(temp))

stack[-1] += s[i]

stack.append("")

elif i == len(s)-1:

stack[-1] += s[i]

print(stack[-2][-1])

if len(stack) > 1:

if stack[-2][-1] == "\*":

temp = int(stack.pop())

stack[-1] = str(int(stack[-1][0:len(stack[-1])-1]) \* int(temp))

elif stack[-2][-1] == "/":

temp = int(stack.pop())

stack[-1] = str(int(stack[-1][0:len(stack[-1])-1]) // int(temp))

else:

stack[-1] += s[i]

print(stack)

while len(stack) > 1:

temp = stack.pop()

if stack[-1][-1] == "+":

stack[-1] = str(int(stack[-1][0:len(stack[-1])-1])+int(temp))

elif stack[-1][-1] == "-":

stack[-1] = str(int(stack[-1][0:len(stack[-1])-1])-int(temp))

return stack[0]'''

i = 0

while 1:

if i >= len(s)-1: break

if s[i] == "\*":

s[i-1] \*= int(s[i+1])

del s[i]

del s[i]

elif s[i] == "/":

try:

s[i-1] = s[i-1]//int(s[i+1])

except:

print(s[i-1],s[i+1])

del s[i]

del s[i]

else: i += 1

#print(s)

i = 0

while 1:

if i >= len(s)-1: break

if s[i] == "+":

s[i-1] += int(s[i+1])

del s[i]

del s[i]

elif s[i] == "-":

s[i-1] -= int(s[i+1])

del s[i]

del s[i]

else: i += 1

#print(s)

return s[0]

ass = s.split(" ")

s = "".join(ass)

temp = []

i = 0

while 1:

if i >= len(s):break

if s[i] == "-":

if i ==0 or (isinstance(s[i-1],str) and s[i-1] in "+-\*/("):

cnt = 1

while s[i] == "-":

cnt \*= -1

i += 1

temp.append(cnt)

temp.append("\*")

continue

if s[i].isdigit():

if len(temp) >0 and isinstance(temp[-1],int): temp[-1] = 10\*temp[-1]+int(s[i])

else:temp.append(int(s[i]))

else: temp.append(s[i])

i += 1

s = temp

print(s)

stack = [[]]

for x in s:

if x == '(': stack.append([])

elif x == ")":

temp = stack.pop()

temp\_result = solve\_simple(temp)

stack[-1].append(temp\_result)

elif isinstance(x,str) and x in "+-\*/":

stack[-1][-1] = int(stack[-1][-1])

stack[-1].append(x)

else:

if len(stack[-1]) == 0: stack[-1].append(x)

elif isinstance(stack[-1][-1] ,str) and stack[-1][-1] in "+-\*/": stack[-1].append(x)

else: stack[-1][-1] += x

print(stack)

stack[-1][-1] = int(stack[-1][-1])

#print(stack)

return int(solve\_simple(stack[0]))

**759. Employee Free Time**

Hard

25018FavoriteShare

We are given a list schedule of employees, which represents the working time for each employee.

Each employee has a list of non-overlapping Intervals, and these intervals are in sorted order.

Return the list of finite intervals representing **common, positive-length free time** for *all* employees, also in sorted order.

**Example 1:**

**Input:** schedule = [[[1,2],[5,6]],[[1,3]],[[4,10]]]

**Output:** [[3,4]]

**Explanation:**

There are a total of three employees, and all common

free time intervals would be [-inf, 1], [3, 4], [10, inf].

We discard any intervals that contain inf as they aren't finite.

**Example 2:**

**Input:** schedule = [[[1,3],[6,7]],[[2,4]],[[2,5],[9,12]]]

**Output:** [[5,6],[7,9]]

(Even though we are representing Intervals in the form [x, y], the objects inside are Intervals, not lists or arrays. For example, schedule[0][0].start = 1, schedule[0][0].end = 2, and schedule[0][0][0] is not defined.)

Also, we wouldn't include intervals like [5, 5] in our answer, as they have zero length.

**Note:**

1. schedule and schedule[i] are lists with lengths in range [1, 50].
2. 0 <= schedule[i].start < schedule[i].end <= 10^8.

**NOTE:** input types have been changed on June 17, 2019. Please reset to default code definition to get new method signature.

"""

# Definition for an Interval.

class Interval:

def \_\_init\_\_(self, start, end):

self.start = start

self.end = end

"""

class Solution:

def employeeFreeTime(self, schedule: 'list<list<Interval>>') -> 'list<Interval>':

n = len(schedule)

if n == 1: return schedule[0]

event = []

ENTER = 1

LEAVE = 0

for y in schedule:

for x in y:

event.append((x.start,ENTER))

event.append((x.end,LEAVE))

event.sort()

#print(event)

result = []

prev = None

count = 0

for x in event:

if x[1] == ENTER:

count += 1

if count == 1 and prev != None:

if prev != x[0]:

result.append(Interval(prev,x[0]))

elif x[1] == LEAVE:

count -= 1

if count == 0: prev = x[0]

return result

**734. Sentence Similarity**

Easy

138225FavoriteShare

Given two sentences words1, words2 (each represented as an array of strings), and a list of similar word pairs pairs, determine if two sentences are similar.

For example, "great acting skills" and "fine drama talent" are similar, if the similar word pairs are pairs = [["great", "fine"], ["acting","drama"], ["skills","talent"]].

Note that the similarity relation is not transitive. For example, if "great" and "fine" are similar, and "fine" and "good" are similar, "great" and "good" are **not** necessarily similar.

However, similarity is symmetric. For example, "great" and "fine" being similar is the same as "fine" and "great" being similar.

Also, a word is always similar with itself. For example, the sentences words1 = ["great"], words2 = ["great"], pairs = [] are similar, even though there are no specified similar word pairs.

Finally, sentences can only be similar if they have the same number of words. So a sentence like words1 = ["great"] can never be similar to words2 = ["doubleplus","good"].

**Note:**

* The length of words1 and words2 will not exceed 1000.
* The length of pairs will not exceed 2000.
* The length of each pairs[i] will be 2.
* The length of each words[i] and pairs[i][j] will be in the range [1, 20].

class Solution:

def areSentencesSimilar(self, words1: List[str], words2: List[str], pairs: List[List[str]]) -> bool:

if len(words1) != len(words2): return False

dic = {}

for x,y in pairs:

if x not in dic: dic[x] = [y]

else: dic[x].append(y)

for i in range(len(words1)):

if words1[i] == words2[i]: continue

if words1[i] not in dic:

if words2[i] not in dic: return False

if words1[i] not in dic[words2[i]]: return False

continue

if words2[i] not in dic:

if words2[i] not in dic[words1[i]]: return False

continue

if (words2[i] not in dic[words1[i]]) and (words1[i] not in dic[words2[i]]):

return False

return True

**727. Minimum Window Subsequence**

Hard

35016FavoriteShare

Given strings S and T, find the minimum (contiguous) **substring** W of S, so that T is a **subsequence** of W.

If there is no such window in S that covers all characters in T, return the empty string "". If there are multiple such minimum-length windows, return the one with the left-most starting index.

**Example 1:**

**Input:**

S = "abcdebdde", T = "bde"

**Output:** "bcde"

**Explanation:**

"bcde" is the answer because it occurs before "bdde" which has the same length.

"deb" is not a smaller window because the elements of T in the window must occur in order.

**Note:**

* All the strings in the input will only contain lowercase letters.
* The length of S will be in the range [1, 20000].
* The length of T will be in the range [1, 100].

class Solution:

def minWindow(self, S: str, T: str) -> str:

n = len(S)

dp = [None]\*n

for key,val in enumerate(S):

if val == T[0]: dp[key] = key

for i in range(1,len(T)):

dp\_new = [None]\*n

start = None

for key,val in enumerate(S):

if start != None and val == T[i]:

dp\_new[key] = start

if dp[key] != None: start = dp[key]

dp = dp\_new

result = (0,2\*\*31)

for key,val in enumerate(dp):

if val != None:

if key-val < result[1] - result[0]:

result = (val,key)

if result == (0,2\*\*31): return ""

return S[result[0]:result[1]+1]

**708. Insert into a Cyclic Sorted List**

Medium

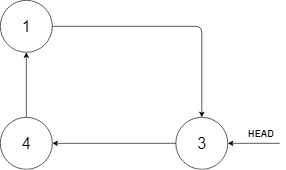
161186FavoriteShare

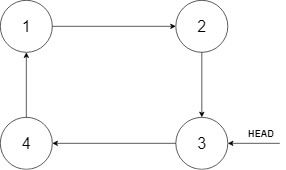
Given a node from a cyclic linked list which is sorted in ascending order, write a function to insert a value into the list such that it remains a cyclic sorted list. The given node can be a reference to *any* single node in the list, and may not be necessarily the smallest value in the cyclic list.

If there are multiple suitable places for insertion, you may choose any place to insert the new value. After the insertion, the cyclic list should remain sorted.

If the list is empty (i.e., given node is null), you should create a new single cyclic list and return the reference to that single node. Otherwise, you should return the original given node.

The following example may help you understand the problem better:

  
  
In the figure above, there is a cyclic sorted list of three elements. You are given a reference to the node with value 3, and we need to insert 2 into the list.

  
  
The new node should insert between node 1 and node 3. After the insertion, the list should look like this, and we should still return node 3.

"""

# Definition for a Node.

class Node:

def \_\_init\_\_(self, val, next):

self.val = val

self.next = next

"""

class Solution:

def insert(self, head: 'Node', insertVal: int) -> 'Node':

if head == None:

result = Node(insertVal,None)

result.next = result

return result

if head.next == head:

head.next = Node(insertVal,head)

return head

start = head

prev = head

curr = head.next

end = None

while 1:

if curr.val > insertVal and prev.val <= insertVal:

prev.next = Node(insertVal,curr)

return head

if curr.val < prev.val: end = prev

prev = curr

curr = curr.next

if curr == head.next: break

if end == None: end = head.next

end.next = Node(insertVal,end.next)

return head

**683. K Empty Slots**

Hard

490518FavoriteShare

You have N bulbs in a row numbered from 1 to N. Initially, all the bulbs are turned off. We turn on exactly one bulb everyday until all bulbs are on after N days.

You are given an array bulbs of length N where bulbs[i] = x means that on the (i+1)th day, we will turn on the bulb at position x where i is 0-indexed and x is 1-indexed.

Given an integer K, find out the **minimum day number** such that there exists two **turned on** bulbs that have **exactly** K bulbs between them that are **all turned off**.

If there isn't such day, return -1.

**Example 1:**

**Input:**

bulbs: [1,3,2]

K: 1

**Output:** 2

**Explanation:**

On the first day: bulbs[0] = 1, first bulb is turned on: [1,0,0]

On the second day: bulbs[1] = 3, third bulb is turned on: [1,0,1]

On the third day: bulbs[2] = 2, second bulb is turned on: [1,1,1]

We return 2 because on the second day, there were two on bulbs with one off bulb between them.

**Example 2:**

**Input:**

bulbs: [1,2,3]

K: 1

**Output:** -1

**Note:**

1. 1 <= N <= 20000
2. 1 <= bulbs[i] <= N
3. bulbs is a permutation of numbers from 1 to N.
4. 0 <= K <= 20000

class Solution:

def kEmptySlots(self, bulbs: List[int], K: int) -> int:

my\_Ass = [0]\*len(bulbs)

for key,val in enumerate(bulbs):

my\_Ass[val-1] = key+1

bulbs = my\_Ass

#print(bulbs)

l = 0

r = K+1

result = 2\*\*31

while r < len(bulbs):

great = True

for i in range(l+1,r):

if bulbs[i] < bulbs[l] or bulbs[i] < bulbs[r]:

l = i

r = l + K + 1

great = False

break

if great:

result = min(result,max(bulbs[l],bulbs[r]))

l += 1

r += 1

#print(l,r,result)

if result == 2\*\*31: result = -1

return result

**681. Next Closest Time**

Medium

369588FavoriteShare

Given a time represented in the format "HH:MM", form the next closest time by reusing the current digits. There is no limit on how many times a digit can be reused.

You may assume the given input string is always valid. For example, "01:34", "12:09" are all valid. "1:34", "12:9" are all invalid.

**Example 1:**

**Input:** "19:34"

**Output:** "19:39"

**Explanation:** The next closest time choosing from digits **1**, **9**, **3**, **4**, is **19:39**, which occurs 5 minutes later. It is not **19:33**, because this occurs 23 hours and 59 minutes later.

