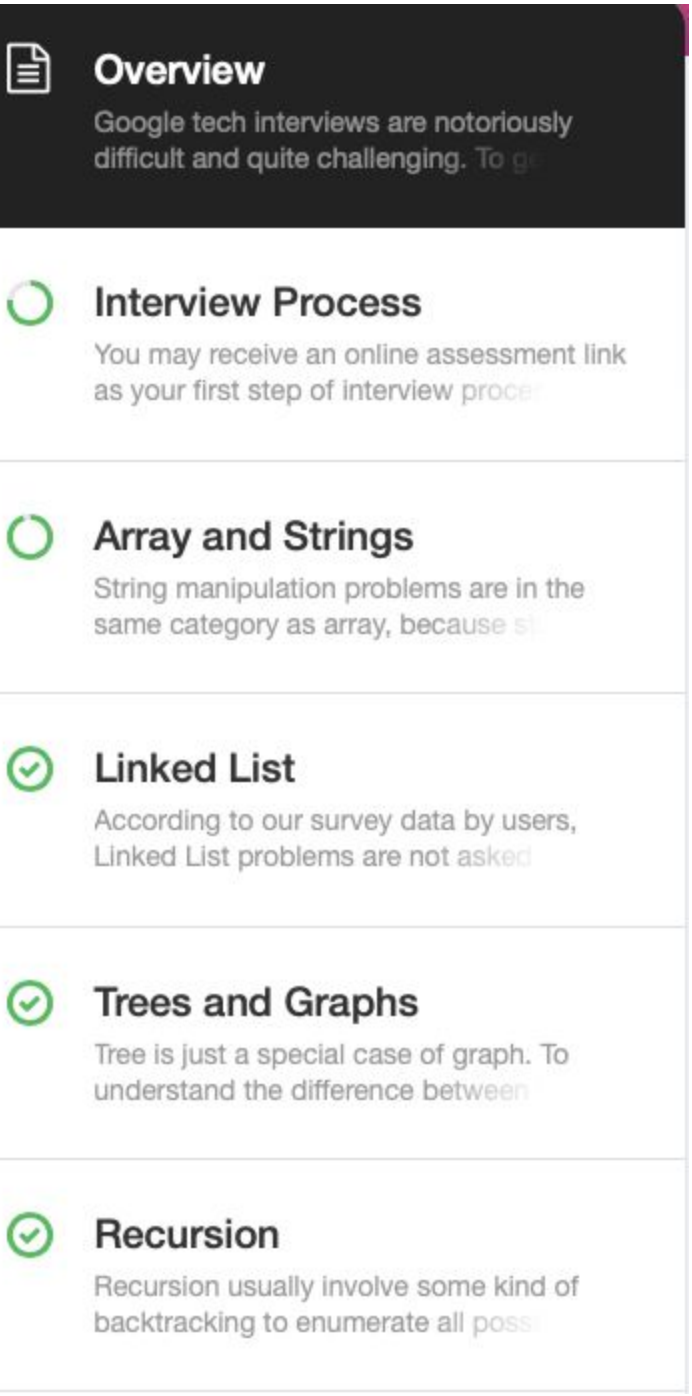
Introduction

Google tech interviews are notoriously difficult and quite challenging. To get a phone screen, you will submit your resume to their online application system or via an internal referral from a Googler.

Assuming you passed their resume screen, a recruiter will reach out to you. Usually there will be two phone screens, and if you do well you'll be invited to onsite interviews.

Since Google operates at a large scale, be prepared to answer lots of follow up questions on how to scale the algorithm you wrote for multiple machines. Some examples are: Number of Islands, Intersection of Two Arrays II.


Overview
Google tech interviews are notoriously difficult and quite challenging. To g
Interview Process
You may receive an online assessment link as your first step of interview proce
Array and Strings
String manipulation problems are in the same category as array, because s
Linked List
According to our survey data by users, Linked List problems are not asked
Trees and Graphs
Tree is just a special case of graph. To understand the difference between
Recursion
Recursion usually involve some kind of backtracking to enumerate all poss

Sorting and Searching

Interval related problems are quite often asked at Google interviews. Similar

Dynamic Programming

It can be tricky to identify the subproblems and connecting them, which is esse

Design

Google loves to ask lots of question variations based on the Iterator pat

Others

Here are other type of problems you may encounter in a Google interview, su

Discuss

0 topics - share ideas and ask questions about this card

Introduction

Google tech interviews are notoriously difficult and quite challenging. To get a phone screen, you will submit your resume to their online application system or via an internal referral from a Googler.

Assuming you passed their resume screen, a recruiter will reach out to you. Usually there will be two phone screens, and if you do well you'll be invited to onsite interviews.

Since Google operates at a large scale, be prepared to answer lots of follow up questions on how to scale the algorithm you wrote for multiple machines. Some examples are: Number of Islands, Intersection of Two Arrays II.

Interview Process

☑ Repeated String Match	☑ K Empty Slots
☐ Next Closest Time	☐ Longest Univalue Path
☑ License Key Formatting	☑ Google Phone Interview
☑ Google Onsite Interview	☑ Google Hiring Committee
☑ Google Offer Review	

Array and Strings

Spiral Matrix	Plus One
Trapping Rain Water	Longest Substring with At Most K Dis...
Add Bold Tag in String	Game of Life
Read N Characters Given Read4	Read N Characters Given Read4 II - C...
One Edit Distance	Valid Palindrome
Valid Number	Valid Parentheses
Image Smoother	Intersection of Two Arrays
Max Consecutive Ones	Max Consecutive Ones II
Shortest Palindrome	First Missing Positive
First Unique Character in a String	Move Zeroes
Remove Duplicates from Sorted Array	

Linked List

- Merge k Sorted Lists
- Insert into a Cyclic Sorted List

Trees and Graphs

- Evaluate Division
- Inorder Successor in BST
- Robot Room Cleaner
- Redundant Connection II
- Course Schedule
- Validate Binary Search Tree
- Closest Binary Search Tree Value

Recursion

- Word Squares
- Strobogrammatic Number II
- Word Search II
- Android Unlock Patterns

Sorting and Searching

- [x] Minimum Window Substring
- [x] Kth Largest Element in an Array
- [x] Shortest Distance from All Buildings
- [x] Find K-th Smallest Pair Distance
- [x] Find K Pairs with Smallest Sums
- [ ] Range Module
- [x] Sqrt(x)
- [x] Insert Interval
- [x] Sort Transformed Array
- [x] Merge Intervals
- [x] Longest Palindromic Substring
- [x] Diagonal Traverse
- [x] Next Greater Element I
- [x] Pacific Atlantic Water Flow
- [x] Evaluate Reverse Polish Notation

Dynamic Programming

- [x] Decode Ways
- [x] Word Break
- [x] Sentence Screen Fitting
- [x] Maximum Vacation Days
- [x] Edit Distance
- [x] Minimum Path Sum
- [x] House Robber II

Design

- [x] Moving Average from Data Stream
- [ ] Peeking Iterator
- [x] Binary Search Tree Iterator
- [ ] Zigzag Iterator
- [ ] Design Tic-Tac-Toe
- [ ] Range Sum Query 2D - Mutable

Others

- [ ] UTF-8 Validation
- [ ] Maximum Product of Word Lengths

Interview Process

You may receive an online assessment link as your first step of interview process. The assessment will expire within 7 days and contains two coding questions to be completed within an hour. Below are some Online Assessment questions for you to practice.

Near the end of this chapter we provide more details of the different stages of a Google interview.

- [x] **Repeated String Match**
  Online Assessment Question
- [x] **K Empty Slots**
  Online Assessment Question
- [ ] **Next Closest Time**
  Online Assessment Question
- [ ] **Longest Univalue Path**
  Online Assessment Question
- [x] **License Key Formatting**
  Online Assessment Question
- [x] **Google Phone Interview**
  Basics of Google Phone Interview
- [x] **Google Onsite Interview**
  Basics of Google Onsite Interview
- [x] **Google Hiring Committee**
  Basics of Google Hiring Committee stage
- [x] **Google Offer Review**
  Basics of Google Offer Review process

Repeated String Match

Go to Discuss

Given two strings A and B, find the minimum number of times A has to be repeated such that B is a substring of it. If no such solution, return -1.

For example, with A = "abcd" and B = "cdabcdab".

Return 3, because by repeating A three times (“abcdabcdabcd”), B is a substring of it; and B is not a substring of A repeated two times ("abcdabcd").

Note:
The length of `A` and `B` will be between 1 and 10000.

Python

```python
class Solution:
    def repeatedStringMatch(self, A, B):
        """
        :type A: str
        :type B: str
        :rtype: int
        """
        i = 1
        a = A
        while B not in A and len(A)-len(a) < len(B):
            A = A + a
            i = i+1
        return i if B in A else -1
```

K Empty Slots

Go to Discuss

There is a garden with `N` slots. In each slot, there is a flower. The `N` flowers will bloom one by one in `N` days. In each day, there will be `exactly` one flower blooming and it will be in the status of blooming since then.

Given an array `flowers` consists of number from `1` to `N`. Each number in the array represents the place where the flower will open in that day.

For example, `flowers[i] = x` means that the unique flower that blooms at day `i` will be at position `x`, where `i` and `x` will be in the range from `1` to `N`.

Also given an integer `k`, you need to output in which day there exists two flowers in the status of blooming, and also the number of flowers between them is `k` and these flowers are not blooming.

If there isn't such day, output -1.