**Example 2:**

**Input:** "23:59"

**Output:** "22:22"

**Explanation:** The next closest time choosing from digits **2**, **3**, **5**, **9**, is **22:22**. It may be assumed that the returned time is next day's time since it is smaller than the input time numerically.

class Solution:

def nextClosestTime(self, time: str) -> str:

ass = time.split(":")

hour = ass[0]

mins = ass[1]

characters = list(hour)+list(mins)

integers = []

for x in characters: integers.append(int(x))

#print(integers)

digit\_min = min(integers)

for i in range(integers[3]+1,10):

if i in integers:

return hour+":"+mins[0]+str(i)

for i in range(integers[2]+1,6):

if i in integers:

return hour+":"+str(i)+str(digit\_min)

if integers[0] < 2:

for i in range(integers[1]+1,10):

if i in integers:

return hour[0]+str(i)+":"+str(digit\_min)+str(digit\_min)

for i in range(integers[0]+1,3):

if i in integers: return str(i)+str(digit\_min)+":"+str(digit\_min)+str(digit\_min)

else:

for i in range(integers[1]+1,4):

if i in integers:

return "2"+str(i)+":"+str(digit\_min)+str(digit\_min)

return str(digit\_min)+str(digit\_min)+":"+str(digit\_min)+str(digit\_min)

**656. Coin Path**

Hard

12771FavoriteShare

Given an array A (index starts at 1) consisting of N integers: A1, A2, ..., AN and an integer B. The integer B denotes that from any place (suppose the index is i) in the array A, you can jump to any one of the place in the array A indexed i+1, i+2, …, i+B if this place can be jumped to. Also, if you step on the index i, you have to pay Ai coins. If Ai is -1, it means you can’t jump to the place indexed i in the array.

Now, you start from the place indexed 1 in the array A, and your aim is to reach the place indexed Nusing the minimum coins. You need to return the path of indexes (starting from 1 to N) in the array you should take to get to the place indexed N using minimum coins.

If there are multiple paths with the same cost, return the lexicographically smallest such path.

If it's not possible to reach the place indexed N then you need to return an empty array.

**Example 1:**

**Input:** [1,2,4,-1,2], 2

**Output:** [1,3,5]

**Example 2:**

**Input:** [1,2,4,-1,2], 1

**Output:** []

**Note:**

1. Path Pa1, Pa2, ..., Pan is lexicographically smaller than Pb1, Pb2, ..., Pbm, if and only if at the first iwhere Pai and Pbi differ, Pai < Pbi; when no such i exists, then n < m.
2. A1 >= 0. A2, ..., AN (if exist) will in the range of [-1, 100].
3. Length of A is in the range of [1, 1000].
4. B is in the range of [1, 100].

class Solution:

def cheapestJump(self, A: List[int], B: int) -> List[int]:

dic = {}

def dfs(A,B,dic,current):

#print(current)

if current in dic: return dic[current]

if current >= len(A):

return 2\*\*31

if A[current] == -1:

return 2\*\*31

if current == len(A)-1:

dic[current] = A[-1]

return A[-1]

result\_cnt = 2\*\*31

for i in range(B):

temp1 = dfs(A,B,dic,current+i+1)

if temp1 < result\_cnt:

result\_cnt = temp1

if result\_cnt != 2\*\*31:

result\_cnt += A[current]

dic[current] = result\_cnt

return result\_cnt

temp1= dfs(A,B,dic,0)

print(dic)

if temp1 == 2\*\*31: return []

result = []

i = 0

while i < len(A):

result.append(i+1)

if i == len(A)-1: break

next\_val = 2\*\*31

next\_i = 0

for j in range(B):

if i+j+1 >= len(A):continue

if A[i+j+1] == -1: continue

if dic[i+j+1] < next\_val:

next\_val = dic[i+j+1]

next\_i = i+j+1

i = next\_i

return result

**644. Maximum Average Subarray II**

Hard

29932FavoriteShare

Given an array consisting of n integers, find the contiguous subarray whose **length is greater than or equal to** k that has the maximum average value. And you need to output the maximum average value.

**Example 1:**

**Input:** [1,12,-5,-6,50,3], k = 4

**Output:** 12.75

**Explanation:**

when length is 5, maximum average value is 10.8,

when length is 6, maximum average value is 9.16667.

Thus return 12.75.

**Note:**

1. 1 <= k <= n <= 10,000.
2. Elements of the given array will be in range [-10,000, 10,000].
3. The answer with the calculation error less than 10-5 will be accepted.

class Solution:

def findMaxAverage(self, nums: List[int], k: int) -> float:

def is\_good(nums,avg,k):

right = 0

for i in range(k):

right += nums[i] - avg

if right >= 0: return True

best\_left = 0

left = 0

for i in range(k,len(nums)):

right += nums[i] - avg

left += nums[i-k] - avg

best\_left = min(best\_left,left)

if right - best\_left >= 0: return True

return False

l = min(nums)

r = max(nums)

while r - l > 10\*\*-5:

mid = (r+l)/2

if is\_good(nums,mid,k): l = mid

else: r = mid

return l

**642. Design Search Autocomplete System**

Hard

55246FavoriteShare

Design a search autocomplete system for a search engine. Users may input a sentence (at least one word and end with a special character '#'). For **each character** they type **except '#'**, you need to return the **top 3** historical hot sentences that have prefix the same as the part of sentence already typed. Here are the specific rules:

1. The hot degree for a sentence is defined as the number of times a user typed the exactly same sentence before.
2. The returned top 3 hot sentences should be sorted by hot degree (The first is the hottest one). If several sentences have the same degree of hot, you need to use ASCII-code order (smaller one appears first).
3. If less than 3 hot sentences exist, then just return as many as you can.
4. When the input is a special character, it means the sentence ends, and in this case, you need to return an empty list.

Your job is to implement the following functions:

The constructor function:

AutocompleteSystem(String[] sentences, int[] times): This is the constructor. The input is **historical data**. Sentences is a string array consists of previously typed sentences. Times is the corresponding times a sentence has been typed. Your system should record these historical data.

Now, the user wants to input a new sentence. The following function will provide the next character the user types:

List<String> input(char c): The input c is the next character typed by the user. The character will only be lower-case letters ('a' to 'z'), blank space (' ') or a special character ('#'). Also, the previously typed sentence should be recorded in your system. The output will be the **top 3** historical hot sentences that have prefix the same as the part of sentence already typed.

**Example:**  
**Operation:** AutocompleteSystem(["i love you", "island","ironman", "i love leetcode"], [5,3,2,2])  
The system have already tracked down the following sentences and their corresponding times:  
"i love you" : 5 times  
"island" : 3 times  
"ironman" : 2 times  
"i love leetcode" : 2 times  
Now, the user begins another search:  
  
**Operation:** input('i')  
**Output:** ["i love you", "island","i love leetcode"]  
**Explanation:**  
There are four sentences that have prefix "i". Among them, "ironman" and "i love leetcode" have same hot degree. Since ' ' has ASCII code 32 and 'r' has ASCII code 114, "i love leetcode" should be in front of "ironman". Also we only need to output top 3 hot sentences, so "ironman" will be ignored.  
  
**Operation:** input(' ')  
**Output:** ["i love you","i love leetcode"]  
**Explanation:**  
There are only two sentences that have prefix "i ".  
  
**Operation:** input('a')  
**Output:** []  
**Explanation:**  
There are no sentences that have prefix "i a".  
  
**Operation:** input('#')  
**Output:** []  
**Explanation:**  
The user finished the input, the sentence "i a" should be saved as a historical sentence in system. And the following input will be counted as a new search.

**Note:**

1. The input sentence will always start with a letter and end with '#', and only one blank space will exist between two words.
2. The number of **complete sentences** that to be searched won't exceed 100. The length of each sentence including those in the historical data won't exceed 100.
3. Please use double-quote instead of single-quote when you write test cases even for a character input.
4. Please remember to **RESET** your class variables declared in class AutocompleteSystem, as static/class variables are **persisted across multiple test cases**. Please see [here](https://leetcode.com/faq/#different-output) for more details.

class Node:

def \_\_init\_\_(self,val):

self.val = val

self.baby = [None]\*27

class AutocompleteSystem:

def to\_int(self,char):

if char == " ": return 0

else:

char = char.lower()

return ord(char) - ord("a")+1

def to\_char(self,num):

if num == 0: return " "

else: return chr(ord('a') + num - 1)

def get\_all\_sentence(self,root):

val = []

result = []

if root.val != 0:

result.append("")

val.append(root.val)

for i in range(len(root.baby)):

x = root.baby[i]

if x != None:

temp,hot = self.get\_all\_sentence(x)

if len(temp) == 0:

result.append(self.to\_char(i))

else:

for y in temp:

result.append(self.to\_char(i)+y)

val.extend(hot)

return result,val

def \_\_init\_\_(self, sentences: List[str], times: List[int]):

self.trie = Node(0)

self.cur = self.trie

self.history = ""

for i in range(len(sentences)):

curr\_node = self.trie

for j in range(len(sentences[i])):

branch = self.to\_int(sentences[i][j])

if curr\_node.baby[branch] == None:

curr\_node.baby[branch] = Node(0)

curr\_node = curr\_node.baby[branch]

if j == len(sentences[i]) - 1:

curr\_node.val = times[i]

break

def input(self, c: str) -> List[str]:

if c == "#":

self.cur.val += 1

self.cur = self.trie

self.history = ""

return []

branch = self.to\_int(c)

self.history += c

if self.cur.baby[branch] == None:

self.cur.baby[branch] = Node(0)

self.cur = self.cur.baby[branch]

return []

self.cur = self.cur.baby[branch]

result = []

word,val = self.get\_all\_sentence(self.cur)

if len(word) < 2: result = word

else:

poop = []

for x,y in zip(word,val):

poop.append([x,y])

poop.sort(key = lambda x:(-x[1],x[0]))

poop

for i in range(min(len(poop),3)):

result.append(poop[i][0])

final\_fuck = []

if len(result) == 0:final\_fuck = [self.history]

else:

for x in result:final\_fuck.append(self.history+x)

return final\_fuck

# Your AutocompleteSystem object will be instantiated and called as such:

# obj = AutocompleteSystem(sentences, times)

# param\_1 = obj.input(c)

**616. Add Bold Tag in String**

Medium

35538FavoriteShare

Given a string **s** and a list of strings **dict**, you need to add a closed pair of bold tag <b> and </b> to wrap the substrings in s that exist in dict. If two such substrings overlap, you need to wrap them together by only one pair of closed bold tag. Also, if two substrings wrapped by bold tags are consecutive, you need to combine them.

**Example 1:**

**Input:**

s = "abcxyz123"

dict = ["abc","123"]

**Output:**

"<b>abc</b>xyz<b>123</b>"

**Example 2:**

**Input:**

s = "aaabbcc"

dict = ["aaa","aab","bc"]

**Output:**

"<b>aaabbc</b>c"

**Note:**

1. The given dict won't contain duplicates, and its length won't exceed 100.
2. All the strings in input have length in range [1, 1000].

class Solution:

def addBoldTag(self, s: str, dict: List[str]) -> str:

bold = [0]\*len(s)

for word in dict:

for i in range(len(s)-len(word)+1):

if s[i:i+len(word)] == word:

for j in range(i,i+len(word)):

bold[j] = 1

not\_bold = True

result = ""

for i in range(len(s)):

if not\_bold:

if bold[i]:

result += "<b>"+s[i]

not\_bold = False

else:

result += s[i]

else:

if bold[i]: result += s[i]

else:

result += "</b>"+s[i]

not\_bold = True

if not\_bold == False: result += "</b>"

return result

**68. Maximum Vacation Days**

Hard

23432FavoriteShare

LeetCode wants to give one of its best employees the option to travel among **N** cities to collect algorithm problems. But all work and no play makes Jack a dull boy, you could take vacations in some particular cities and weeks. Your job is to schedule the traveling to maximize the number of vacation days you could take, but there are certain rules and restrictions you need to follow.

**Rules and restrictions:**

1. You can only travel among **N** cities, represented by indexes from 0 to N-1. Initially, you are in the city indexed 0 on **Monday**.
2. The cities are connected by flights. The flights are represented as a **N\*N** matrix (not necessary symmetrical), called **flights** representing the airline status from the city i to the city j. If there is no flight from the city i to the city j, **flights[i][j] = 0**; Otherwise, **flights[i][j] = 1**. Also, **flights[i][i] = 0**for all i.
3. You totally have **K** weeks (**each week has 7 days**) to travel. You can only take flights at most once **per day** and can only take flights on each week's **Monday** morning. Since flight time is so short, we don't consider the impact of flight time.
4. For each city, you can only have restricted vacation days in different weeks, given an **N\*K** matrix called **days** representing this relationship. For the value of **days[i][j]**, it represents the maximum days you could take vacation in the city **i** in the week **j**.

You're given the **flights** matrix and **days** matrix, and you need to output the maximum vacation days you could take during **K** weeks.

**Example 1:**

**Input:**flights = [[0,1,1],[1,0,1],[1,1,0]], days = [[1,3,1],[6,0,3],[3,3,3]]

**Output:** 12

**Explanation:**   
Ans = 6 + 3 + 3 = 12.

One of the best strategies is:

1st week : fly from city 0 to city 1 on Monday, and play 6 days and work 1 day.   
(Although you start at city 0, we could also fly to and start at other cities since it is Monday.)

2nd week : fly from city 1 to city 2 on Monday, and play 3 days and work 4 days.

3rd week : stay at city 2, and play 3 days and work 4 days.

**Example 2:**

**Input:**flights = [[0,0,0],[0,0,0],[0,0,0]], days = [[1,1,1],[7,7,7],[7,7,7]]

**Output:** 3

**Explanation:**   
Ans = 1 + 1 + 1 = 3.

Since there is no flights enable you to move to another city, you have to stay at city 0 for the whole 3 weeks.   
For each week, you only have one day to play and six days to work.   
So the maximum number of vacation days is 3.

**Example 3:**

**Input:**flights = [[0,1,1],[1,0,1],[1,1,0]], days = [[7,0,0],[0,7,0],[0,0,7]]

**Output:** 21

**Explanation:**  
Ans = 7 + 7 + 7 = 21

One of the best strategies is:

1st week : stay at city 0, and play 7 days.

2nd week : fly from city 0 to city 1 on Monday, and play 7 days.

3rd week : fly from city 1 to city 2 on Monday, and play 7 days.

**Note:**

1. **N and K** are positive integers, which are in the range of [1, 100].
2. In the matrix **flights**, all the values are integers in the range of [0, 1].
3. In the matrix **days**, all the values are integers in the range [0, 7].
4. You could stay at a city beyond the number of vacation days, but you should **work** on the extra days, which won't be counted as vacation days.
5. If you fly from the city A to the city B and take the vacation on that day, the deduction towards vacation days will count towards the vacation days of city B in that week.
6. We don't consider the impact of flight hours towards the calculation of vacation days.

class Solution:

def maxVacationDays(self, flights: List[List[int]], days: List[List[int]]) -> int:

city = len(flights)

k = len(days[0])

dp = [-2\*\*31]\*city

dp[0] = 0

for j in range(k):

new\_dp = [-2\*\*31]\*city

for i in range(city):

vacation = days[i][j]

best = -2\*\*31

for old in range(city):

if old == i or flights[old][i]: best = max(best,dp[old])

new\_dp[i] = best + vacation

dp = new\_dp

#print(dp)

return max(dp)

**544. Output Contest Matches**

Medium

21766FavoriteShare

During the NBA playoffs, we always arrange the rather strong team to play with the rather weak team, like make the rank 1 team play with the rank nth team, which is a good strategy to make the contest more interesting. Now, you're given **n** teams, you need to output their **final** contest matches in the form of a string.

The **n** teams are given in the form of positive integers from 1 to n, which represents their initial rank. (Rank 1 is the strongest team and Rank n is the weakest team.) We'll use parentheses('(', ')') and commas(',') to represent the contest team pairing - parentheses('(' , ')') for pairing and commas(',') for partition. During the pairing process in each round, you always need to follow the strategy of making the rather strong one pair with the rather weak one.

**Example 1:**

**Input:** 2

**Output:** (1,2)

**Explanation:**

Initially, we have the team 1 and the team 2, placed like: 1,2.

Then we pair the team (1,2) together with '(', ')' and ',', which is the final answer.

**Example 2:**

**Input:** 4

**Output:** ((1,4),(2,3))

**Explanation:**

In the first round, we pair the team 1 and 4, the team 2 and 3 together, as we need to make the strong team and weak team together.

And we got (1,4),(2,3).

In the second round, the winners of (1,4) and (2,3) need to play again to generate the final winner, so you need to add the paratheses outside them.

And we got the final answer ((1,4),(2,3)).

**Example 3:**

**Input:** 8

**Output:** (((1,8),(4,5)),((2,7),(3,6)))

**Explanation:**

First round: (1,8),(2,7),(3,6),(4,5)

Second round: ((1,8),(4,5)),((2,7),(3,6))

Third round: (((1,8),(4,5)),((2,7),(3,6)))

Since the third round will generate the final winner, you need to output the answer (((1,8),(4,5)),((2,7),(3,6))).

**Note:**

1. The **n** is in range [2, 212].
2. We ensure that the input **n** can be converted into the form 2k, where k is a positive integer.

class Solution:

def get\_list(self,current,n):

result = []

if current == n:

for i in range(n//2):

result.append("("+str(i+1)+","+str(n-i)+")")

return result

else:

temp = self.get\_list(current\*2,n)

for i in range(len(temp)//2):

#print(i+1,len(temp)-i)

result.append("("+temp[i]+","+temp[len(temp)-i-1]+")")

return result

def findContestMatch(self, n: int) -> str:

result = self.get\_list(2,n)

return result[0]

**527. Word Abbreviation**

Hard

16199FavoriteShare

Given an array of n distinct non-empty strings, you need to generate **minimal** possible abbreviations for every word following rules below.

1. Begin with the first character and then the number of characters abbreviated, which followed by the last character.
2. If there are any conflict, that is more than one words share the same abbreviation, a longer prefix is used instead of only the first character until making the map from word to abbreviation become unique. In other words, a final abbreviation cannot map to more than one original words.
3. If the abbreviation doesn't make the word shorter, then keep it as original.

**Example:**

**Input:** ["like", "god", "internal", "me", "internet", "interval", "intension", "face", "intrusion"]

**Output:** ["l2e","god","internal","me","i6t","interval","inte4n","f2e","intr4n"]

**Note:**

1. Both n and the length of each word will not exceed 400.
2. The length of each word is greater than 1.
3. The words consist of lowercase English letters only.
4. The return answers should be **in the same order** as the original array.

class Trie:

def \_\_init\_\_ (self,val):

self.val = val

self.next = {}

class Solution:

def wordsAbbreviation(self, dict: List[str]) -> List[str]:

forest = {}

for word in dict:

if len(word)<4: continue

encode = word[0]+str(len(word))+word[-1]

if encode not in forest: forest[encode] = Trie(0)

current = forest[encode]

for x in word:

if x not in current.next:

current.next[x] = Trie(1)

else:

current.next[x].val += 1

current = current.next[x]

result = []

for word in dict:

if len(word)<4:

result.append(word)

continue

encode = word[0]+str(len(word))+word[-1]

current = forest[encode]

for i,x in enumerate(word):

if i >= len(word)-3:

result.append(word)

break

current = current.next[x]

if current.val == 1:

temp = word[:i+1]+str(len(word)-2-i)+word[-1]

result.append(temp)

break

return result

**499. The Maze III**

Hard

12236FavoriteShare

There is a **ball** in a maze with empty spaces and walls. The ball can go through empty spaces by rolling **up**(u), **down** (d), **left** (l) or **right** (r), but it won't stop rolling until hitting a wall. When the ball stops, it could choose the next direction. There is also a **hole** in this maze. The ball will drop into the hole if it rolls on to the hole.

Given the **ball position**, the **hole position** and the **maze**, find out how the ball could drop into the hole by moving the **shortest distance**. The distance is defined by the number of **empty spaces** traveled by the ball from the start position (excluded) to the hole (included). Output the moving **directions** by using 'u', 'd', 'l' and 'r'. Since there could be several different shortest ways, you should output the **lexicographically smallest** way. If the ball cannot reach the hole, output "impossible".

The maze is represented by a binary 2D array. 1 means the wall and 0 means the empty space. You may assume that the borders of the maze are all walls. The ball and the hole coordinates are represented by row and column indexes.

**Example 1:**

**Input 1:** a maze represented by a 2D array

0 0 0 0 0

1 1 0 0 1

0 0 0 0 0

0 1 0 0 1

0 1 0 0 0

**Input 2:** ball coordinate (rowBall, colBall) = (4, 3)

**Input 3:** hole coordinate (rowHole, colHole) = (0, 1)

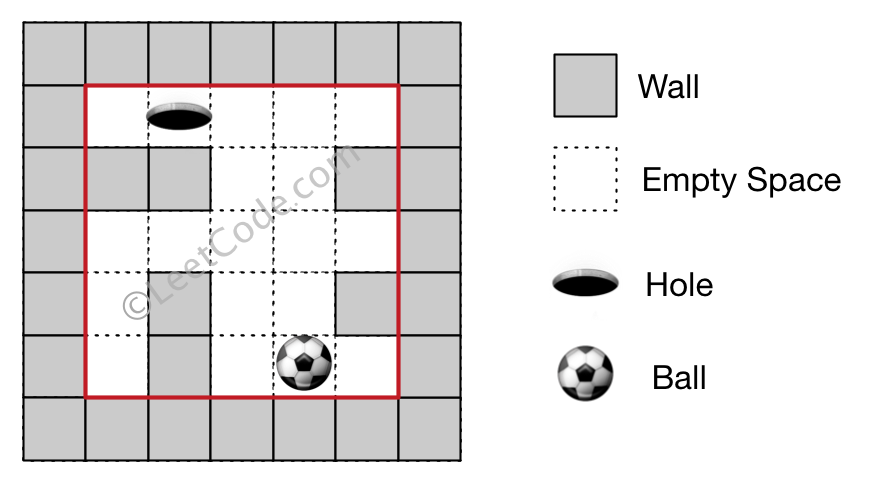
**Output:** "lul"

**Explanation:** There are two shortest ways for the ball to drop into the hole.

The first way is left -> up -> left, represented by "lul".

The second way is up -> left, represented by 'ul'.

Both ways have shortest distance 6, but the first way is lexicographically smaller because 'l' < 'u'. So the output is "lul".



**Example 2:**

**Input 1:** a maze represented by a 2D array

0 0 0 0 0

1 1 0 0 1

0 0 0 0 0

0 1 0 0 1

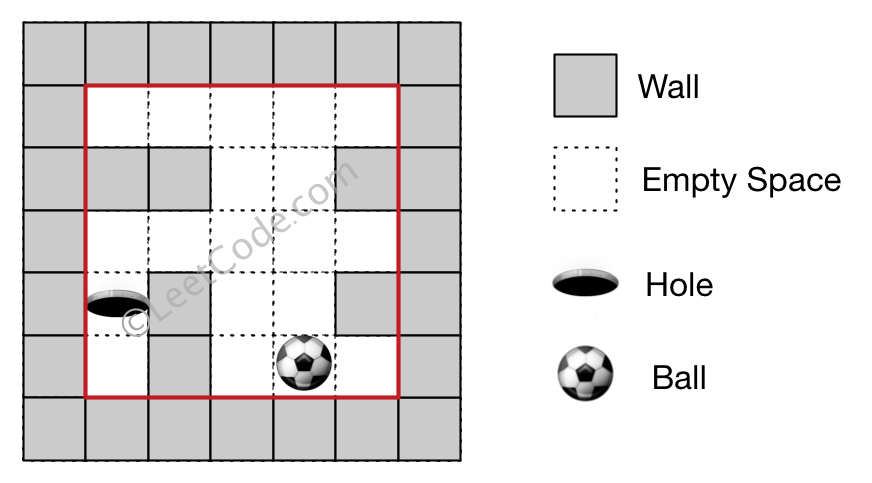
0 1 0 0 0

**Input 2:** ball coordinate (rowBall, colBall) = (4, 3)

**Input 3:** hole coordinate (rowHole, colHole) = (3, 0)

**Output:** "impossible"

**Explanation:** The ball cannot reach the hole.



**Note:**

1. There is only one ball and one hole in the maze.
2. Both the ball and hole exist on an empty space, and they will not be at the same position initially.
3. The given maze does not contain border (like the red rectangle in the example pictures), but you could assume the border of the maze are all walls.
4. The maze contains at least 2 empty spaces, and the width and the height of the maze won't exceed 30.

class Solution:

def findShortestWay(self, maze: List[List[int]], ball: List[int], hole: List[int]) -> str:

result = ["Z",2\*\*31]

def dfs(maze,hole,i,j,distance,path,visited,result):

#print(result)

if (i,j) in visited and visited[(i,j)] < distance: return

#print(i,j)

visited[(i,j)] = distance

# go up

traveled = 0

ii = i

while ii > 0 and maze[ii-1][j] == 0:

ii -= 1

traveled += 1

if [ii,j] == hole:

if traveled + distance <= result[1]:

path.append("u")

temp\_result = "".join(path)

if traveled + distance < result[1] or temp\_result < result[0]:

result[0] = temp\_result

result[1] = traveled + distance

path.pop()

return

return

if ii != i:

path.append("u")

dfs(maze,hole,ii,j,distance+traveled,path,visited,result)

path.pop()

# go down

traveled = 0

ii = i

while ii < len(maze)-1 and maze[ii+1][j] == 0:

ii += 1

traveled += 1

if [ii,j] == hole:

if traveled + distance <= result[1]:

path.append("d")

temp\_result = "".join(path)

if traveled + distance < result[1] or temp\_result < result[0]:

result[0] = temp\_result

result[1] = traveled + distance

path.pop()

return

return

if ii != i:

path.append("d")

dfs(maze,hole,ii,j,distance+traveled,path,visited,result)

path.pop()

# go left

traveled = 0

jj = j

while jj > 0 and maze[i][jj-1] == 0:

jj -=1

traveled += 1

if [i,jj] == hole:

if traveled + distance <= result[1]:

#print(traveled + distance)

path.append("l")

temp\_result = "".join(path)

if traveled + distance < result[1] or temp\_result < result[0]:

result[0] = temp\_result

result[1] = traveled + distance

path.pop()

return

return

if jj != j:

path.append("l")

dfs(maze,hole,i,jj,distance+traveled,path,visited,result)

path.pop()

# go right

traveled = 0

jj = j

while jj < len(maze[0])-1 and maze[i][jj+1] == 0:

jj +=1

traveled += 1

if [i,jj] == hole:

if traveled + distance <= result[1]:

path.append("r")

temp\_result = "".join(path)

if traveled + distance < result[1] or temp\_result < result[0]:

result[0] = temp\_result

result[1] = traveled + distance

path.pop()

return

return

if jj != j:

path.append("r")

dfs(maze,hole,i,jj,distance+traveled,path,visited,result)

path.pop()

visited = {}

dfs(maze,hole,ball[0],ball[1],0,[],visited,result)

if result[0] == "Z": return "impossible"

return result[0]

**489. Robot Room Cleaner**

Hard

60333FavoriteShare

Given a robot cleaner in a room modeled as a grid.