Example 1:

```
Input:
flowers: [1,3,2]
k: 1
Output: 2
Explanation: In the second day, the first and the third flower have become blooming.
```

Example 2:

```
Input:
flowers: [1,2,3]
k: 1
Output: -1
```

Note:

1. The given array will be in the range [1, 20000].

```python
class Solution:
    def kEmptySlots(self, flowers, k):
        """
        :type flowers: List[int]
        :type k: int
        :rtype: int
        """
        N = len(flowers)

        slot_blooming_day = [0] * N
        for day, slot in enumerate(flowers, 1):
            slot_blooming_day[slot-1] = day

        left, right = 0, k + 1
        ans = N + 1
        i = 0
        while right < N:
            if slot_blooming_day[i] < slot_blooming_day[left] or slot_blooming_day[i] <= slot_blooming_day[right]:
                if i == right:
                    ans = min(ans, max(slot_blooming_day[left], slot_blooming_day[right]))
                left, right = i, i + k + 1
            i += 1
        return ans if ans <= N else -1
```

Next Closest Time

Go to Discuss

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, beca
use 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.
```

Solution 1

Just turn the clock forwards one minute at a time until you reach a time with the original digits.

```
from datetime import *

class Solution(object):
    def nextClosestTime(self, time):
        digits = set(time)
        while True:
            time = (datetime.strptime(time, '%H:%M') + timedelta(minutes=1)).strftime('%H:%M')
            if set(time) <= digits:
                return time
```

Solution 2

Return the smallest time that uses the given digits, just make being larger than the input a priority.

```
def nextClosestTime(self, time):
    return min((t <= time, t)
               for i in range(24 * 60)
               for t in ['%02d:%02d' % divmod(i, 60)]
               if set(t) <= set(time))[1]
```

Longest Univalue Path

Go to Discuss

Given a binary tree, find the length of the longest path where each node in the path has the same value. This path may or may not pass through the root.

Note: The length of path between two nodes is represented by the number of edges between them.

Example 1:

Input:

```
          5
         / \
        4   5
       / \   \
      1   1   5
```

Output:

```
2
```

Example 2:

Input:

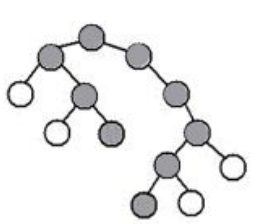
```
          1
         / \
        4   5
       / \   \
      4   4   5
```

Output:

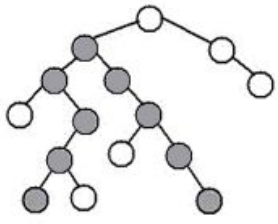
```
2
```

The approach is similar to the Diameter of Binary Tree question except that we reset the left/right to 0 whenever the current node does not match the children node value.

In the Diameter of Binary Tree question, the path can either go through the root or it doesn't.



diameter, 9 nodes, through root



diameter, 9 nodes, NOT through root

Hence at the end of each recursive loop, return the longest length using that node as the root so that the node's parent can potentially use it in its longest path computation.

We also use an external variable `longest` that keeps track of the longest path seen so far.

```python
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution(object):
    def longestUnivaluePath(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        # Time: O(n)
        # Space: O(n)
        longest = [0]
        def traverse(node):
            if not node:
                return 0
            left_len, right_len = traverse(node.left), traverse(node.right)
            left = (left_len + 1) if node.left and node.left.val == node.val else 0
            right = (right_len + 1) if node.right and node.right.val == node.val else 0
            longest[0] = max(longest[0], left + right)
            return max(left, right)
        traverse(root)
        return longest[0]
```

License Key Formatting

Go to Discuss

You are given a license key represented as a string S which consists only alphanumeric character and dashes. The string is separated into N+1 groups by N dashes.

Given a number K, we would want to reformat the strings such that each group contains *exactly* K characters, except for the first group which could be shorter than K, but still must contain at least one character. Furthermore, there must be a dash inserted between two groups and all lowercase letters should be converted to uppercase.

Given a non-empty string S and a number K, format the string according to the rules described above.

Example 1:

```
Input: S = "5F3Z-2e-9-w", K = 4

Output: "5F3Z-2E9W"

Explanation: The string S has been split into two parts, each part has 4 characters.
Note that the two extra dashes are not needed and can be removed.
```

Example 2:

```
Input: S = "2-5g-3-J", K = 2

Output: "2-5G-3J"

Explanation: The string S has been split into three parts, each part has 2 characters except the first part as it could be shorter
 as mentioned above.
```

Note:

1. The length of string S will not exceed 12,000, and K is a positive integer.
2. String S consists only of alphanumerical characters (a-z and/or A-Z and/or 0-9) and dashes(-).
3. String S is non-empty.

Python

```python
class Solution(object):
    def licenseKeyFormatting(self, S, K):
        S = S.replace('-','').upper()
        r = len(S) % K
        pattern = re.compile(r'[A-Z0-9]{%d}' % K)
        matched = re.findall(pattern, S[r:])
        if not r:
            return "-".join(matched)
        return "-".join([S[:r]] + matched)
```

A Google Phone Interview

The phone screen usually last between 30 and 60 minutes.

Your phone interview will cover data structures and algorithms. Be prepared to write around 20-30 lines of code in your strongest language. This should be clean, rich, robust code.

Your interviewer will ask one or two coding questions with lots of follow up questions. Be prepared to talk through your thought process while writing code in a Google Doc that you'll share with your interviewer. We recommend using a hands-free headset or speakerphone so you can type freely.

1. You will be asked an open ended question. Ask clarifying questions, devise requirements.
2. You will be asked to explain it in an algorithm.
3. Convert it to a workable code.
   (Hint: Don't worry about getting it perfect because time is limited. Write what comes but then refine it later. Also make sure you consider corner cases and edge cases, production ready.)
4. Optimize the code, follow it with test cases and find any bugs.

A Google Onsite Interview

This is it! You've passed the phone interviews and now you have received an onsite interview invitation. Onsite interview is the most important process of a Google interview, and your ability to receive an offer will largely depend on your performance during onsite interviews. Traditionally, the percentage of candidates who will receive an offer after onsite interview is about 1 of 5 to 1 of 7.

You'll usually meet with five Googlers for about 30 to 45 minutes each, and one of them will be your lunch interviewer who does not submit interview feedback, so take this opportunity to freely ask questions.

All candidates will have the chance to highlight strengths in four different areas:

- General cognitive ability: Open-ended questions are asked to learn how you approach and solve problems. And there's no one right answer—your ability to explain your thought process and how you use data to inform decisions is what's most important.
- Leadership: Be prepared to discuss how you have used your communication and decision-making skills to mobilize others. This might be by stepping up to a leadership role at work or with an organization, or by helping a team succeed even when you weren't officially the leader.
- Role-related knowledge: Google is interested in how your individual strengths combine with your experience to drive impact. Google don't just look for how you can contribute today, but how you can grow into different roles—including ones that haven't even been invented yet.
- Googleyness: Share how you work individually and on a team, how you help others, how you navigate ambiguity, and how you push yourself to grow outside of your comfort zone.

Note that Google put heavy emphasis on your analytical ability *regardless* of your job experience. Be prepared to showcase solid understanding of data structures and algorithms. If you have years of experiences in the industry and are rusty in your Computer Science fundamentals, invest more time in brushing up your basic CS fundamental skills. Otherwise, you may end up being rejected like Max Howell for not able to invert a binary tree.

Google wants to understand your coding skills and technical areas of expertise, including tools or programming languages and general knowledge on topics like data structures and algorithms. There's generally some back and forth in these discussions, just like there is on the job to push each other's thinking and learn about different approaches. So be prepared to talk through your solutions in depth. Push your own boundaries and find the best answer—that's probably how you work anyway.

Technical onsite interviews at Google were historically conducted on whiteboards, but to provide a more authentic coding experience that's less time-consuming, Google has started to offer laptops for coding interviews in some sites. These chromebooks have an interview app that let's you choose a coding language of your preference.

Throughout the interview process, feel free to ask your interviewers for clarification to make sure you fully understand their questions.

A Google Hiring Committee

After the onsite interview, each of your interviewers (except the lunch interviewer) will have to submit detailed interview feedback. Each interviewer's feedback is kept hidden from other interviewers to ensure one's feedback does not influence the others. Each interviewer will assign a score and make a Hire / No Hire decision.

Assume the scores from the feedback are good enough, your whole packet will be moved forward to the hiring committee for review.

The hiring committee will review all submitted feedback thoroughly and make a hire / no hire recommendation. For more details about this process, you can read here.

A Google Offer Review

If Hiring Committee recommends hire, congratulations! You have arrived at the final stage of the process. There are about 10-12% candidates not extended offers at this stage. Recruiter requests for External References and Compensation History, conducts reference checks, and prepares a packet for the Offer Review Process.

The Offer Review process:

1. Pre-Review Committee
2. Compensation Committee Review
3. Senior VP Review

There is a small percentage chance of the offer not getting approved at the final SVP review.

Array and Strings

String manipulation problems are in the same category as array, because string is represented as an array of characters internally. Array problems usually does not require knowledge of advanced data structures, so just basic data structures such as Hash Table and basic techniques like Two Pointers should suffice.

Google likes to test your ability to think at large scale by asking variation of problems represented in data stream model. For example, instead of giving you an integer array, you are given a stream of integers and all integers are too large to be fit in memory. A great example of such problem which can be represented in data stream model is Longest Substring with At Most K Distinct Characters.

Spiral Matrix

Go to Discuss

Given a matrix of *m* x *n* elements (*m* rows, *n* columns), return all elements of the matrix in spiral order.

Example 1:

```
Input:
[
 [ 1, 2, 3 ],
 [ 4, 5, 6 ],
 [ 7, 8, 9 ]
]
Output: [1,2,3,6,9,8,7,4,5]
```

Example 2:

```
Input:
[
  [1, 2, 3, 4],
  [5, 6, 7, 8],
  [9,10,11,12]
]
Output: [1,2,3,4,8,12,11,10,9,5,6,7]
```

Python3

```python
class Solution:
    def spiralOrder(self, matrix):
        """
        :type matrix: List[List[int]]
        :rtype: List[int]
        """
        ret = []
        while matrix:
            ret += matrix.pop(0)
            if matrix and matrix [0]:
                for row in matrix:
                    ret.append(row.pop())
            if matrix:
                ret += matrix.pop()[::-1]
            if matrix and matrix[0]:
                for row in matrix[::-1]:
                    ret.append(row.pop(0))
        return ret

```

Plus One

Go to Discuss

Given a **non-empty** array of digits representing a non-negative integer, plus one to the integer.

The digits are stored such that the most significant digit is at the head of the list, and each element in the array contain a single digit.

You may assume the integer does not contain any leading zero, except the number 0 itself.

Example 1:

```
Input: [1,2,3]
Output: [1,2,4]
Explanation: The array represents the integer 123.
```

Example 2:

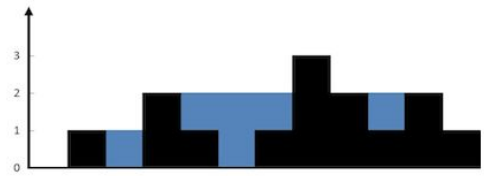
```
Input: [4,3,2,1]
Output: [4,3,2,2]
Explanation: The array represents the integer 4321.
```

```python
class Solution(object):
    def plusOne(self, digits):
        """
        :type digits: List[int]
        :rtype: List[int]
        """
        if len(digits) == 0:
            digits = [1]
        elif digits[-1] == 9:
            digits = self.plusOne(digits[:-1])
            digits.extend([0])
        else:
            digits[-1] += 1
        return digits
```

Trapping Rain Water

Go to Discuss

Given *n* non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.



The above elevation map is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped. **Thanks Marcos** for contributing this image!

Example:

```
Input: [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
```

```python
class Solution(object):
    def trap(self, nums):
        """
        :type height: List[int]
        :rtype: int
        """
        if not nums:
            return 0

        left = [-1]*len(nums)

        left[0] = nums[0]

        for i in range(1, len(nums)):
            left[i] = max(left[i-1], nums[i])

        right = nums[-1]
        ans = 0
        #print(left)

        for i in range(len(nums)-2, -1, -1):
            mini = min(left[i], right)
            #print(mini)
            if mini > nums[i]:
                ans += mini - nums[i]
            right = max(right, nums[i])

        return ans
```

Longest Substring with At Most K Distinct Characters

Go to Discuss

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.
```

```python
from collections import defaultdict
class Solution(object):
    def lengthOfLongestSubstringKDistinct(self, s, k):
        """
        :type s: str
        :type k: int
        :rtype: int
        """
        if k == 0:
            return 0
        start, end, max_w, freq = 0, 0, 0, defaultdict(set)
        for end in range(len(s)):
            freq[s[end]].add(end)
            if len(freq) <= k:
                max_w = max(max_w, end-start+1)
            else:
                while start < end and len(freq) > k:
                    freq[s[start]].remove(start)
                    if len(freq[s[start]]) == 0:
                        del freq[s[start]]
                    start += 1
        return max_w
```

Add Bold Tag in String

Go to Discuss

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].

Two Pointer Pattern

- Use two pointers start and end. Move end to right till you get a valid solution. Then move start to right till you invalidate the solution. Do the necessary book-keeping.

```python
class Solution:
    def addBoldTag(self, s, dict):
        """
        :type s: str
        :type dict: List[str]
        :rtype: str
        """
        status = [False]*len(s)
        final = ""
        for word in dict:
            start = s.find(word)
            last = len(word)
            while start != -1:
                for i in range(start, last+start):
                    status[i] = True
                start = s.find(word,start+1)
        i = 0
        i = 0
        while i < len(s):
            if status[i]:
                final += "<b>"
                while i < len(s) and status[i]:
                    final += s[i]
                    i += 1
                final += "</b>"
            else:
                final += s[i]
                i += 1
        return final
```

Game of Life

Go to Discuss

According to the Wikipedia's article: "The **Game of Life**, also known simply as **Life**, is a cellular automaton devised by the British mathematician John Horton Conway in 1970."

Given a *board* with *m* by *n* cells, each cell has an initial state *live* (1) or *dead* (0). Each cell interacts with its eight neighbors (horizontal, vertical, diagonal) using the following four rules (taken from the above Wikipedia article):

1. Any live cell with fewer than two live neighbors dies, as if caused by under-population.
2. Any live cell with two or three live neighbors lives on to the next generation.
3. Any live cell with more than three live neighbors dies, as if by over-population..
4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

Write a function to compute the next state (after one update) of the board given its current state. The next state is created by applying the above rules simultaneously to every cell in the current state, where births and deaths occur simultaneously.

Example:

Example:

```
Input:
[
  [0,1,0],
  [0,0,1],
  [1,1,1],
  [0,0,0]
]
Output:
[
  [0,0,0],
  [1,0,1],
  [0,1,1],
  [0,1,0]
]
```

Follow up:

1. Could you solve it in-place? Remember that the board needs to be updated at the same time: You cannot update some cells first and then use their updated values to update other cells.
2. In this question, we represent the board using a 2D array. In principle, the board is infinite, which would cause problems when the active area encroaches the border of the array. How would you address these problems?

the key point is to understand that the change to a cell is only decided by its nearby 8 cell in the original grid.

we should not use the updated cell to compute to change decision for other cell,therefore, the brute force way is to store the result in a new grid,then assign result back

But usually it requires use O(1) space, so the problem becomes: How can we store store the middle result without use extra space.

the solution is to store the result in the origin grid as different number by some rule, so when we compute decision for other cell, we can know the original value of those nearby cell which has already been updated based on the rule

for example, we can do like this

living cells nearby | change | new value

for example, we can do like this

living cells nearby | change | new value

```
<2          1->0      2
2,3         1->1      1
>3          1->0      2
3           0->1      3
```

so when we count living cells nearby, we need to count those value equals to 1 and 2

```python
class Solution(object):
    def gameOfLife(self, board):
        if not board or len(board[0]) == 0:
            return
        m, n = len(board), len(board[0])
        for i, row in enumerate(board):
            for j, ele in enumerate(row):
                count = 0
                for a in xrange(max(0, i - 1), min(i + 2, m)):
                    for b in xrange(max(0, j - 1), min(j + 2, n)):
                        if (a, b) != (i, j) and 1 <= board[a][b] <= 2:
                            count += 1
                if board[i][j] == 0:
                    if count == 3:
                        board[i][j] = 3
                else:
                    if count < 2 or count > 3:
                        board[i][j] = 2
        for i in xrange(m):
            for j in xrange(n):
                if board[i][j] == 2:
                    board[i][j] = 0
                elif board[i][j] == 3:
                    board[i][j] = 1
```

Read N Characters Given Read4

Go to Discuss

Given a file and assume that you can only read the file using a given method `read4`, implement a method to read *n* characters.

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:

```
Input: file = "abc", n = 4
Output: 3
Explanation: After calling your read method, buf should contain "abc". We read a total of 3 c
haracters from the file, so return 3. Note that "abc" is the file's content, not buf. buf is
the destination buffer that you will have to write the results to.
```

Example 2:

```
Input: file = "abcde", n = 5
Output: 5
Explanation: After calling your read method, buf should contain "abcde". We read a total of 5
 characters from the file, so return 5.
```

Example 3:

```
Input: file = "abcdABCD1234", n = 12
Output: 12
Explanation: After calling your read method, buf should contain "abcdABCD1234". We read a tot
al of 12 characters from the file, so return 12.
```

```python
class Solution(object):
    def read(self, buf, n):
        idx = 0
        while n > 0:
            # read file to buf4
            buf4 = [""]*4
            l = read4(buf4)
            # if no more char in file, return
            if not l:
                return idx
            # write buf4 into buf directly
            for i in range(min(l, n)):
                buf[idx] = buf4[i]
                idx += 1
                n -= 1
        return idx
```

One Edit Distance

Go to Discuss

Given two strings ***s*** and ***t***, determine if they are both one edit distance apart.

Note:

There are 3 possiblities to satisify one edit distance apart:

1. Insert a character into ***s*** to get ***t***
2. Delete a character from ***s*** to get ***t***
3. Replace a character of ***s*** to get ***t***

Example 1:

```
Input: s = "ab", t = "acb"
Output: true
Explanation: We can insert 'c' into s to get t.
```

Example 2:

```
Input: s = "cab", t = "ad"
Output: false
Explanation: We cannot get t from s by only one step.
```

```python
class Solution(object):
    def isOneEditDistance(self, s, t):
        if len(s) > len(t):
            return self.isOneEditDistance(t, s)
        if abs(len(s) - len(t)) > 1 or s == t:
            return False
        for i in range(len(s)):
            if s[i] != t[i]:
                return s[i+1:] == t[i+1:] or s[i:] == t[i+1:]
        return True
```

Valid Palindrome

Go to Discuss

Given a string, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.

Note: For the purpose of this problem, we define empty string as valid palindrome.

Example 1:

```
Input: "A man, a plan, a canal: Panama"
Output: true
```

Example 2:

```
Input: "race a car"
Output: false
```

```python
class Solution:
    def isPalindrome(self, s):
        charList = []
        for ch in s:
            if ch.isalnum():
                charList.append(ch.lower())
        return charList == charList[::-1]
```

Valid Number

Go to Discuss

Validate if a given string can be interpreted as a decimal number.

Some examples:

`"0"` => `true`

`" 0.1 "` => `true`

`"abc"` => `false`

`"1 a"` => `false`

`"2e10"` => `true`

`" -90e3   "` => `true`

`" 1e"` => `false`

`"e3"` => `false`

`" 6e-1"` => `true`

`" 99e2.5 "` => `false`

`"53.5e93"` => `true`

`" --6 "` => `false`

`"-+3"` => `false`

`"95a54e53"` => `false`

Note: It is intended for the problem statement to be ambiguous. You should gather all requirements up front before implementing one. However, here is a list of characters that can be in a valid decimal number:

- Numbers 0-9
- Exponent - "e"
- Positive/negative sign - "+"/"-"
- Decimal point - "."

Of course, the context of these characters also matters in the input.