Each cell in the grid can be empty or blocked.

The robot cleaner with 4 given APIs can move forward, turn left or turn right. Each turn it made is 90 degrees.

When it tries to move into a blocked cell, its bumper sensor detects the obstacle and it stays on the current cell.

Design an algorithm to clean the entire room using only the 4 given APIs shown below.

interface Robot {

  // returns true if next cell is open and robot moves into the cell.

  // returns false if next cell is obstacle and robot stays on the current cell.

  boolean move();

// Robot will stay on the same cell after calling turnLeft/turnRight.

  // Each turn will be 90 degrees.

  void turnLeft();

  void turnRight();

// Clean the current cell.

void clean();

}

**Example:**

**Input:**

room = [

[1,1,1,1,1,0,1,1],

[1,1,1,1,1,0,1,1],

[1,0,1,1,1,1,1,1],

[0,0,0,1,0,0,0,0],

[1,1,1,1,1,1,1,1]

],

row = 1,

col = 3

**Explanation:**

All grids in the room are marked by either 0 or 1.

0 means the cell is blocked, while 1 means the cell is accessible.

The robot initially starts at the position of row=1, col=3.

From the top left corner, its position is one row below and three columns right.

**Notes:**

1. The input is only given to initialize the room and the robot's position internally. You must solve this problem "blindfolded". In other words, you must control the robot using only the mentioned 4 APIs, without knowing the room layout and the initial robot's position.
2. The robot's initial position will always be in an accessible cell.
3. The initial direction of the robot will be facing up.
4. All accessible cells are connected, which means the all cells marked as 1 will be accessible by the robot.
5. Assume all four edges of the grid are all surrounded by wall.

# """

# This is the robot's control interface.

# You should not implement it, or speculate about its implementation

# """

#class Robot:

# def move(self):

# """

# Returns true if the cell in front is open and robot moves into the cell.

# Returns false if the cell in front is blocked and robot stays in the current cell.

# :rtype bool

# """

#

# def turnLeft(self):

# """

# Robot will stay in the same cell after calling turnLeft/turnRight.

# Each turn will be 90 degrees.

# :rtype void

# """

#

# def turnRight(self):

# """

# Robot will stay in the same cell after calling turnLeft/turnRight.

# Each turn will be 90 degrees.

# :rtype void

# """

#

# def clean(self):

# """

# Clean the current cell.

# :rtype void

# """

class Solution:

def cleanRoom(self, robot):

"""

:type robot: Robot

:rtype: None

"""

def clean\_your\_ass(robot,x,y,DP):

robot.clean()

DP[y][x] = 4

#print(DP)

#print(x,y)

if DP[y-1][x] == -1:

if robot.move() == False: DP[y-1][x] = 0

else:

clean\_your\_ass(robot,x,y-1,DP)

robot.turnRight();

robot.turnRight();

robot.move();

robot.turnRight();

robot.turnRight();

if DP[y+1][x] == -1:

robot.turnRight();

robot.turnRight();

if robot.move() == False:

DP[y+1][x] = 0

robot.turnRight();

robot.turnRight();

else:

robot.turnRight();

robot.turnRight();

clean\_your\_ass(robot,x,y+1,DP)

robot.move();

if DP[y][x-1] == -1:

robot.turnLeft();

if robot.move() == False:

DP[y+1][x] = 0

robot.turnRight();

else:

robot.turnRight();

clean\_your\_ass(robot,x-1,y,DP)

robot.turnRight();

robot.move();

robot.turnLeft();

if DP[y][x+1] == -1:

robot.turnRight();

if robot.move() == False:

DP[y+1][x] = 0

robot.turnLeft();

else:

robot.turnLeft();

clean\_your\_ass(robot,x+1,y,DP)

robot.turnLeft();

robot.move();

robot.turnRight();

DP = []

for i in range(1000):

ass = [-1]\*1000

DP.append(ass)

startx = 500

starty = 500

clean\_your\_ass(robot,startx,starty,DP)

return

**471. Encode String with Shortest Length**

Hard

25516FavoriteShare

Given a **non-empty** string, encode the string such that its encoded length is the shortest.

The encoding rule is: k[encoded\_string], where the *encoded\_string* inside the square brackets is being repeated exactly *k* times.

**Note:**

1. *k* will be a positive integer and encoded string will not be empty or have extra space.
2. You may assume that the input string contains only lowercase English letters. The string's length is at most 160.
3. If an encoding process does not make the string shorter, then do not encode it. If there are several solutions, return any of them is fine.

**Example 1:**

Input: "aaa"

Output: "aaa"

Explanation: There is no way to encode it such that it is shorter than the input string, so we do not encode it.

**Example 2:**

Input: "aaaaa"

Output: "5[a]"

Explanation: "5[a]" is shorter than "aaaaa" by 1 character.

**Example 3:**

Input: "aaaaaaaaaa"

Output: "10[a]"

Explanation: "a9[a]" or "9[a]a" are also valid solutions, both of them have the same length = 5, which is the same as "10[a]".

**Example 4:**

Input: "aabcaabcd"

Output: "2[aabc]d"

Explanation: "aabc" occurs twice, so one answer can be "2[aabc]d".

**Example 5:**

Input: "abbbabbbcabbbabbbc"

Output: "2[2[abbb]c]"

Explanation: "abbbabbbc" occurs twice, but "abbbabbbc" can also be encoded to "2[abbb]c", so one answer can be "2[2[abbb]c]".

class Solution:

def encode(self, s: str) -> str:

data = {}

def dfs(s,data):

if len(s)<5: return s

if s in data: return data[s]

result = s

for i in range(1,len(s)//2+1):

temp = s[:i]

if len(s)%len(temp) == 0 and s == temp\*(len(s)//len(temp)):

temp\_result = str(len(s)//len(temp))+"["+dfs(temp,data)+"]"

if len(temp\_result)<len(result):result = temp\_result

for i in range(len(s)-1):

temp\_result = dfs(s[0:i+1],data)+dfs(s[i+1:],data)

if len(temp\_result)<len(result):result = temp\_result

data[s] = result

return result

return dfs(s,data)

**465. Optimal Account Balancing**

Hard

27848FavoriteShare

A group of friends went on holiday and sometimes lent each other money. For example, Alice paid for Bill's lunch for $10. Then later Chris gave Alice $5 for a taxi ride. We can model each transaction as a tuple (x, y, z) which means person x gave person y $z. Assuming Alice, Bill, and Chris are person 0, 1, and 2 respectively (0, 1, 2 are the person's ID), the transactions can be represented as [[0, 1, 10], [2, 0, 5]].

Given a list of transactions between a group of people, return the minimum number of transactions required to settle the debt.

**Note:**

1. A transaction will be given as a tuple (x, y, z). Note that x ≠ y and z > 0.
2. Person's IDs may not be linear, e.g. we could have the persons 0, 1, 2 or we could also have the persons 0, 2, 6.

**Example 1:**

**Input:**

[[0,1,10], [2,0,5]]

**Output:**

2

**Explanation:**

Person #0 gave person #1 $10.

Person #2 gave person #0 $5.

Two transactions are needed. One way to settle the debt is person #1 pays person #0 and #2 $5 each.

**Example 2:**

**Input:**

[[0,1,10], [1,0,1], [1,2,5], [2,0,5]]

**Output:**

1

**Explanation:**

Person #0 gave person #1 $10.

Person #1 gave person #0 $1.

Person #1 gave person #2 $5.

Person #2 gave person #0 $5.

Therefore, person #1 only need to give person #0 $4, and all debt is settled.

from collections import defaultdict

class Solution:

def minTransfers(self, transactions: List[List[int]]) -> int:

people = defaultdict(int)

for x,y,z in transactions:

people[x] -= z

people[y] += z

#print(people)

debt = []

for x in people.values():

if x != 0: debt.append(x)

#print(debt)

def dfs(start):

#print(start,"=>")

while start < len(debt) and debt[start] == 0: start += 1

result = 2\*\*31

seen = set()

for i in range(start+1,len(debt)):

if debt[i] not in seen and debt[i]\*debt[start] < 0:

if start == 0: print(debt)

debt[i] += debt[start]

result = min(result,dfs(start+1)+1)

#print(i,result,"aaaaaa")

debt[i] -= debt[start]

seen.add(debt[i])

if result == 2\*\*31: result = 0

#print(start,result)

return result

return dfs(0)

**428. Serialize and Deserialize N-ary Tree**

Hard

22910FavoriteShare

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 an N-ary tree. An N-ary tree is a rooted tree in which each node has no more than N children. There is no restriction on how your serialization/deserialization algorithm should work. You just need to ensure that an N-ary tree can be serialized to a string and this string can be deserialized to the original tree structure.

For example, you may serialize the following 3-ary tree



as [1 [3[5 6] 2 4]]. You do not necessarily need to follow this format, so please be creative and come up with different approaches yourself.

**Note:**

1. N is in the range of [1, 1000]
2. Do not use class member/global/static variables to store states. Your serialize and deserialize algorithms should be stateless.

"""

# Definition for a Node.

class Node(object):

def \_\_init\_\_(self, val, children):

self.val = val

self.children = children

"""

class Codec:

def serialize(self, root):

"""Encodes a tree to a single string.

:type root: Node

:rtype: str

"""

if root == None: return "shityourass"

def dfs(root):

result = [str(root.val),str(len(root.children))]

for x in root.children:

result.extend(dfs(x))

return result

return ",".join(dfs(root))

def deserialize(self, data):

"""Decodes your encoded data to tree.

:type data: str

:rtype: Node

"""

if data == "shityourass": return None

data = data.split(",")

#print(data)

def dfs(index,data):

root = Node(int(data[index]),[])

if data[index+1] == "0": return root, index+2

temp = []

new\_index = index+2

for i in range(int(data[index+1])):

node,new\_index = dfs(new\_index,data)

temp.append(node)

root.children = temp

return root,new\_index

return dfs(0,data)[0]

# Your Codec object will be instantiated and called as such:

# codec = Codec()

# codec.deserialize(codec.serialize(root))

**425. Word Squares**

Hard

38533FavoriteShare

Given a set of words **(without duplicates)**, find all [word squares](https://en.wikipedia.org/wiki/Word_square) you can build from them.

A sequence of words forms a valid word square if the *k*th row and column read the exact same string, where 0 ≤ *k* < max(numRows, numColumns).

For example, the word sequence ["ball","area","lead","lady"] forms a word square because each word reads the same both horizontally and vertically.

b a l l

a r e a

l e a d

l a d y

**Note:**

1. There are at least 1 and at most 1000 words.
2. All words will have the exact same length.
3. Word length is at least 1 and at most 5.
4. Each word contains only lowercase English alphabet a-z.

**Example 1:**

**Input:**

["area","lead","wall","lady","ball"]

**Output:**

[

[ "wall",

"area",

"lead",

"lady"

],

[ "ball",

"area",

"lead",

"lady"

]

]

**Explanation:**

The output consists of two word squares. The order of output does not matter (just the order of words in each word square matters).