Update (2015-02-10):

The signature of the `C++` function had been updated. If you still see your function signature accepts a `const char *` argument, please click the reload button to reset your code definition.

```python
class Solution(object):
    def isNumber(self, s):
        """
        :type s: str
        :rtype: bool
        """
        try: float(s)
        except ValueError: return False
        else: return True
```

try and except in Python

try() is used in **Error and Exception Handling**

There are two kinds of errors :

- **Syntax Error** : Also known as Parsing Errors, most basic. Arise when the Python parser is unable to understand a line of code.
- **Exception** : Errors which are detected during execution. eg – ZeroDivisionError.

List of Exception Errors :

- **IOError :** if file can't be opened
- **KeyboardInterrupt :** when an unrequired key is pressed by the user
- **ValueError :** when built-in function receives a wrong argument
- **EOFError :** if End-Of-File is hit without reading any data
- **ImportError :** if it is unable to find the module

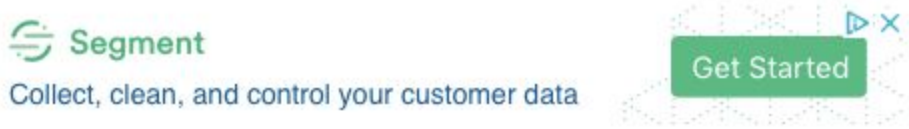
Now, here comes the task to handle these errors within our code in Python. So here we need **try-except** statements.

```
Basic Syntax :
 try:
    // Code
 except:
    // Code
```

How try() works?

- First **try** clause is executed i.e. the code between **try** and **except** clause.
- If there is no exception, then only **try** clause will run, **except** clause is finished.
- If any exception occured, **try** clause will be skipped and **except** clause will run.
- If any exception occurs, but the **except** clause within the code doesn't handle it, it is passed on to the outer **try** statements. If the exception left unhandled, then the execution stops.
- A **try** statement can have more than one **except** clause

Code 1 : No exception, so **try** clause will run.




```
# Python code to illustrate
# working of try()
def divide(x, y):
    try:
        # Floor Division : Gives only Fractional Part
        result = x // y
        print("Yeah ! Your answer is :", result)
    except ZeroDivisionError:
        print("Sorry ! You are dividing by zero ")

# Look at parameters and note the working of Program
divide(3, 2)
```

Output :

```
('Yeah ! Your answer is :', 1)
```

Code 1 : There is an exception so only **except** clause will run.

```
# Python code to illustrate
# working of try()
def divide(x, y):
    try:
        # Floor Division : Gives only Fractional Part
        result = x // y
        print("Yeah ! Your answer is :", result)
    except ZeroDivisionError:
        print("Sorry ! You are dividing by zero ")

# Look at parameters and note the working of Program
divide(3, 0)
```

Output :

```
Sorry ! You are dividing by zero
```

Valid Parentheses

Go to Discuss

Given a string containing just the characters `'('`, `')'`, `'{'`, `'}'`, `'['` and `']'`, determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.

Note that an empty string is also considered valid.

Example 1:

```
Input: "()"
Output: true
```

Example 2:

```
Input: "()[]{}"
Output: true
```

Example 3:

```
Input: "(]"
Output: false
```

```python
class Solution:
    def isValid(self, s):
        """
        :type s: str
        :rtype: bool
        """
        if not s:
            return True
        mapping = {']':'[','}':'{',')':'('}
        stack = [None]
        for i in s:
            if i in ['(','{','[']:
                stack.append(i)
            elif i in mapping:
                if mapping[i] == stack[-1]:
                    stack.pop()
                else:
                    stack.append(i)
            else:
                return False
        return len(stack) == 1
```

Image Smoother

Go to Discuss

Given a 2D integer matrix M representing the gray scale of an image, you need to design a smoother to make the gray scale of each cell becomes the average gray scale (rounding down) of all the 8 surrounding cells and itself. If a cell has less than 8 surrounding cells, then use as many as you can.

Example 1:

```
Input:
[[1,1,1],
 [1,0,1],
 [1,1,1]]
Output:
[[0, 0, 0],
 [0, 0, 0],
 [0, 0, 0]]
Explanation:
For the point (0,0), (0,2), (2,0), (2,2): floor(3/4) = floor(0.75) = 0
For the point (0,1), (1,0), (1,2), (2,1): floor(5/6) = floor(0.83333333) = 0
For the point (1,1): floor(8/9) = floor(0.88888889) = 0
```

Note:

1. The value in the given matrix is in the range of [0, 255].
2. The length and width of the given matrix are in the range of [1, 150].

```python
class Solution(object):
    def imageSmoother(self, M):
        """
        :type M: List[List[int]]
        :rtype: List[List[int]]
        """
        r, c = len(M), len(M[0])
        smooth = [[0]*c for i in range(r)]

        for i in range(r):
            for j in range(c):
                count = 0
                for nr in (i-1, i, i+1):
                    for nc in (j-1, j, j+1):
                        if 0 <= nr < r and 0 <= nc < c:
                            smooth[i][j] += M[nr][nc]
                            count += 1
                smooth[i][j] /= count
        return smooth
```

Intersection of Two Arrays

Go to Discuss

Given two arrays, write a function to compute their intersection.

Example 1:

```
Input: nums1 = [1,2,2,1], nums2 = [2,2]
Output: [2]
```

Example 2:

```
Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]
Output: [9,4]
```

Note:

- Each element in the result must be unique.
- The result can be in any order.

```python
class Solution(object):
    def intersection(self, nums1, nums2):
        """
        :type nums1: List[int]
        :type nums2: List[int]
        :rtype: List[int]
        """
        nums1_set = set(nums1)
        result_set = set([])
        for x in nums2:
            if x in nums1_set and x not in result_set:
                result_set.add(x)
        return [x for x in result_set]
```

Max Consecutive Ones

Go to Discuss

Given a binary array, find the maximum number of consecutive 1s in this array.

Example 1:

```
Input: [1,1,0,1,1,1]
Output: 3
Explanation: The first two digits or the last three digits are consecutive 1s.
    The maximum number of consecutive 1s is 3.
```

Note:

- The input array will only contain `0` and `1`.
- The length of input array is a positive integer and will not exceed 10,000

```python
class Solution(object):
    def findMaxConsecutiveOnes(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if not nums:
            return None

        counts = []
        temp = 0

        for num in nums:
            if num == 1:
                temp += 1
            else:
                counts.append(temp)
                temp = 0

        counts.append(temp)

        return max(counts)
```

Max Consecutive Ones II

Go to Discuss

Given a binary array, find the maximum number of consecutive 1s in this array if you can flip at most one 0.

Example 1:

```
Input: [1,0,1,1,0]
Output: 4
Explanation: Flip the first zero will get the the maximum number of consecutive 1s.
    After flipping, the maximum number of consecutive 1s is 4.
```

Note:

- The input array will only contain `0` and `1`.
- The length of input array is a positive integer and will not exceed 10,000

Follow up:

What if the input numbers come in one by one as an **infinite stream**? In other words, you can't store all numbers coming from the stream as it's too large to hold in memory. Could you solve it efficiently?

```python
class Solution(object):
    def findMaxConsecutiveOnes(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        pre_sum = 0
        max_len = 0
        post_sum = 0


        for i in nums:
            if i ==1:
                post_sum = post_sum+i
            else:
                if pre_sum+post_sum >max_len:
                    max_len = pre_sum+post_sum

                pre_sum = post_sum+1
                post_sum = 0

        return max(max_len,pre_sum+post_sum)
```

```python
class Solution(object):
    def intersection(self, nums1, nums2):
        """
        :type nums1: List[int]
        :type nums2: List[int]
        :rtype: List[int]
        """
        nums1_set = set(nums1)
        result_set = set([])
        for x in nums2:
            if x in nums1_set and x not in result_set:
                result_set.add(x)
        return [x for x in result_set]
```

Max Consecutive Ones

Go to Discuss

Given a binary array, find the maximum number of consecutive 1s in this array.

Example 1:

```
Input: [1,1,0,1,1,1]
Output: 3
Explanation: The first two digits or the last three digits are consecutive 1s.
    The maximum number of consecutive 1s is 3.
```

Note:

- The input array will only contain `0` and `1`.
- The length of input array is a positive integer and will not exceed 10,000

```python
class Solution(object):
    def findMaxConsecutiveOnes(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if not nums:
            return None

        counts = []
        temp = 0

        for num in nums:
            if num == 1:
                temp += 1
            else:
                counts.append(temp)
                temp = 0

        counts.append(temp)

        return max(counts)
```

Max Consecutive Ones II

Go to Discuss

Given a binary array, find the maximum number of consecutive 1s in this array if you can flip at most one 0.

Example 1:

```
Input: [1,0,1,1,0]
Output: 4
Explanation: Flip the first zero will get the the maximum number of consecutive 1s.
    After flipping, the maximum number of consecutive 1s is 4.
```

Note:

- The input array will only contain `0` and `1`.
- The length of input array is a positive integer and will not exceed 10,000

Follow up:

What if the input numbers come in one by one as an **infinite stream**? In other words, you can't store all numbers coming from the stream as it's too large to hold in memory. Could you solve it efficiently?

```python
class Solution(object):
    def findMaxConsecutiveOnes(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        pre_sum = 0
        max_len = 0
        post_sum = 0


        for i in nums:
            if i ==1:
                post_sum = post_sum+i
            else:
                if pre_sum+post_sum >max_len:
                    max_len = pre_sum+post_sum

                pre_sum = post_sum+1
                post_sum = 0

        return max(max_len,pre_sum+post_sum)
```

Shortest Palindrome

Go to Discuss

Given a string ***s***, you are allowed to convert it to a palindrome by adding characters in front of it. Find and return the shortest palindrome you can find by performing this transformation.

Example 1:

```
Input: "aacecaaa"
Output: "aaacecaaa"
```

Example 2:

```
Input: "abcd"
Output: "dcbabcd"
```

```python
class Solution(object):
    def shortestPalindrome(self, s):
        """
        :type s: str
        :rtype: str
        """
        rev_s = s[::-1]
        if (s == rev_s):
            return rev_s
        else:
            l = len(s)
            i = 1
            while(rev_s[i:] != s[:l-i]):
                i += 1
            return rev_s[:i]+s
```

First Missing Positive

Go to Discuss

Given an unsorted integer array, find the smallest missing positive integer.