**Example 2:**

**Input:**

["abat","baba","atan","atal"]

**Output:**

[

[ "baba",

"abat",

"baba",

"atan"

],

[ "baba",

"abat",

"baba",

"atal"

]

]

**Explanation:**

The output consists of two word squares. The order of output does not matter (just the order of words in each word square matters).

class Solution:

def wordSquares(self, words: List[str]) -> List[List[str]]:

result = set()

dic = {}

n = len(words[0])

for word in words:

for i in range(n-1):

prefix = word[:i+1]

if prefix in dic: dic[prefix].append(word)

else: dic[prefix]= [word]

def dfs(curr\_list,n,dic,result):

length = len(curr\_list)

if length == n:

result.add(tuple(curr\_list))

return

prefix = ""

for x in curr\_list:

prefix += x[length]

if prefix in dic:

for poop in dic[prefix]:

dfs(curr\_list+[poop],n,dic,result)

return

for word in words:

dfs([word],n,dic,result)

true\_result = []

for x in result: true\_result.append(x)

return true\_result

**411. Minimum Unique Word Abbreviation**

Hard

100103FavoriteShare

A string such as "word" contains the following abbreviations:

["word", "1ord", "w1rd", "wo1d", "wor1", "2rd", "w2d", "wo2", "1o1d", "1or1", "w1r1", "1o2", "2r1", "3d", "w3", "4"]

Given a target string and a set of strings in a dictionary, find an abbreviation of this target string with the ***smallest possible*** length such that it does not conflict with abbreviations of the strings in the dictionary.

Each **number** or letter in the abbreviation is considered length = 1. For example, the abbreviation "a32bc" has length = 4.

**Note:**

* In the case of multiple answers as shown in the second example below, you may return any one of them.
* Assume length of target string = **m**, and dictionary size = **n**. You may assume that **m ≤ 21**, **n ≤ 1000**, and **log2(n) + m ≤ 20**.

**Examples:**

"apple", ["blade"] -> "a4" (because "5" or "4e" conflicts with "blade")

"apple", ["plain", "amber", "blade"] -> "1p3" (other valid answers include "ap3", "a3e", "2p2", "3le", "3l1").

class Solution:

def minAbbreviation(self, target: str, dictionary: List[str]) -> str:

# reference https://leetcode.com/problems/minimum-unique-word-abbreviation/discuss/89880/C%2B%2B-Bit-Manipulation-%2B-DFS-solution

n = len(target)

limit = 1<<n

bit\_list = []

all\_diff\_bits = 0

for x in dictionary:

if len(x) != n: continue

temp = 0

bit = 1

for i in reversed(range(n)):

if x[i] != target[i]: temp += bit

bit <<= 1

bit\_list.append(temp)

all\_diff\_bits = all\_diff\_bits | temp

#print(bit\_list,all\_diff\_bits)

def get\_word\_length(mask):

# Return the length of abbreviation given bit sequence

cnt = 0

bit = 1

i = 0

while i < n:

if mask & bit:

cnt += 1

bit <<= 1

i+=1

else:

while i < n and mask & bit == 0:

bit <<= 1

i+=1

cnt += 1

return cnt

#print(get\_word\_length(16))

def dfs(bit,mask,best):

length = get\_word\_length(mask)

if length >= best[0]: return

is\_valid = True

for x in bit\_list:

if mask & x == 0:

is\_valid = False

break

if is\_valid:

best[0] = length

best[1] = mask

else:

for i in range(bit+1,n+1):

next\_bit = 1<<(i-1)

if next\_bit & all\_diff\_bits: dfs(i,mask+next\_bit,best)

best = [n+1,2\*\*n] # [length of abbreviation, actual mask]

dfs(0,0,best)

#print(best)

result = []

mask = best[1]

bit = 1

i = n-1

while i >= 0:

if mask & bit:

result.append(target[i])

bit <<= 1

i-=1

else:

cnt = 0

while i >=0 and mask & bit == 0:

bit <<= 1

i-=1

cnt += 1

result.append(str(cnt))

ans = ""

for x in reversed(result):

ans += x

return ans

**366. Find Leaves of Binary Tree**

Medium

66410FavoriteShare

Given a binary tree, collect a tree's nodes as if you were doing this: Collect and remove all leaves, repeat until the tree is empty.

**Example:**

**Input:** [1,2,3,4,5]

  1

/ \

2 3

/ \

4 5

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

**Explanation:**

1. Removing the leaves [4,5,3] would result in this tree:

1

/

2

2. Now removing the leaf [2] would result in this tree:

1

3. Now removing the leaf [1] would result in the empty tree:

[]

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

class Solution {

public:

int dfs(TreeNode\* root, vector<vector<int>> &result){

if (root == NULL) return -1;

int left = dfs(root->left,result);

int right = dfs(root->right,result);

int level = max(left,right)+1;

if (level >= result.size()){

result.push\_back(vector<int>{root->val});

}

else {

result[level].push\_back(root->val);

}

return level;

}

vector<vector<int>> findLeaves(TreeNode\* root) {

vector<vector<int>> result;

cout<<dfs(root,result);

return result;

}

};

**359. Logger Rate Limiter**

Easy

24072FavoriteShare

Design a logger system that receive stream of messages along with its timestamps, each message should be printed if and only if it is **not printed in the last 10 seconds**.

Given a message and a timestamp (in seconds granularity), return true if the message should be printed in the given timestamp, otherwise returns false.

It is possible that several messages arrive roughly at the same time.

**Example:**

Logger logger = new Logger();

// logging string "foo" at timestamp 1

logger.shouldPrintMessage(1, "foo"); returns true;

// logging string "bar" at timestamp 2

logger.shouldPrintMessage(2,"bar"); returns true;

// logging string "foo" at timestamp 3

logger.shouldPrintMessage(3,"foo"); returns false;

// logging string "bar" at timestamp 8

logger.shouldPrintMessage(8,"bar"); returns false;

// logging string "foo" at timestamp 10

logger.shouldPrintMessage(10,"foo"); returns false;

// logging string "foo" at timestamp 11

logger.shouldPrintMessage(11,"foo"); returns true;

class Logger:

def \_\_init\_\_(self):

"""

Initialize your data structure here.

"""

self.data = {}

def shouldPrintMessage(self, timestamp: int, message: str) -> bool:

"""

Returns true if the message should be printed in the given timestamp, otherwise returns false.

If this method returns false, the message will not be printed.

The timestamp is in seconds granularity.

"""

if message not in self.data:

self.data[message] = timestamp

return True

if timestamp - self.data[message] < 10: return False

self.data[message] = timestamp

return True

# Your Logger object will be instantiated and called as such:

# obj = Logger()

# param\_1 = obj.shouldPrintMessage(timestamp,message)

**358. Rearrange String k Distance Apart**

Hard

29514FavoriteShare

Given a non-empty string **s** and an integer **k**, rearrange the string such that the same characters are at least distance **k** from each other.

All input strings are given in lowercase letters. If it is not possible to rearrange the string, return an empty string "".

**Example 1:**

**Input:** s = "aabbcc", k = 3

**Output:** "abcabc"

**Explanation:** The same letters are at least distance 3 from each other.

**Example 2:**

**Input:** s = "aaabc", k = 3

**Output:** ""

**Explanation:** It is not possible to rearrange the string.

**Example 3:**

**Input:** s = "aaadbbcc", k = 2

**Output:** "abacabcd"

**Explanation:** The same letters are at least distance 2 from each other.

class Solution:

def rearrangeString(self, s: str, k: int) -> str:

if k ==0: return s

dic = {}

for x in s:

if x in dic: dic[x] += 1

else: dic[x] = 1

result = ""

heap = []

for x in dic.keys():

heap.append([-dic[x],x])

heapq.heapify(heap)

queue = []

for i in range(len(s)):

if len(heap) == 0: return ""

current = heapq.heappop(heap)

result += current[1]

current[0] += 1

queue.append(current)

if len(queue) == k:

add = queue.pop(0)

if add[0] < 0: heapq.heappush(heap,add)

return result

**353. Design Snake Game**

Medium

25494FavoriteShare

Design a [Snake game](https://en.wikipedia.org/wiki/Snake_(video_game)) that is played on a device with screen size = *width* x *height*. [Play the game online](http://patorjk.com/games/snake/) if you are not familiar with the game.

The snake is initially positioned at the top left corner (0,0) with length = 1 unit.

You are given a list of food's positions in row-column order. When a snake eats the food, its length and the game's score both increase by 1.

Each food appears one by one on the screen. For example, the second food will not appear until the first food was eaten by the snake.

When a food does appear on the screen, it is guaranteed that it will not appear on a block occupied by the snake.

**Example:**

Given width = 3, height = 2, and food = [[1,2],[0,1]].

Snake snake = new Snake(width, height, food);

Initially the snake appears at position (0,0) and the food at (1,2).

|S| | |

| | |F|

snake.move("R"); -> Returns 0

| |S| |

| | |F|

snake.move("D"); -> Returns 0

| | | |

| |S|F|

snake.move("R"); -> Returns 1 (Snake eats the first food and right after that, the second food appears at (0,1) )

| |F| |

| |S|S|

snake.move("U"); -> Returns 1

| |F|S|

| | |S|

snake.move("L"); -> Returns 2 (Snake eats the second food)

| |S|S|

| | |S|

snake.move("U"); -> Returns -1 (Game over because snake collides with border)

class SnakeGame:

def \_\_init\_\_(self, width: int, height: int, food: List[List[int]]):

"""

Initialize your data structure here.

@param width - screen width

@param height - screen height

@param food - A list of food positions

E.g food = [[1,1], [1,0]] means the first food is positioned at [1,1], the second is at [1,0].

"""

self.width = width

self.height = height

self.food = food

self.body = []

self.head = [0,0]

self.score = 0

def move(self, direction: str) -> int:

"""

Moves the snake.

@param direction - 'U' = Up, 'L' = Left, 'R' = Right, 'D' = Down

@return The game's score after the move. Return -1 if game over.

Game over when snake crosses the screen boundary or bites its body.

"""

new\_head = self.head.copy()

if direction == "U": new\_head[0] -= 1

elif direction == "D": new\_head[0] += 1

elif direction == "L": new\_head[1] -= 1

elif direction == "R": new\_head[1] += 1

else: return

if new\_head[0] < 0 or new\_head[0] >= self.height or new\_head[1] < 0 or new\_head[1] >= self.width: return -1

print(new\_head)

if len(self.food) >0 and new\_head == self.food[0]:

self.food.pop(0)

self.body.append(self.head)

self.head = new\_head

self.score += 1

return self.score

else:

self.body.append(self.head)

self.body.pop(0)

self.head = new\_head

if new\_head in self.body: return -1

return self.score

# Your SnakeGame object will be instantiated and called as such:

# obj = SnakeGame(width, height, food)

# param\_1 = obj.move(direction)

**346. Moving Average from Data Stream**

Easy

33938FavoriteShare

Given a stream of integers and a window size, calculate the moving average of all integers in the sliding window.

**Example:**

MovingAverage m = new MovingAverage(3);

m.next(1) = 1

m.next(10) = (1 + 10) / 2

m.next(3) = (1 + 10 + 3) / 3

m.next(5) = (10 + 3 + 5) / 3

class MovingAverage:

def \_\_init\_\_(self, size: int):

"""

Initialize your data structure here.

"""

self.queue = []

self.max\_size = size

def next(self, val: int) -> float:

if len(self.queue) < self.max\_size:

self.queue.append(val)

return sum(self.queue)/len(self.queue)

self.queue.pop(0)

self.queue.append(val)

return sum(self.queue)/self.max\_size

# Your MovingAverage object will be instantiated and called as such:

# obj = MovingAverage(size)

# param\_1 = obj.next(val)

**340. Longest Substring with At Most K Distinct Characters**

Hard

63320FavoriteShare

Given a string, find the length of the longest substring T that contains at most *k* distinct characters.

**Example 1:**

**Input:** s = "eceba", k = 2

**Output:** 3

**Explanation:** T is "ece" which its length is 3.

**Example 2:**

**Input:** s = "aa", k = 1

**Output:** 2

**Explanation:** T is "aa" which its length is 2.

class Solution:

def lengthOfLongestSubstringKDistinct(self, s: str, k: int) -> int:

if k == 0: return 0

if len(s) == 0: return 0

dic = {}

result\_l = 0

result\_r = 0

result\_length = 0

l = 0

r = 0

for i in range(len(s)):

if s[i] in dic:

dic[s[i]] += 1

else:

if len(dic) < k:

dic[s[i]] = 1

else:

while len(dic) >= k:

dic[s[l]] -= 1

if dic[s[l]] == 0:

del dic[s[l]]

l += 1

dic[s[i]] = 1

r = i

if (r-l+1) > result\_length:

result\_length = r-l+1

result\_l = l

result\_r = r

#return s[result\_l:result\_r+1]

return result\_length

**317. Shortest Distance from All Buildings**

Hard

52029FavoriteShare

You want to build a house on an *empty* land which reaches all buildings in the shortest amount of distance. You can only move up, down, left and right. You are given a 2D grid of values **0**, **1** or **2**, where:

* Each **0** marks an empty land which you can pass by freely.
* Each **1** marks a building which you cannot pass through.
* Each **2** marks an obstacle which you cannot pass through.