Example 1:

```
Input: [1,2,0]
Output: 3
```

Example 2:

```
Input: [3,4,-1,1]
Output: 2
```

Example 3:

```
Input: [7,8,9,11,12]
Output: 1
```

Note:

Your algorithm should run in *O*(*n*) time and uses constant extra space.

```python
class Solution(object):
    def firstMissingPositive(self, nums):
        nums.sort()
        count, ans = 0, 1
        while count < len(nums):
            if ans == nums[count]: ans += 1
            count += 1
        return ans
```

First Unique Character in a String

Go to Discuss

Given a string, find the first non-repeating character in it and return it's index. If it doesn't exist, return -1.

Examples:

```
s = "leetcode"
return 0.

s = "loveleetcode",
return 2.
```

Note: You may assume the string contain only lowercase letters.

```python
class Solution(object):
    def firstUniqChar(self, s):
        """
        :type s: str
        :rtype: int
        """
        for i in range(len(s)):
            c = s[i]
            if s.count(c) == 1:
                return i
        return -1
```

Move Zeroes

Go to Discuss

Given an array `nums`, write a function to move all `0`'s to the end of it while maintaining the relative order of the non-zero elements.

Example:

```
Input: [0,1,0,3,12]
Output: [1,3,12,0,0]
```

Note:

1. You must do this **in-place** without making a copy of the array.
2. Minimize the total number of operations.

```python
class Solution:
    def moveZeroes(self, nums):
        """
        :type nums: List[int]
        :rtype: void Do not return anything, modify nums in-place instead.
        """
        counter = 0
        while(0 in nums):
            nums.remove(0)
            counter+=1
        while(counter!=0):
            nums.append(0)
            counter-=1
```

Remove Duplicates from Sorted Array

Go to Discuss

Given a sorted array *nums*, remove the duplicates **in-place** such that each element appear only *once* and return the new length.

Do not allocate extra space for another array, you must do this by **modifying the input array in-place** with O(1) extra memory.

Example 1:

```
Given nums = [1,1,2],

Your function should return length = 2, with the first two elements of nums being 1 and 2 res
pectively.

It doesn't matter what you leave beyond the returned length.
```

Example 2:

```
Given nums = [0,0,1,1,1,2,2,3,3,4],

Your function should return length = 5, with the first five elements of nums being modified t
o 0, 1, 2, 3, and 4 respectively.

It doesn't matter what values are set beyond the returned length.
```

Clarification:

Confused why the returned value is an integer but your answer is an array?

Clarification:

Confused why the returned value is an integer but your answer is an array?

Note that the input array is passed in by **reference**, which means modification to the input array will be known to the caller as well.

Internally you can think of this:

```
// nums is passed in by reference. (i.e., without making a copy)
int len = removeDuplicates(nums);

// any modification to nums in your function would be known by the caller.
// using the length returned by your function, it prints the first len elements.
for (int i = 0; i < len; i++) {
    print(nums[i]);
}
```

```
class Solution:
    # @param a list of integers
    # @return an integer
    def removeDuplicates(self, A):
        if not A:
            return 0

        newTail = 0

        for i in range(1, len(A)):
            if A[i] != A[newTail]:
                newTail += 1
                A[newTail] = A[i]

        return newTail + 1
```

Linked List

According to our survey data by users, Linked List problems are not asked frequently at Google. Perhaps most linked list problems are not that complex and is harder to ask follow up questions and complexity analysis is usually pretty straightforward.

Nonetheless, we strongly recommend you to still practice classic Linked List interview questions such as: Linked List Cycle, Intersection of Two Linked Lists and Copy List with Random Pointers. These problems are really fun problems and they teach you how to think outside of the box.

Of course, Merge k Sorted Lists is one of our all-time favorite interview questions and Google seems to love this question as well. Make sure you understand how to analyze the time complexity! This is a must-asked follow up question for this problem.

Merge k Sorted Lists
Editor's choice: Frequently asked in Google phone interview.

Insert into a Cyclic Sorted List
Editor's choice: Frequently asked in Google onsite interview.

Merge k Sorted Lists

Go to Discuss

Merge *k* sorted linked lists and return it as one sorted list. Analyze and describe its complexity.

Example:

```
Input:
[
  1->4->5,
  1->3->4,
  2->6
]
Output: 1->1->2->3->4->4->5->6
```

```python
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution(object):
    def mergeKLists(self, lists):
        """
        :type lists: List[ListNode]
        :rtype: ListNode
        """
        if lists is None or len(lists) == 0:
            return None

        head = ListNode(None)
        hlist = []
        for listnode in lists:
            if listnode:
                heapq.heappush(hlist, (listnode.val, listnode))

        if len(hlist) > 0:
            curr = head
            while len(hlist) > 0:
                curr.next = heapq.heappop(hlist)[1]
                curr = curr.next
                if curr.next:
                    heapq.heappush(hlist, (curr.next.val, curr.next))

        return head.next
```

Insert into a Cyclic Sorted List

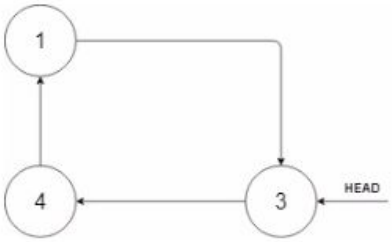
Go to Discuss

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.

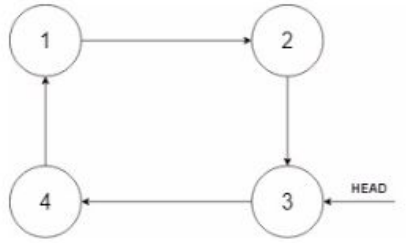
If there are multiple suitable places for insertion, you may choose any place to insert the new value. After the insertion, the cyclic list should remain sorted.

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.

```python
"""
# Definition for a Node.
class Node(object):
    def __init__(self, val, next):
        self.val = val
        self.next = next
"""
class Solution(object):
    def insert(self, head, insertVal):
        """
        :type head: Node
        :type insertVal: int
        :rtype: Node
        """

        new_node = Node(insertVal, head)

        if not head:
            return new_node

        node = head
        while True:
            if node.next.val < node.val and (insertVal <= node.next.val or insertVal >= node.val):
                break
            elif node.val <= insertVal <= node.next.val:
                break
            elif node.next == head:
                break
            node = node.next

        new_node.next = node.next
        node.next = new_node
        return head
```

Trees and Graphs

Tree is just a special case of graph. To understand the difference between trees and graphs, you can work on Graph Valid Tree.

Graphs are generally search breath first or depth first. The same applies to Tree.

Graphs are generally more complex than trees. Similarly, trees are generally more complex than a linear data structure such as array or linked list.

Prepping for your knowledge in Graphs is essential for Google interviews as you would most likely encounter a tree or graph question. A great way to brush up is to implement a tree or graph by coding it from scratch in Playground.

Evaluate Division

Go to Discuss

Equations are given in the format `A / B = k`, where `A` and `B` are variables represented as strings, and `k` is a real number (floating point number). Given some queries, return the answers. If the answer does not exist, return `-1.0`.

Example:
Given `a / b = 2.0, b / c = 3.0.`
queries are: `a / c = ?, b / a = ?, a / e = ?, a / a = ?, x / x = ? .`
return `[6.0, 0.5, -1.0, 1.0, -1.0 ].`

The input is: `vector<pair<string, string>> equations, vector<double>& values, vector<pair<string, string>> queries` , where `equations.size() == values.size()`, and the values are positive. This represents the equations. Return `vector<double>`.

According to the example above:

```
equations = [ ["a", "b"], ["b", "c"] ],
values = [2.0, 3.0],
queries = [ ["a", "c"], ["b", "a"], ["a", "e"], ["a", "a"], ["x", "x"] ].
```

The input is always valid. You may assume that evaluating the queries will result in no division by zero and there is no contradiction.

Although this looks like a math problem, we can easily model it with graph.

For example:
Given:
a/b = 2.0, b/c = 3.0
We can build a directed graph:
a -- 2.0 --> b -- 3.0 --> c
If we were asked to find a/c, we have:
a/c = a/b * b/c = 2.0 * 3.0
In the graph, it is the product of costs of edges.

Do notice that, 2 edges need to added into the graph with one given equation, because with a/b we also get result of b/a, which is the reciprocal of a/b.

so the previous example also gives edges:
c -- 0.333 --> b -- 0.5 --> a

Now we know how to model this problem, what we need to do is simply build the graph with given equations, and traverse the graph, either DFS or BFS, to find a path for a given query, and the result is the product of costs of edges on the path.

One optimization, which is not implemented in the code, is to "compress" paths for past queries, which will make future searches faster. This is the same idea used in compressing paths in union find set. So after a query is conducted and a result is found, we add two edges for this query if these edges are not already in the graph.

Given the number of variables N, and number of equations E,
building the graph takes O(E), each query takes O(N), space for graph takes O(E)

I think if we start to compress paths, the graph will grow to O(N^2), and we can optimize the query to O(1), please correct me if I'm wrong.

```python
class Solution(object):
    def calcEquation(self, equations, values, queries):

        graph = {}

        def build_graph(equations, values):
            def add_edge(f, t, value):
                if f in graph:
                    graph[f].append((t, value))
                else:
                    graph[f] = [(t, value)]

            for vertices, value in zip(equations, values):
                f, t = vertices
                add_edge(f, t, value)
                add_edge(t, f, 1/value)

        def find_path(query):
            b, e = query

            if b not in graph or e not in graph:
                return -1.0

            q = collections.deque([(b, 1.0)])
            visited = set()

            q = collections.deque([(b, 1.0)])
            visited = set()

            while q:
                front, cur_product = q.popleft()
                if front == e:
                    return cur_product
                visited.add(front)
                for neighbor, value in graph[front]:
                    if neighbor not in visited:
                        q.append((neighbor, cur_product*value))

            return -1.0

        build_graph(equations, values)
        return [find_path(q) for q in queries]
```

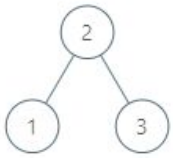
Inorder Successor in BST

Go to Discuss

Given a binary search tree and a node in it, find the in-order successor of that node in the BST.