**Example:**

**Input:** [[1,0,2,0,1],[0,0,0,0,0],[0,0,1,0,0]]

1 - 0 - 2 - 0 - 1

| | | | |

0 - 0 - 0 - 0 - 0

| | | | |

0 - 0 - 1 - 0 - 0

**Output:** 7

**Explanation:** Given three buildings at (0,0), (0,4), (2,2), and an obstacle at (0,2),

the point (1,2) is an ideal empty land to build a house, as the total

  travel distance of 3+3+1=7 is minimal. So return 7.

**Note:**  
There will be at least one building. If it is not possible to build such house according to the above rules, return -1.

class Solution:

def shortestDistance(self, grid: List[List[int]]) -> int:

def bfs(row,col,grid,visited,score,visit\_count):

queue = [[row+1,col],[row-1,col],[row,col+1],[row,col-1]]

step = 1

while len(queue) > 0:

new\_queue = []

for row,col in queue:

#print(row,col)

if row < 0 or row >= len(grid): continue

if col < 0 or col >= len(grid[0]): continue

if grid[row][col] != 0: continue

if visited[row][col] == 1: continue

#print(row,col)

score[row][col] += step

visited[row][col] = 1

visit\_count[row][col] += 1

new\_queue.append([row+1,col])

new\_queue.append([row-1,col])

new\_queue.append([row,col+1])

new\_queue.append([row,col-1])

step += 1

queue = new\_queue

house = []

score = []

for i in range(len(grid)):

temp = [0] \* len(grid[0])

for j in range(len(grid[0])):

if grid[i][j] == 1:

house.append([i,j])

temp[j] = 2\*\*31

elif grid[i][j] == 2:

temp[j] = 2\*\*31

else:

temp[j] = 0

score.append(temp)

#print(house)

#print(score)

temp = [0]\*len(grid[0])

visit\_count = [temp[:] for i in range(len(grid))]

for row,col in house:

temp = [0]\*len(grid[0])

visited = [temp[:] for i in range(len(grid))]

bfs(row,col,grid,visited,score,visit\_count)

result = 2\*\*31

for i in range(len(score)):

for j in range(len(score[0])):

if score[i][j] > 0 and visit\_count[i][j] == len(house): result = min(result,score[i][j])

print(score)

if result == 2\*\*31: result = -1

return result

**308. Range Sum Query 2D - Mutable**

Hard

30948FavoriteShare

Given a 2D matrix *matrix*, find the sum of the elements inside the rectangle defined by its upper left corner (*row*1, *col*1) and lower right corner (*row*2, *col*2).

  
The above rectangle (with the red border) is defined by (row1, col1) = **(2, 1)** and (row2, col2) = **(4, 3)**, which contains sum = **8**.

**Example:**

Given matrix = [

[3, 0, 1, 4, 2],

[5, 6, 3, 2, 1],

[1, 2, 0, 1, 5],

[4, 1, 0, 1, 7],

[1, 0, 3, 0, 5]

]

sumRegion(2, 1, 4, 3) -> 8

update(3, 2, 2)

sumRegion(2, 1, 4, 3) -> 10

**Note:**

1. The matrix is only modifiable by the *update* function.
2. You may assume the number of calls to *update* and *sumRegion* function is distributed evenly.
3. You may assume that *row*1 ≤ *row*2 and *col*1 ≤ *col*2.

class NumMatrix:

def \_\_init\_\_(self, matrix: List[List[int]]):

self.data = matrix

h = len(self.data)

if h == 0:

self.tree = []

return

w = len(self.data[0])

goal = max(h,w)

curr = 1

while curr < goal: curr \*= 2

self.tree = [0]\*(curr\*curr+curr\*curr//2)

def build\_tree(tree,data,wl,wr,hl,hr,current):

if wl == wr and hl == hr:

#print(wl,hl,current)

tree[current] = data[hl][wl]

return data[hl][wl]

elif wl == wr:

hmid = (hl+hr)//2

a = build\_tree(tree,data,wl,wr,hl,hmid,current\*4+1)

b = build\_tree(tree,data,wl,wr,hmid+1,hr,current\*4+3)

tree[current] = a+b

return a+b

elif hl == hr:

wmid = (wl+wr)//2

a = build\_tree(tree,data,wl,wmid,hl,hr,current\*4+1)

b = build\_tree(tree,data,wmid+1,wr,hl,hr,current\*4+2)

tree[current] = a+b

return a+b

else:

hmid = (hl+hr)//2

wmid = (wl+wr)//2

a = build\_tree(tree,data,wl,wmid,hl,hmid,current\*4+1)

b = build\_tree(tree,data,wmid+1,wr,hl,hmid,current\*4+2)

c = build\_tree(tree,data,wl,wmid,hmid+1,hr,current\*4+3)

d = build\_tree(tree,data,wmid+1,wr,hmid+1,hr,current\*4+4)

tree[current] = a+b+c+d

return a+b+c+d

build\_tree(self.tree,self.data,0,w-1,0,h-1,0)

#print(self.tree)

def update(self, row: int, col: int, val: int) -> None:

def change(tree,diff,w\_goal,h\_goal,wl,wr,hl,hr,current):

tree[current] += diff

if w\_goal == wl and w\_goal == wr and h\_goal == hl and h\_goal == hr: return

elif wl == wr:

hmid = (hl+hr)//2

if h\_goal <= hmid: change(tree,diff,w\_goal,h\_goal,wl,wr,hl,hmid,current\*4+1)

else: change(tree,diff,w\_goal,h\_goal,wl,wr,hmid+1,hr,current\*4+3)

elif hl == hr:

wmid = (wl+wr)//2

if w\_goal <= wmid: change(tree,diff,w\_goal,h\_goal,wl,wmid,hl,hr,current\*4+1)

else: change(tree,diff,w\_goal,h\_goal,wmid+1,wr,hl,hr,current\*4+2)

else:

hmid = (hl+hr)//2

wmid = (wl+wr)//2

if h\_goal <= hmid:

if w\_goal <= wmid:change(tree,diff,w\_goal,h\_goal,wl,wmid,hl,hmid,current\*4+1)

else:change(tree,diff,w\_goal,h\_goal,wmid+1,wr,hl,hmid,current\*4+2)

else:

if w\_goal <= wmid:change(tree,diff,w\_goal,h\_goal,wl,wmid,hmid+1,hr,current\*4+3)

else:change(tree,diff,w\_goal,h\_goal,wmid+1,wr,hmid+1,hr,current\*4+4)

diff = val - self.data[row][col]

self.data[row][col] = val

h = len(self.data)

w = len(self.data[0])

change(self.tree,diff,col,row,0,w-1,0,h-1,0)

#print(self.tree)

def sumRegion(self, row1: int, col1: int, row2: int, col2: int) -> int:

def get(tree,row1,row2,col1,col2,wl,wr,hl,hr,current):

if row1 == wl and row2 == wr and col1 == hl and col2 == hr: return tree[current]

hmid = (hl+hr)//2

wmid = (wl+wr)//2

if row2 <= wmid:

if col2 <= hmid:

return get(tree,row1,row2,col1,col2,wl,wmid,hl,hmid,current\*4+1)

elif col1 > hmid:

return get(tree,row1,row2,col1,col2,wl,wmid,hmid+1,hr,current\*4+3)

else:

a = get(tree,row1,row2,col1,hmid,wl,wmid,hl,hmid,current\*4+1)

b = get(tree,row1,row2,hmid+1,col2,wl,wmid,hmid+1,hr,current\*4+3)

return a + b

elif row1 > wmid:

if col2 <= hmid:

return get(tree,row1,row2,col1,col2,wmid+1,wr,hl,hmid,current\*4+2)

elif col1 > hmid:

return get(tree,row1,row2,col1,col2,wmid+1,wr,hmid+1,hr,current\*4+4)

else:

a = get(tree,row1,row2,col1,hmid,wmid+1,wr,hl,hmid,current\*4+2)

b = get(tree,row1,row2,hmid+1,col2,wmid+1,wr,hmid+1,hr,current\*4+4)

return a + b

else:

if col2 <= hmid:

a = get(tree,row1,wmid,col1,col2,wl,wmid,hl,hmid,current\*4+1)

b = get(tree,wmid+1,row2,col1,col2,wmid+1,wr,hl,hmid,current\*4+2)

return a + b

elif col1 > hmid:

a = get(tree,row1,wmid,col1,col2,wl,wmid,hmid+1,hr,current\*4+3)

b = get(tree,wmid+1,row2,col1,col2,wmid+1,wr,hmid+1,hr,current\*4+4)

return a + b

else:

a = get(tree,row1,wmid,col1,hmid,wl,wmid,hl,hmid,current\*4+1)

b = get(tree,wmid+1,row2,col1,hmid,wmid+1,wr,hl,hmid,current\*4+2)

c = get(tree,row1,wmid,hmid+1,col2,wl,wmid,hmid+1,hr,current\*4+3)

d = get(tree,wmid+1,row2,hmid+1,col2,wmid+1,wr,hmid+1,hr,current\*4+4)

return a+b+c+d

h = len(self.data)

w = len(self.data[0])

return get(self.tree,col1,col2,row1,row2,0,w-1,0,h-1,0)

# Your NumMatrix object will be instantiated and called as such:

# obj = NumMatrix(matrix)

# obj.update(row,col,val)

# param\_2 = obj.sumRegion(row1,col1,row2,col2)

**305. Number of Islands II**

Hard

5909FavoriteShare

A 2d grid map of m rows and n columns is initially filled with water. We may perform an *addLand*operation which turns the water at position (row, col) into a land. Given a list of positions to operate, **count the number of islands after each *addLand* operation**. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

**Example:**

**Input:** m = 3, n = 3, positions = [[0,0], [0,1], [1,2], [2,1]]

**Output:** [1,1,2,3]

**Explanation:**

Initially, the 2d grid grid is filled with water. (Assume 0 represents water and 1 represents land).

0 0 0

0 0 0

0 0 0

Operation #1: addLand(0, 0) turns the water at grid[0][0] into a land.

1 0 0

0 0 0 Number of islands = 1

0 0 0

Operation #2: addLand(0, 1) turns the water at grid[0][1] into a land.

1 1 0

0 0 0 Number of islands = 1

0 0 0

Operation #3: addLand(1, 2) turns the water at grid[1][2] into a land.

1 1 0

0 0 1 Number of islands = 2

0 0 0

Operation #4: addLand(2, 1) turns the water at grid[2][1] into a land.

1 1 0

0 0 1 Number of islands = 3

0 1 0

**Follow up:**

Can you do it in time complexity O(k log mn), where k is the length of the positions?

class Solution:

def numIslands2(self, m: int, n: int, positions: List[List[int]]) -> List[int]:

data = [-1 for i in range(m\*n)]

island = [[0]\*n for i in range(m)]

result = []

overlap = set()

def father(data,k):

temp = k

while data[temp] != temp:

temp = data[temp]

return temp

cnt = 0

for i,j in positions:

#print(data)

id = i\*n+j

if island[i][j] != 0:

result.append(result[-1])

continue

data[id] = id

island[i][j] = 1

cnt += 1

neighboor = []

overlap.clear()

if i > 0 and island[i-1][j] == 1:

#print(father(result,(i-1)\*n+j))

neighboor.append((i-1)\*n+j)

if i < m-1 and island[i+1][j] == 1:

#print(father(result,(i+1)\*n+j))

neighboor.append((i+1)\*n+j)

if j > 0 and island[i][j-1] == 1:

#print(father(result,i\*n+j-1))

neighboor.append(i\*n+j-1)

if j < n-1 and island[i][j+1] == 1:

#print(father(result,i\*n+j+1))

neighboor.append(i\*n+j+1)

#print(neighboor)

for x in neighboor:

#print(x)

overlap.add(father(data,x))

cnt -= len(overlap)

for x in overlap:

data[x] = id

#print(data)

#print(overlap)

'''count = set()

for k in range(m\*n):

if data[k] == -1: continue

count.add(father(result,k))

result.append(len(count))'''

result.append(cnt)

return result

**302. Smallest Rectangle Enclosing Black Pixels**

Hard

15243FavoriteShare

An image is represented by a binary matrix with 0 as a white pixel and 1 as a black pixel. The black pixels are connected, i.e., there is only one black region. Pixels are connected horizontally and vertically. Given the location (x, y) of one of the black pixels, return the area of the smallest (axis-aligned) rectangle that encloses all black pixels.