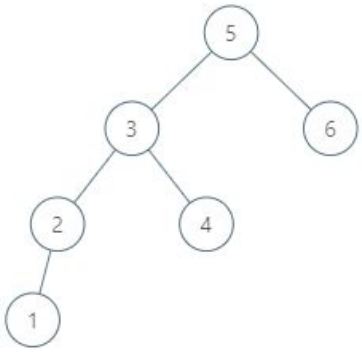
The successor of a node `p` is the node with the smallest key greater than `p.val`.

Example 1:




```
Input: root = [2,1,3], p = 1
Output: 2
Explanation: 1's in-order successor node is 2. Note that both p and the return value is of Tr
eeNode type.
```

Example 2:




```
Input: root = [5,3,6,2,4,null,null,1], p = 6
Output: null
Explanation: There is no in-order successor of the current node, so the answer is null.
```

Note:

1. If the given node has no in-order successor in the tree, return `null`.
2. It's guaranteed that the values of the tree are unique.

```python
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution(object):
    def inorderSuccessor(self, root, p):
        """
        :type root: TreeNode
        :type p: TreeNode
        :rtype: TreeNode
        """
        mn = None
        node = root
        while node:
            if  p.val<node.val:
                mn = node.val
                node = node.left
            else:
                node = node.right
        return mn
```

Robot Room Cleaner

Go to Discuss

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(object):
#    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
#        """
```

```python
class Solution(object):
    def cleanRoom(self, robot):
        """
        :type robot: Robot
        :rtype: None
        """
        self.dfs(robot, 0, 0, 0, 1, set())

    def dfs(self, robot, x, y, direction_x, direction_y, visited):
        robot.clean()
        visited.add((x, y))

        for k in range(4):
            neighbor_x = x + direction_x
            neighbor_y = y + direction_y
            if (neighbor_x, neighbor_y) not in visited and robot.move():
                self.dfs(robot, neighbor_x, neighbor_y, direction_x, direction_y, visited)
                robot.turnLeft()
                robot.turnLeft()
                robot.move()
                robot.turnLeft()
                robot.turnLeft()
            robot.turnLeft()
            direction_x, direction_y = -direction_y, direction_x
```

Redundant Connection II

Go to Discuss

In this problem, a rooted tree is a **directed** graph such that, there is exactly one node (the root) for which all other nodes are descendants of this node, plus every node has exactly one parent, except for the root node which has no parents.

The given input is a directed graph that started as a rooted tree with N nodes (with distinct values 1, 2, ..., N), with one additional directed edge added. The added edge has two different vertices chosen from 1 to N, and was not an edge that already existed.

The resulting graph is given as a 2D-array of `edges`. Each element of `edges` is a pair `[u, v]` that represents a **directed** edge connecting nodes `u` and `v`, where `u` is a parent of child `v`.

Return an edge that can be removed so that the resulting graph is a rooted tree of N nodes. If there are multiple answers, return the answer that occurs last in the given 2D-array.

Example 1:

```
Input: [[1,2], [1,3], [2,3]]
Output: [2,3]
Explanation: The given directed graph will be like this:
  1
 / \
v   v
2-->3
```

Example 2:

```
Input: [[1,2], [2,3], [3,4], [4,1], [1,5]]
Output: [4,1]
Explanation: The given directed graph will be like this:
5 <- 1 -> 2
     ^    |
     |    v
     4 <- 3
```

Note:

- The size of the input 2D-array will be between 3 and 1000.
- Every integer represented in the 2D-array will be between 1 and N, where N is the size of the input array.

```python
class Solution(object):
    def findRedundantDirectedConnection(self, edges):
        """
        :type edges: List[List[int]]
        :rtype: List[int]
        """
        candidate=[]
        self.par=[0]*(len(edges)+1)
        for u,v in edges:
            if self.par[v]!=0:
                candidate.append([self.par[v],v])
                candidate.append([u,v])
                break
            else:
                self.par[v]=u
        self.par=range(len(edges)+1)
        for u,v in edges:
            if candidate and [u,v]==candidate[1]:
                continue
            if self.unionFind(u)==v:
                if candidate:
                    return candidate[0]
                return [u,v]
            self.par[v]=u
        return candidate[1]
```

Course Schedule

Go to Discuss

There are a total of *n* courses you have to take, labeled from `0` to `n-1`.

Some courses may have prerequisites, for example to take course 0 you have to first take course 1, which is expressed as a pair: `[0,1]`

Given the total number of courses and a list of prerequisite **pairs**, is it possible for you to finish all courses?

Example 1:

```
Input: 2, [[1,0]]
Output: true
Explanation: There are a total of 2 courses to take.
             To take course 1 you should have finished course 0. So it is possible.
```

Example 2:

```
Input: 2, [[1,0],[0,1]]
Output: false
Explanation: There are a total of 2 courses to take.
             To take course 1 you should have finished course 0, and to take course 0 you sho
uld
             also have finished course 1. So it is impossible.
```

Note:

1. The input prerequisites is a graph represented by **a list of edges**, not adjacency matrices. Read more about how a graph is represented.
2. You may assume that there are no duplicate edges in the input prerequisites.

```python
class Solution(object):
    def canFinish(self, numCourses, prerequisites):
        graph = [[] for _ in xrange(numCourses)]
        visit = [0 for _ in xrange(numCourses)]
        for x, y in prerequisites:
            graph[x].append(y)
        def dfs(i):
            if visit[i] == -1:
                return False
            if visit[i] == 1:
                return True
            visit[i] = -1
            for j in graph[i]:
                if not dfs(j):
                    return False
            visit[i] = 1
            return True
        for i in xrange(numCourses):
            if not dfs(i):
                return False
        return True
```

Validate Binary Search Tree

Go to Discuss

Given a binary tree, determine if it is a valid binary search tree (BST).

Assume a BST is defined as follows:

- The left subtree of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

Example 1:

```
Input:
    2
   / \
  1   3
Output: true
```

```python
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None


class Solution(object):
    def isValidBST(self, root):

        def helper(node, lower, upper):  # set boundaries for node.val.

            if (not node):
                return True
            if lower < node.val < upper:
                return helper(node.left, lower, node.val) and helper(node.right, node.val, upper)
            else:
                return False

        return helper(root, -2147483649, 2147483648)  # min int and max int will be the bound for root.val
```

Closest Binary Search Tree Value

Go to Discuss

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

Note:

- Given target value is a floating point.
- You are guaranteed to have only one unique value in the BST that is closest to the target.

Example:

```
Input: root = [4,2,5,1,3], target = 3.714286

    4
   / \
  2   5
 / \
1   3

Output: 4
```

```python
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution(object):
    def closestValue(self, root, target):
        r = root.val
        while root:
            if abs(root.val - target) < abs(r - target):
                r = root.val
            root = root.left if target < root.val else root.right
        return r
```

Recursion

Recursion usually involve some kind of backtracking to enumerate all possibilities.

Note that Backtracking is a more general purpose algorithm. Depth-First search is a specific form of backtracking related to searching tree structures. Therefore we categorize those problems in "Trees and Graphs", even though they involve recursion.

There are usually some smart pruning techniques which optimizes backtracking. For a great introduction of how pruning works, take a look at this article. A great example is "Word Search II" (aka the Boggle solver), which uses a data structure to optimize the search.

- Word Squares
  Editor's choice: Frequently asked in Google onsite interview.
- Strobogrammatic Number II
- Word Search II
- Android Unlock Patterns

Word Squares

Go to Discuss

Given a set of words **(without duplicates)**, find all word squares 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 $0 \le k <$ 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:

```
    5
   / \
  1   4
     / \
    3   6
Output: false
Explanation: The input is: [5,1,4,null,null,3,6]. The root node's value
             is 5 but its right child's value is 4.
```

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).
```

```python
class Solution(object):
    def wordSquares(self, words):
        """
        :type words: List[str]
        :rtype: List[List[str]]
        """
        if len(words) == 0:
            return []
        dic = {}
        n = len(words[0])
        for word in words:
            for i in range(n):
                pref = word[:i + 1]
                if pref in dic:
                    dic[pref].append(word)
                else:
                    dic[pref] = [word]


        def nextsq(square = []):
            k = len(square)
            if k == n:
                res.append(square)
            else:
                pref = ""
                for wrd in square:
                    pref = pref + wrd[k]
                if pref in dic:
                    for word in dic[pref]:
                        nextsq(square + [word])
```

```
        res = []
        for tword in words:
            nextsq([tword])
        return res
```

Strobogrammatic Number II

Go to Discuss

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(object):
    def findStrobogrammatic(self, n):
        """
        :type n: int
        :rtype: List[str]
        """
        evenMidCandidate = ["11","69","88","96", "00"]
        oddMidCandidate = ["0", "1", "8"]
        if n == 1:
            return oddMidCandidate
        if n == 2:
            return evenMidCandidate[:-1]
        if n % 2:
            pre, midCandidate = self.findStrobogrammatic(n-1), oddMidCandidate
        else:
            pre, midCandidate = self.findStrobogrammatic(n-2), evenMidCandidate
        premid = (n-1)//2
        return [p[:premid] + c + p[premid:] for c in midCandidate for p in pre]
```

Word Search II

Go to Discuss

Given a 2D board and a list of words from the dictionary, find all words in the board.

Each word must be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once in a word.

Example:

```
Input:
words = ["oath","pea","eat","rain"] and board =
[
  ['o','a','a','n'],
  ['e','t','a','e'],
  ['i','h','k','r'],
  ['i','f','l','v']
]

Output: ["eat","oath"]
```

Note:

You may assume that all inputs are consist of lowercase letters `a-z`.