**Example:**

**Input:**

[

"0010",

"0110",

"0100"

]

and x = 0, y = 2

**Output:** 6

class Solution:

def minArea(self, image: List[List[str]], x: int, y: int) -> int:

def left\_bs(image,l,r):

while (r-l) > 1:

mid = (l+r)//2

found = False

for i in range(len(image)):

if image[i][mid] == "1":

found = True

break

if found: r = mid

else: l = mid

found = False

for i in range(len(image)):

if image[i][l] == "1":

found = True

break

if found: return l

else: return r

def right\_bs(image,l,r):

while (r-l) > 1:

mid = (l+r)//2

found = False

for i in range(len(image)):

if image[i][mid] == "1":

found = True

break

if found: l = mid

else: r = mid

found = False

for i in range(len(image)):

if image[i][r] == "1":

found = True

break

if found: return r

else: return l

def up\_bs(image,l,r):

while (r-l) > 1:

mid = (l+r)//2

found = False

for i in range(len(image[0])):

if image[mid][i] == "1":

found = True

break

if found: r = mid

else: l = mid

found = False

for i in range(len(image[0])):

if image[l][i] == "1":

found = True

break

if found: return l

else: return r

def down\_bs(image,l,r):

while (r-l) > 1:

mid = (l+r)//2

found = False

for i in range(len(image[0])):

if image[mid][i] == "1":

found = True

break

if found: l = mid

else: r = mid

found = False

for i in range(len(image[0])):

#print(r,i)

if image[r][i] == "1":

found = True

break

if found: return r

else: return l

height = len(image)-1

width = len(image[0])-1

print(right\_bs(image,y,width) )

print(left\_bs(image,0,y))

print(down\_bs(image,x,height))

print(up\_bs(image,0,x))

return (right\_bs(image,y,width) - left\_bs(image,0,y)+1)\*(down\_bs(image,x,height) - up\_bs(image,0,x)+1)

**296. Best Meeting Point**

Hard

31322FavoriteShare

A group of two or more people wants to meet and minimize the total travel distance. You are given a 2D grid of values 0 or 1, where each 1 marks the home of someone in the group. The distance is calculated using [Manhattan Distance](http://en.wikipedia.org/wiki/Taxicab_geometry), where distance(p1, p2) = |p2.x - p1.x| + |p2.y - p1.y|.

**Example:**

**Input:**

1 - 0 - 0 - 0 - 1

| | | | |

0 - 0 - 0 - 0 - 0

| | | | |

0 - 0 - 1 - 0 - 0

**Output: 6**

**Explanation:** Given three people living at (0,0), (0,4), and (2,2):

  The point (0,2) is an ideal meeting point, as the total travel distance

  of 2+2+2=6 is minimal. So return 6.

class Solution:

def minTotalDistance(self, grid: List[List[int]]) -> int:

def med(li):

if len(li)%2: return li[len(li)//2]

else: return

hori= []

ver = []

for i in range(len(grid)):

for j in range(len(grid[0])):

if grid[i][j]:

hori.append(j)

ver.append(i)

#print(hori)

#print(ver)

hori.sort()

ver.sort()

mid\_x = hori[len(hori)//2]

mid\_y = ver[len(ver)//2]

result = 0

for x in hori:

result += abs(x-mid\_x)

for x in ver:

result += abs(x-mid\_y)

return result

**291. Word Pattern II**

Hard

29120FavoriteShare

Given a pattern and a string str, find if str follows the same pattern.

Here **follow** means a full match, such that there is a bijection between a letter in pattern and a **non-empty** substring in str.

**Example 1:**

**Input:** pattern = "abab", str = "redblueredblue"

**Output:** true

**Example 2:**

**Input:** pattern = pattern = "aaaa", str = "asdasdasdasd"

**Output:** true

**Example 3:**

**Input:** pattern = "aabb", str = "xyzabcxzyabc"

**Output:** false

**Notes:**  
You may assume both pattern and str contains only lowercase letters.

class Solution:

def wordPatternMatch(self, pattern: str, word: str) -> bool:

dic = {}

seen = set()

def dfs(pi,wi,dic,seen):

#print(dic,seen,pi,wi)

if pi == len(pattern) and wi == len(word): return True

elif pi >= len(pattern) or wi >= len(word): return False

if pattern[pi] in dic:

mapping = dic[pattern[pi]]

if mapping != word[wi:wi+len(mapping)]: return False

return dfs(pi+1,wi+len(mapping),dic,seen)

else:

for i in range(wi,len(word)-len(pattern)+pi+1):

mapping = word[wi:i+1]

if mapping in seen: continue

else:

dic[pattern[pi]] = mapping

seen.add(mapping)

if dfs(pi+1,i+1,dic,seen) == True: return True

else:

del dic[pattern[pi]]

seen.remove(mapping)

return False

return dfs(0,0,dic,seen)

**288. Unique Word Abbreviation**

Medium

75978FavoriteShare

An abbreviation of a word follows the form <first letter><number><last letter>. Below are some examples of word abbreviations:

a) it --> it (no abbreviation)

1

↓

b) d|o|g --> d1g

1 1 1

1---5----0----5--8

↓ ↓ ↓ ↓ ↓

c) i|nternationalizatio|n --> i18n

1

1---5----0

  ↓ ↓ ↓

d) l|ocalizatio|n --> l10n

Assume you have a dictionary and given a word, find whether its abbreviation is unique in the dictionary. A word's abbreviation is unique if no *other* word from the dictionary has the same abbreviation.

**Example:**

Given dictionary = [ "deer", "door", "cake", "card" ]

isUnique("dear") -> false

isUnique("cart") -> true

isUnique("cane") -> false

isUnique("make") -> true

class ValidWordAbbr:

def abv(self,word):

if len(word) < 3: return word

return word[0]+str(len(word)-2)+word[-1]

def \_\_init\_\_(self, dictionary: List[str]):

self.dictionary = dictionary

self.dic = {}

for x in set(dictionary):

y = self.abv(x)

if y not in self.dic: self.dic[y] = 1

else: self.dic[y] += 1

print(self.dic)

print(self.dictionary)

def isUnique(self, word: str) -> bool:

x = self.abv(word)

if x not in self.dic: return True

if word not in self.dictionary: return False

if self.dic[x] == 1: return True

return False

# Your ValidWordAbbr object will be instantiated and called as such:

# obj = ValidWordAbbr(dictionary)

# param\_1 = obj.isUnique(word)

**272. Closest Binary Search Tree Value II**

Hard

41517FavoriteShare

Given a non-empty binary search tree and a target value, find *k* values in the BST that are closest to the target.

**Note:**

* Given target value is a floating point.
* You may assume *k* is always valid, that is: *k* ≤ total nodes.
* You are guaranteed to have only one unique set of *k* values in the BST that are closest to the target.

**Example:**

**Input:** root = [4,2,5,1,3], target = 3.714286, and *k* = 2

4

/ \

2 5

/ \

1 3

**Output:** [4,3]

**Follow up:**  
Assume that the BST is balanced, could you solve it in less than *O*(*n*) runtime (where *n* = total nodes)?

Accepted

41,453

Submissions

89,426

# Definition for a binary tree node.

# class TreeNode:

# def \_\_init\_\_(self, x):

# self.val = x

# self.left = None

# self.right = None

class Solution:

def closestKValues(self, root: TreeNode, target: float, k: int) -> List[int]:

def build\_low\_stack(root,target,stack):

curr = root

while curr != None:

if curr.val == target:

stack.append(curr)

break

elif curr.val < target:

stack.append(curr)

curr = curr.right

else: curr = curr.left

def build\_high\_stack(root,target,stack):

curr = root

while curr != None:

if curr.val == target:

stack.append(curr)

break

elif curr.val > target:

stack.append(curr)

curr = curr.left

else: curr = curr.right

def get\_next\_low(low\_stack):

result = low\_stack.pop()

curr = result.left

while curr != None:

low\_stack.append(curr)

curr = curr.right

return result.val

def get\_next\_high(high\_stack):

result = high\_stack.pop()

curr = result.right

while curr != None:

high\_stack.append(curr)

curr = curr.left

return result.val

result = []

low\_stack = []

high\_stack = []

build\_low\_stack(root,target,low\_stack)

build\_high\_stack(root,target,high\_stack)

#print(low\_stack)

#print(high\_stack)

if low\_stack and high\_stack and low\_stack[-1] == high\_stack[-1]:

get\_next\_low(low\_stack)

while k:

if len(low\_stack) == 0:

result.append(get\_next\_high(high\_stack))

elif len(high\_stack) == 0:

result.append(get\_next\_low(low\_stack))

else:

#print(low\_stack[-1].val,high\_stack[-1].val)

if abs(low\_stack[-1].val-target) <= abs(high\_stack[-1].val-target):

result.append(get\_next\_low(low\_stack))

else:

result.append(get\_next\_high(high\_stack))

#print(low\_stack,high\_stack)

k -= 1

#print(low\_stack,high\_stack)

return result

**271. Encode and Decode Strings**

Medium

27298FavoriteShare

Design an algorithm to encode **a list of strings** to **a string**. The encoded string is then sent over the network and is decoded back to the original list of strings.

Machine 1 (sender) has the function:

string encode(vector<string> strs) {

// ... your code

return encoded\_string;

}

Machine 2 (receiver) has the function:

vector<string> decode(string s) {

//... your code

return strs;

}

So Machine 1 does:

string encoded\_string = encode(strs);

and Machine 2 does:

vector<string> strs2 = decode(encoded\_string);

strs2 in Machine 2 should be the same as strs in Machine 1.

Implement the encode and decode methods.

**Note:**

* The string may contain any possible characters out of 256 valid ascii characters. Your algorithm should be generalized enough to work on any possible characters.
* Do not use class member/global/static variables to store states. Your encode and decode algorithms should be stateless.
* Do not rely on any library method such as eval or serialize methods. You should implement your own encode/decode algorithm.

class Codec:

def encode(self, strs):

"""Encodes a list of strings to a single string.

:type strs: List[str]

:rtype: str

"""

if strs == []: return ""

result = ""

for x in strs:

result += str(len(x))+" "+x

return result

def decode(self, s):

"""Decodes a single string to a list of strings.

:type s: str

:rtype: List[str]

"""

if s == "": return []

print(s)

result = []

number\_mode = True

number = ""

i = 0

while 1:

if i >= len(s): break

if number\_mode:

if s[i] == " ":number\_mode = False

else: number+=s[i]

i+=1

else:

if int(number) == 0:

result.append("")

else:

result.append(s[i:i+int(number)])

i = i+int(number)

number\_mode = True

number = ""

if number != "": result.append("")

return result

# Your Codec object will be instantiated and called as such:

# codec = Codec()

# codec.decode(codec.encode(strs))

**269. Alien Dictionary**

Hard

998205FavoriteShare

There is a new alien language which uses the latin alphabet. However, the order among letters are unknown to you. You receive a list of **non-empty** words from the dictionary, where **words are sorted lexicographically by the rules of this new language**. Derive the order of letters in this language.

**Example 1:**

**Input:**

[

"wrt",

"wrf",

"er",

"ett",

"rftt"

]

**Output:** "wertf"

**Example 2:**

**Input:**

[

"z",

"x"

]

**Output:** "zx"

**Example 3:**

**Input:**

[

"z",

"x",

"z"

]

**Output:** ""

**Explanation:** The order is invalid, so return "".

**Note:**

1. You may assume all letters are in lowercase.
2. You may assume that if a is a prefix of b, then a must appear before b in the given dictionary.
3. If the order is invalid, return an empty string.
4. There may be multiple valid order of letters, return any one of them is fine.

class Solution:

def alienOrder(self, words: List[str]) -> str:

n = len(words)

if n < 1: return ""

child = {} # record chars that are behind the key char

parent = {} # record how many chars are in front of the key char

for word in words:

for char in word:

if char not in parent: parent[char] = 0

for i in range(n-1):

a = words[i]

b = words[i+1]

for j in range(min(len(a),len(b))):

if a[j] != b[j]:

if a[j] not in child: child[a[j]] = [b[j]]

else: child[a[j]].append(b[j])

parent[b[j]] += 1

break

#print(child,parent)

queue = []

for key,val in parent.items():

if val == 0: queue.append(key)

result = []

while queue:

x = queue.pop(0)

result.append(x)

if x not in child: continue

for baby in child[x]:

parent[baby] -= 1

if parent[baby] == 0: queue.append(baby)

if len(result) != len(parent): return ""

return "".join(result)

**265. Paint House II**

Hard

37312FavoriteShare

There are a row of *n* houses, each house can be painted with one of the *k* colors. The cost of painting each house with a certain color is different. You have to paint all the houses such that no two adjacent houses have the same color.

The cost of painting each house with a certain color is represented by a *n* x *k* cost matrix. For example, costs[0][0] is the cost of painting house 0 with color 0; costs[1][2] is the cost of painting house 1 with color 2, and so on... Find the minimum cost to paint all houses.

**Note:**  
All costs are positive integers.

**Example:**

**Input:** [[1,5,3],[2,9,4]]

**Output:** 5

**Explanation:** Paint house 0 into color 0, paint house 1 into color 2. Minimum cost: 1 + 4 = 5;

  Or paint house 0 into color 2, paint house 1 into color 0. Minimum cost: 3 + 2 = 5.