```
class Solution(object):
    def checkList(self, board, row, col, word, trie, rList):
        if row<0 or row>=len(board) or col<0 or col>=len(board[0]) or board[row][col] == '.' or board[row][col] not in trie: return
        c = board[row][col]
        _word= word + c
        if '#' in trie[c]:
            rList.add(_word)
            if len(trie[c]) == 1: return # if next node is empty, return as no there is no need to search further
        board[row][col] = '.'
        self.checkList(board, row-1, col, _word, trie[c], rList) #up
        self.checkList(board, row+1, col, _word, trie[c], rList) #down
        self.checkList(board, row, col-1, _word, trie[c], rList) #left
        self.checkList(board, row, col+1, _word, trie[c], rList) #right
        board[row][col] = c
```

```python
    def findWords(self, board, words):
        """
        :type board: List[List[str]]
        :type words: List[str]
        :rtype: List[str]
        """
        if not board or not words: return []
        # building Trie
        trie, rList = {}, set()
        for word in words:
            t = trie
            for c in word:
                if c not in t: t[c] = {}
                t = t[c]
            t['#'] = None
        for row in range(len(board)):
            for col in range(len(board[0])):
                if board[row][col] not in trie: continue
                self.checkList(board, row, col, "", trie, rList)
        return list(rList)
```

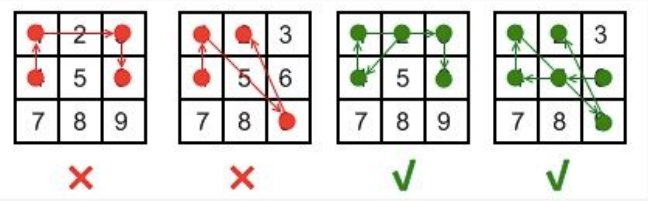
Android Unlock Patterns

Go to Discuss

Given an Android **3x3** key lock screen and two integers **m** and **n**, where $1 \leq m \leq n \leq 9$, count the total number of unlock patterns of the Android lock screen, which consist of minimum of **m** keys and maximum **n** keys.

Rules for a valid pattern:

1. Each pattern must connect at least **m** keys and at most **n** keys.
2. All the keys must be distinct.
3. If the line connecting two consecutive keys in the pattern passes through any other keys, the other keys must have previously selected in the pattern. No jumps through non selected key is allowed.
4. The order of keys used matters.



Explanation:

```
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |
```

Invalid move: `4 - 1 - 3 - 6`

Line 1 - 3 passes through key 2 which had not been selected in the pattern.

Invalid move: `4 - 1 - 9 - 2`

Line 1 - 9 passes through key 5 which had not been selected in the pattern.

Valid move: `2 - 4 - 1 - 3 - 6`

Line 1 - 3 is valid because it passes through key 2, which had been selected in the pattern

Valid move: `6 - 5 - 4 - 1 - 9 - 2`

Line 1 - 9 is valid because it passes through key 5, which had been selected in the pattern.

Example:

```
Input: m = 1, n = 1
Output: 9
```

```python
class Solution(object):
    def numberOfPatterns(self, m, n):
        """
        :type m: int
        :type n: int
        :rtype: int
        """
        if m is None or n is None or n==0:
            return 0
        jump=[[0]*10 for i in range(10)]
        jump[1][3]=jump[3][1]=2
        jump[4][6]=jump[6][4]=5
        jump[7][9]=jump[9][7]=8
        jump[1][7]=jump[7][1]=4
        jump[2][8]=jump[8][2]=5
        jump[3][9]=jump[9][3]=6
        jump[1][9]=jump[9][1]=5
        jump[3][7]=jump[7][3]=5
        visited=[False]*10
        res=0
        res+=self.helper(1,1,jump,visited,m,n,0)*4
        res+=self.helper(2,1,jump,visited,m,n,0)*4
        res+=self.helper(5,1,jump,visited,m,n,0)
        return res
```

```
    def helper(self,num,length,jump,visited,m,n,cnt):
        if length>=m:
            cnt+=1
        if length==n:
            return cnt
        visited[num]=True
        for i in range(1,10):
            if visited[i]==False and (jump[num][i]==0 or visited[jump[num][i]]):
                cnt=self.helper(i,length+1,jump,visited,m,n,cnt)
        visited[num]=False
        return cnt
```

Sorting and Searching

Interval related problems are quite often asked at Google interviews.

Similar to "Array and Strings", interval related problems can be asked in the context of data stream. Some good examples are: My Calendar I, My Calendar II, Find Median from Data Stream, and Sliding Window Median.

729. My Calendar I

Medium 328 26 Favorite Share

Implement a `MyCalendar` class to store your events. A new event can be added if adding the event will not cause a double booking.

Your class will have the method, `book(int start, int end)`. Formally, this represents a booking on the half open interval `[start, end)`, the range of real numbers `x` such that `start <= x < end`.

A *double booking* happens when two events have some non-empty intersection (ie., there is some time that is common to both events.)

For each call to the method `MyCalendar.book`, return `true` if the event can be added to the calendar successfully without causing a double booking. Otherwise, return `false` and do not add the event to the calendar.

Your class will be called like this: `MyCalendar cal = new MyCalendar(); MyCalendar.book(start, end)`
Example 1:

```
MyCalendar();
MyCalendar.book(10, 20); // returns true
MyCalendar.book(15, 25); // returns false
MyCalendar.book(20, 30); // returns true
Explanation:
The first event can be booked.  The second can't because time 15 is already booked by another
event.
The third event can be booked, as the first event takes every time less than 20, but not
including 20.
```

Approach #1: Brute Force [Accepted]

Intuition

When booking a new event `[start, end)`, check if every current event conflicts with the new event. If none of them do, we can book the event.

Algorithm

We will maintain a list of interval *events* (not necessarily sorted). Evidently, two events `[s1, e1)` and `[s2, e2)` do *not* conflict if and only if one of them starts after the other one ends: either `e1 <= s2` OR `e2 <= s1`. By De Morgan's laws, this means the events conflict when `s1 < e2` AND `s2 < e1`.

Java Python Copy

```python
class MyCalendar(object):
    def __init__(self):
        self.calendar = []

    def book(self, start, end):
        for s, e in self.calendar:
            if s < end and start < e:
                return False
        self.calendar.append((start, end))
        return True
```

Complexity Analysis

- Time Complexity: $O(N^2)$, where N is the number of events booked. For each new event, we process every previous event to decide whether the new event can be booked. This leads to $\sum_k^N O(k) = O(N^2)$ complexity.
- Space Complexity: $O(N)$, the size of the `calendar`.

Approach #2: Balanced Tree [Accepted]

Intuition

If we maintained our events in *sorted* order, we could check whether an event could be booked in $O(\log N)$ time (where N is the number of events already booked) by binary searching for where the event should be placed. We would also have to insert the event in our sorted structure.

Algorithm

We need a data structure that keeps elements sorted and supports fast insertion. In Java, a `TreeMap` is the perfect candidate. In Python, we can build our own binary tree structure.

For Java, we will have a `TreeMap` where the keys are the start of each interval, and the values are the ends of those intervals. When inserting the interval `[start, end)`, we check if there is a conflict on each side with neighboring intervals: we would like `calendar.get(prev)) <= start <= end <= next` for the booking to be valid (or for `prev` or `next` to be null respectively.)

For Python, we will create a binary tree. Each node represents some interval `[self.start, self.end)` while `self.left, self.right` represents nodes that are smaller or larger than the current node.

Java | Python

```python
class Node:
    __slots__ = 'start', 'end', 'left', 'right'
    def __init__(self, start, end):
        self.start = start
        self.end = end
        self.left = self.right = None

    def insert(self, node):
        if node.start >= self.end:
            if not self.right:
                self.right = node
                return True
            return self.right.insert(node)
        elif node.end <= self.start:
            if not self.left:
                self.left = node
                return True
            return self.left.insert(node)
        else:
            return False
```

```python
class MyCalendar(object):
    def __init__(self):
        self.root = None

    def book(self, start, end):
        if self.root is None:
            self.root = Node(start, end)
            return True
        return self.root.insert(Node(start, end))
```

Complexity Analysis

- Time Complexity (Java): $O(N \log N)$, where N is the number of events booked. For each new event, we search that the event is legal in $O(\log N)$ time, then insert it in $O(1)$ time.
- Time Complexity (Python): $O(N^2)$ worst case, with $O(N \log N)$ on random data. For each new event, we insert the event into our binary tree. As this tree may not be balanced, it may take a linear number of steps to add each event.
- Space Complexity: $O(N)$, the size of the data structures used.

731. My Calendar II

Medium 308 54 Favorite Share

Implement a `MyCalendarTwo` class to store your events. A new event can be added if adding the event will not cause a **triple** booking.

Your class will have one method, `book(int start, int end)`. Formally, this represents a booking on the half open interval `[start, end)`, the range of real numbers `x` such that `start <= x < end`.

A *triple booking* happens when **three** events have some non-empty intersection (ie., there is some time that is common to all 3 events.)

For each call to the method `MyCalendar.book`, return `true` if the event can be added to the calendar successfully without causing a **triple** booking. Otherwise, return `false` and do not add the event to the calendar.

Your class will be called like this: `MyCalendar cal = new MyCalendar(); MyCalendar.book(start, end)`
Example 1:

```
MyCalendar();
MyCalendar.book(10, 20); // returns true
MyCalendar.book(50, 60); // returns true
MyCalendar.book(10, 40); // returns true
MyCalendar.book(5, 15); // returns false
MyCalendar.book(5, 10); // returns true
MyCalendar.book(25, 55); // returns true
```