**Follow up:**  
Could you solve it in *O*(*nk*) runtime?

class Solution:

def minCostII(self, costs: List[List[int]]) -> int:

if costs == []: return 0

color = len(costs[0])

house = len(costs)

if house == 1 and color == 1: return costs[0][0]

dp = [[2\*\*31]\*color for i in range(house+1)]

for i in range(color):

dp[-1][i] = 0

#print(dp)

for i in reversed(range(house)):

for j in range(color):

for k in range(color):

if k == j: continue

dp[i][j] = min(dp[i][j],costs[i][j]+dp[i+1][k])

#print(dp)

return min(dp[0])

**253. Meeting Rooms II**

Medium

156426FavoriteShare

Given an array of meeting time intervals consisting of start and end times [[s1,e1],[s2,e2],...] (si < ei), find the minimum number of conference rooms required.

**Example 1:**

**Input:** [[0, 30],[5, 10],[15, 20]]

**Output:** 2

**Example 2:**

**Input:** [[7,10],[2,4]]

**Output:** 1

**NOTE:** input types have been changed on April 15, 2019. Please reset to default code definition to get new method signature.

class Solution:

def minMeetingRooms(self, intervals: List[List[int]]) -> int:

intervals.sort()

heap = []

max\_room = 0

for start,stop in intervals:

if len(heap) == 0: heapq.heappush(heap, stop)

else:

while len(heap) != 0 and heap[0] <= start:

heapq.heappop(heap)

heapq.heappush(heap, stop)

current\_room = len(heap)

if current\_room > max\_room: max\_room = current\_room

#print(heap)

return max\_room

**249. Group Shifted Strings**

Medium

29952FavoriteShare

Given a string, we can "shift" each of its letter to its successive letter, for example: "abc" -> "bcd". We can keep "shifting" which forms the sequence:

"abc" -> "bcd" -> ... -> "xyz"

Given a list of strings which contains only lowercase alphabets, group all strings that belong to the same shifting sequence.

**Example:**

**Input:** ["abc", "bcd", "acef", "xyz", "az", "ba", "a", "z"],

**Output:**

[

["abc","bcd","xyz"],

["az","ba"],

["acef"],

["a","z"]

]

class Solution {

public:

vector<vector<string>> groupStrings(vector<string>& strings) {

unordered\_map<string,vector<string>> map;

for (string x : strings){

string temp = x;

for (int i = 0; i < x.size();i++){

if (x[i] < x[0]){

temp[i] = '0'+26 + x[i]-x[0];

}

else temp[i] = '0'+x[i]-x[0];

}

//cout<<temp<<endl;

if (map.find(temp)!=map.end()){

map[temp].push\_back(x);

}

else map[temp] = vector<string>{x};

}

vector<vector<string>> result;

for (auto it = map.begin();it != map.end();it++){

result.push\_back(it->second);

}

return result;

}

};

**248. Strobogrammatic Number III**

Hard

130118FavoriteShare

A strobogrammatic number is a number that looks the same when rotated 180 degrees (looked at upside down).

Write a function to count the total strobogrammatic numbers that exist in the range of low <= num <= high.

**Example:**

**Input:** low = "50", high = "100"

**Output:** 3

**Explanation:** 69, 88, and 96 are three strobogrammatic numbers.

**Note:**  
Because the range might be a large number, the *low* and *high* numbers are represented as string.

class Solution:

def strobogrammaticInRange(self, low: str, high: str) -> int:

if int(high) < int(low): return 0

l = len(low)

r = len(high)

result = 0

for i in range(l+1,r):

if i%2:result += 3\*4\*(5\*\*((i-3)//2))

else: result += 4\*(5\*\*((i-2)//2))

dic = {'6': '9', '9': '6', '8': '8', '0': '0', '1': '1'}

buf = collections.deque()

def is\_valid(buf):

x = "".join(buf)

# print(x)

if l == r: return low <= x <= high

elif len(x) == l:

# print(x >= low)

return x >= low

elif len(x) == r:

# print(x <= high)

return x <= high

return True

def make\_strobo(buf,length):

result = 0

if len(buf) == length:

if buf[0] != '0' and is\_valid(buf):

result += 1

return result

elif len(buf) == length-1:

for x in ('0','1','8'):

buf.insert(len(buf)//2,x)

if buf[0] != '0' and is\_valid(buf):

result += 1

del buf[len(buf)//2]

return result

else:

for x in dic:

buf.append(x)

buf.appendleft(dic[x])

result += make\_strobo(buf,length)

buf.pop()

buf.popleft()

# print(result)

return result

result += make\_strobo(buf,l)

if r != l: result += make\_strobo(buf,r)

if low == "0": result += 1

return result

**247. Strobogrammatic Number II**

Medium

26285FavoriteShare

A strobogrammatic number is a number that looks the same when rotated 180 degrees (looked at upside down).

Find all strobogrammatic numbers that are of length = n.

**Example:**

**Input:** n = 2

**Output:** ["11","69","88","96"]

class Solution:

def findStrobogrammatic(self, n: int) -> List[str]:

if n == 0: return []

if n == 1: return ["0","1","8"]

result = [""]

end = [""]

start = True

for i in range(n//2):

temp = []

end\_temp = []

for i in range(len(result)):

x = result[i]

z = end[i]

if start:

for y in ["1","6","8","9"]:

temp.append(x+y)

if y == "6":end\_temp.append("9"+z)

elif y == "9":end\_temp.append("6"+z)

else:end\_temp.append(y+z)

start = False

else:

for y in ["0","1","6","8","9"]:

temp.append(x+y)

if y == "6":end\_temp.append("9"+z)

elif y == "9":end\_temp.append("6"+z)

else:end\_temp.append(y+z)

result = temp

end = end\_temp

"""for i in range(len(end)):

end[i] = reversed(end[i])"""

print(result)

print(end)

ass = []

if n%2:

for i in range(len(result)):

for y in ["0","1","8"]:

ass.append(result[i]+y+end[i])

else:

for i in range(len(result)):

ass.append(result[i]+end[i])

return ass

**246. Strobogrammatic Number**

Easy

140339FavoriteShare

A strobogrammatic number is a number that looks the same when rotated 180 degrees (looked at upside down).

Write a function to determine if a number is strobogrammatic. The number is represented as a string.

**Example 1:**

**Input:** "69"

**Output:** true

**Example 2:**

**Input:** "88"

**Output:** true

**Example 3:**

**Input:** "962"

**Output:** false

class Solution:

def isStrobogrammatic(self, num: str) -> bool:

ass = ["0","1","8"]

fuck = ["0","1","6","8","9"]

length = len(num)

if length == 1: return num in ass

if length%2:

mid = length//2

if num[mid] not in ass: return False

left = mid-1

right = mid+1

else:

left = length//2-1

right = length//2

while left >= 0:

#print (num[left],num[right])

if num[left] not in fuck or num[right] not in fuck: return False

if num[left] == "6":

if num[right] != "9": return False

else:

left -= 1

right += 1

continue

if num[left] == "9":

if num[right] != "6": return False

else:

left -= 1

right += 1

continue

if num[left] != num[right]:

return False

left -= 1

right += 1

return True

**163. Missing Ranges**

Medium

2291340FavoriteShare

Given a sorted integer array ***nums***, where the range of elements are in the **inclusive range [*lower*, *upper*]**, return its missing ranges.

**Example:**

**Input:** ***nums*** = [0, 1, 3, 50, 75], ***lower*** = 0 and ***upper*** = 99,

**Output:** ["2", "4->49", "51->74", "76->99"]

class Solution:

def findMissingRanges(self, nums: List[int], lower: int, upper: int) -> List[str]:

if nums == []:

if lower == upper: return[str(lower)]

return[str(lower)+"->"+str(upper)]

prev = lower-1

result = []

for x in nums:

print(prev,x)

if x == prev+2:

result.append(str(prev+1))

elif x > prev+2:

result.append(str(prev+1)+"->"+str(x-1))

prev = x

if x == upper-1: result.append(str(upper))

elif x < upper -1: result.append(str(x+1)+"->"+str(upper))

return result

**159. Longest Substring with At Most Two Distinct Characters**

Hard

60312FavoriteShare

Given a string ***s*** , find the length of the longest substring ***t***that contains **at most**2 distinct characters.

**Example 1:**

**Input:** "eceba"

**Output:** 3

**Explanation: *t***is "ece" which its length is 3.

**Example 2:**

**Input:** "ccaabbb"

**Output:** 5

**Explanation: *t***is "aabbb" which its length is 5.

class Solution:

def lengthOfLongestSubstringTwoDistinct(self, s: str) -> int:

result = 0

if len(s)<2:return len(s)

dic = {}

start = 0

for i in range(len(s)):

if len(dic) < 2:

if s[i] in dic: dic[s[i]] += 1

else: dic[s[i]] = 1

elif s[i] in dic: dic[s[i]] += 1

else:

for j in range(start,i):

dic[s[j]] -= 1

if dic[s[j]] == 0:

del dic[s[j]]

start = j+1

break

dic[s[i]] = 1

result = max(result,sum(dic.values()))

return result

**158. Read N Characters Given Read4 II - Call multiple times**

Hard

318814FavoriteShare

Given a file and assume that you can only read the file using a given method read4, implement a method read to read *n* characters. **Your method read may be called multiple times.**

**Method read4:**

The API read4 reads 4 consecutive characters from the file, then writes those characters into the buffer array buf.

The return value is the number of actual characters read.

Note that read4() has its own file pointer, much like FILE \*fp in C.

**Definition of read4:**

Parameter: char[] buf

Returns: int

Note: buf[] is destination not source, the results from read4 will be copied to buf[]

Below is a high level example of how read4 works:

File file("abcdefghijk"); // File is "abcdefghijk", initially file pointer (fp) points to 'a'

char[] buf = new char[4]; // Create buffer with enough space to store characters

read4(buf); // read4 returns 4. Now buf = "abcd", fp points to 'e'

read4(buf); // read4 returns 4. Now buf = "efgh", fp points to 'i'

read4(buf); // read4 returns 3. Now buf = "ijk", fp points to end of file

**Method read:**

By using the read4 method, implement the method read that reads *n* characters from the file and store it in the buffer array buf. Consider that you **cannot** manipulate the file directly.

The return value is the number of actual characters read.

**Definition of read:**

Parameters: char[] buf, int n

Returns: int

Note: buf[] is destination not source, you will need to write the results to buf[]

**Example 1:**

File file("abc");

Solution sol;

// Assume buf is allocated and guaranteed to have enough space for storing all characters from the file.

sol.read(buf, 1); // After calling your read method, buf should contain "a". We read a total of 1 character from the file, so return 1.

sol.read(buf, 2); // Now buf should contain "bc". We read a total of 2 characters from the file, so return 2.

sol.read(buf, 1); // We have reached the end of file, no more characters can be read. So return 0.

**Example 2:**

File file("abc");

Solution sol;

sol.read(buf, 4); // After calling your read method, buf should contain "abc". We read a total of 3 characters from the file, so return 3.

sol.read(buf, 1); // We have reached the end of file, no more characters can be read. So return 0.

**Note:**

1. Consider that you **cannot** manipulate the file directly, the file is only accesible for read4 but **not** for read.
2. The read function may be called **multiple times**.
3. Please remember to **RESET** your class variables declared in Solution, as static/class variables are **persisted across multiple test cases**. Please see [here](https://leetcode.com/faq/) for more details.
4. You may assume the destination buffer array, buf, is guaranteed to have enough space for storing *n* characters.
5. It is guaranteed that in a given test case the same buffer buf is called by read.

"""

The read4 API is already defined for you.

@param buf, a list of characters

@return an integer

def read4(buf):

# Below is an example of how the read4 API can be called.

file = File("abcdefghijk") # File is "abcdefghijk", initially file pointer (fp) points to 'a'

buf = [' '] \* 4 # Create buffer with enough space to store characters

read4(buf) # read4 returns 4. Now buf = ['a','b','c','d'], fp points to 'e'

read4(buf) # read4 returns 4. Now buf = ['e','f','g','h'], fp points to 'i'

read4(buf) # read4 returns 3. Now buf = ['i','j','k',...], fp points to end of file

"""

class Solution:

def \_\_init\_\_(self):

self.n\_left = 0

self.buffer = [""]

def read(self, buf, n):

"""

:type buf: Destination buffer (List[str])

:type n: Number of characters to read (int)

:rtype: The number of actual characters read (int)

"""

#print(self.n\_left,self.buffer)

if n <= self.n\_left:

self.n\_left -= n

for i in range(n):

buf[i] = self.buffer[i]

self.buffer = self.buffer[n:]

return n

n\_total = self.n\_left

for i in range(n\_total):

buf[i] = self.buffer[i]

while n\_total < n:

temp = [""]\*4

n\_actual = read4(temp)

if n\_actual == 0: break

for i in range(n\_total,n\_total+n\_actual):

buf[i] = temp.pop(0)

n\_total += n\_actual

if n\_total < n:

self.n\_left = 0

self.buffer = [""]

return n\_total

self.n\_left = n\_total - n

#print(n\_read,n\_actual,n\_read - n)

#print(buf)

self.buffer = buf[n:n\_total]

return n