```
Explanation:
The first two events can be booked.  The third event can be double booked.
The fourth event (5, 15) can't be booked, because it would result in a triple booking.
The fifth event (5, 10) can be booked, as it does not use time 10 which is already double booked.
The sixth event (25, 55) can be booked, as the time in [25, 40) will be double booked with the third event;
the time [40, 50) will be single booked, and the time [50, 55) will be double booked with the second event.
```

Note:

- The number of calls to `MyCalendar.book` per test case will be at most `1000`.
- In calls to `MyCalendar.book(start, end)`, `start` and `end` are integers in the range `[0, 10^9]`.

Approach #1: Brute Force [Accepted]

Intuition

Maintain a list of bookings and a list of double bookings. When booking a new event `[start, end)`, if it conflicts with a double booking, it will have a triple booking and be invalid. Otherwise, parts that overlap the calendar will be a double booking.

Algorithm

Evidently, two events `[s1, e1)` and `[s2, e2)` do *not* conflict if and only if one of them starts after the other one ends: either `e1 <= s2` OR `e2 <= s1`. By De Morgan's laws, this means the events conflict when `s1 < e2` AND `s2 < e1`.

If our event conflicts with a double booking, it's invalid. Otherwise, we add conflicts with the calendar to our double bookings, and add the event to our calendar.

Java | Python

```python
class MyCalendarTwo:
    def __init__(self):
        self.calendar = []
        self.overlaps = []

    def book(self, start, end):
        for i, j in self.overlaps:
            if start < j and end > i:
                return False
        for i, j in self.calendar:
            if start < j and end > i:
                self.overlaps.append((max(start, i), min(end, j)))
        self.calendar.append((start, end))
        return True
```

Complexity Analysis

- Time Complexity: $O(N^2)$, where N is the number of events booked. For each new event, we process every previous event to decide whether the new event can be booked. This leads to $\sum_k^N O(k) = O(N^2)$ complexity.
- Space Complexity: $O(N)$, the size of the `calendar`.

Approach #2: Boundary Count [Accepted]

Intuition and Algorithm

When booking a new event `[start, end)`, count `delta[start]++` and `delta[end]--`. When processing the values of `delta` in sorted order of their keys, the running sum `active` is the number of events open at that time. If the sum is 3 or more, that time is (at least) triple booked.

A Python implementation was not included for this approach because there is no analog to *TreeMap* available.

295. Find Median from Data Stream

Hard 992 22 Favorite Share

Median is the middle value in an ordered integer list. If the size of the list is even, there is no middle value. So the median is the mean of the two middle value.

For example,
`[2,3,4]`, the median is `3`

`[2,3]`, the median is `(2 + 3) / 2 = 2.5`

Design a data structure that supports the following two operations:

- void addNum(int num) - Add a integer number from the data stream to the data structure.
- double findMedian() - Return the median of all elements so far.

Example:

```
addNum(1)
addNum(2)
findMedian() -> 1.5
addNum(3)
findMedian() -> 2
```

```python
from heapq import *

class MedianFinder:

    def __init__(self):
        self.heaps = [], []

    def addNum(self, num):
        small, large = self.heaps
        heappush(small, -heappushpop(large, num))
        if len(large) < len(small):
            heappush(large, -heappop(small))

    def findMedian(self):
        small, large = self.heaps
        if len(large) > len(small):
            return float(large[0])
        return (large[0] - small[0]) / 2.0
```

Minimum Window Substring

Go to Discuss

Given a string S and a string T, find the minimum window in S which will contain all the characters in T in complexity O(n).

Example:

```
Input: S = "ADOBECODEBANC", T = "ABC"
Output: "BANC"
```

Note:

- If there is no such window in S that covers all characters in T, return the empty string `""`.
- If there is such window, you are guaranteed that there will always be only one unique minimum window in S.

```python
class Solution(object):
    def minWindow(self, s, t):
        need, missing = collections.Counter(t), len(t)
        i = I = J = 0
        for j, c in enumerate(s, 1):
            missing -= need[c] > 0
            need[c] -= 1
            if not missing:
                while i < j and need[s[i]] < 0:
                    need[s[i]] += 1
                    i += 1
                if not J or j - i <= J - I:
                    I, J = i, j
        return s[I:J]
```

Kth Largest Element in an Array

Go to Discuss

Find the **k**th largest element in an unsorted array. Note that it is the kth largest element in the sorted order, not the kth distinct element.

Example 1:

```
Input: [3,2,1,5,6,4] and k = 2
Output: 5
```

Example 2:

```
Input: [3,2,3,1,2,4,5,5,6] and k = 4
Output: 4
```

Note:

You may assume k is always valid, 1 ≤ k ≤ array's length.

```python
class Solution(object):
    # O(k+(n-k)lgk) time, min-heap
    def findKthLargest5(self, nums, k):
        return heapq.nlargest(k, nums)[k]
```

Shortest Distance from All Buildings

Go to Discuss

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.
```

```python
class Solution(object):
    def shortestDistance(self, grid):
        if not grid or not grid[0]:
            return -1

        matrix = [[[0,0] for i in range(len(grid[0]))] for j in range(len(grid))]

        cnt = 0    # count how many building we have visited
        for i in range(len(grid)):
            for j in range(len(grid[0])):
                if grid[i][j] == 1:
                    self.bfs([i,j], grid, matrix, cnt)
                    cnt += 1

        res = float('inf')
        for i in range(len(matrix)):
            for j in range(len(matrix[0])):
                if matrix[i][j][1]==cnt:
                    res = min(res, matrix[i][j][0])

        return res if res!=float('inf') else -1
```

```
    def bfs(self, start, grid, matrix, cnt):
        q = [(start, 0)]
        while q:
            tmp = q.pop(0)
            po, step = tmp[0], tmp[1]
            for dp in [(-1,0), (1,0), (0,1), (0,-1)]:
                i, j = po[0]+dp[0], po[1]+dp[1]
                # only the position be visited by cnt times will append to queue
                if 0<=i<len(grid) and 0<=j<len(grid[0]) and matrix[i][j][1]==cnt and grid[i][j]==0:
                    matrix[i][j][0] += step+1
                    matrix[i][j][1] = cnt+1
                    q.append(([i,j], step+1))
```

Find K-th Smallest Pair Distance

Go to Discuss

Given an integer array, return the k-th smallest **distance** among all the pairs. The distance of a pair (A, B) is defined as the absolute difference between A and B.

Example 1:

```
Input:
nums = [1,3,1]
k = 1
Output: 0
Explanation:
Here are all the pairs:
(1,3) -> 2
(1,1) -> 0
(3,1) -> 2
Then the 1st smallest distance pair is (1,1), and its distance is 0.
```

Note:

1. `2 <= len(nums) <= 10000`.
2. `0 <= nums[i] < 1000000`.
3. `1 <= k <= len(nums) * (len(nums) - 1) / 2`.

```python
class Solution(object):
    def smallestDistancePair(self, nums, k):
        def possible(guess):
            #Is there k or more pairs with distance <= guess?
            count = left = 0
            for right, x in enumerate(nums):
                while x - nums[left] > guess:
                    left += 1
                count += right - left
            return count >= k

        nums.sort()
        lo = 0
        hi = nums[-1] - nums[0]
        while lo < hi:
            mi = (lo + hi) / 2
            if possible(mi):
                hi = mi
            else:
                lo = mi + 1

        return lo
```

Find K Pairs with Smallest Sums

Go to Discuss

You are given two integer arrays **nums1** and **nums2** sorted in ascending order and an integer **k**.

Define a pair **(u,v)** which consists of one element from the first array and one element from the second array.

Find the k pairs $(u_1,v_1),(u_2,v_2) \ldots (u_k,v_k)$ with the smallest sums.

Example 1:

```
Input: nums1 = [1,7,11], nums2 = [2,4,6], k = 3
Output: [[1,2],[1,4],[1,6]]
Explanation: The first 3 pairs are returned from the sequence:
             [1,2],[1,4],[1,6],[7,2],[7,4],[11,2],[7,6],[11,4],[11,6]
```

Example 2:

```
Input: nums1 = [1,1,2], nums2 = [1,2,3], k = 2
Output: [1,1],[1,1]
Explanation: The first 2 pairs are returned from the sequence:
             [1,1],[1,1],[1,2],[2,1],[1,2],[2,2],[1,3],[1,3],[2,3]
```

Example 3:

```
Input: nums1 = [1,2], nums2 = [3], k = 3
Output: [1,3],[2,3]
Explanation: All possible pairs are returned from the sequence: [1,3],[2,3]
```

```python
class Solution(object):
    def kSmallestPairs(self, nums1, nums2, k):
        """
        :type nums1: List[int]
        :type nums2: List[int]
        :type k: int
        :rtype: List[List[int]]
        """
        import heapq
        size1 = len(nums1)
        size2 = len(nums2)
        q = []
        if size1 > 0:
            for i, n in enumerate(nums2):
                heapq.heappush(q, (nums1[0] + n, 0, i))

        result = []
        mx = min(k, size1 * size2)
        while len(result) < mx:
            n, i, j = heapq.heappop(q)
            result.append((nums1[i], nums2[j]))
            if i + 1 < size1:
                heapq.heappush(q, (nums1[i+1] + nums2[j], i+1, j))

        return result
```

Range Module

Go to Discuss

A Range Module is a module that tracks ranges of numbers. Your task is to design and implement the following interfaces in an efficient manner.

- `addRange(int left, int right)` Adds the half-open interval `[left, right)`, tracking every real number in that interval. Adding an interval that partially overlaps with currently tracked numbers should add any numbers in the interval `[left, right)` that are not already tracked.
- `queryRange(int left, int right)` Returns true if and only if every real number in the interval `[left, right)` is currently being tracked.
- `removeRange(int left, int right)` Stops tracking every real number currently being tracked in the interval `[left, right)`.

Example 1:

```
addRange(10, 20): null
removeRange(14, 16): null
queryRange(10, 14): true (Every number in [10, 14) is being tracked)
queryRange(13, 15): false (Numbers like 14, 14.03, 14.17 in [13, 15) are not being tracked)
queryRange(16, 17): true (The number 16 in [16, 17) is still being tracked, despite the remov
e operation)
```

Note:

- A half open interval `[left, right)` denotes all real numbers `left <= x < right`.
- `0 < left < right < 10^9` in all calls to `addRange, queryRange, removeRange`.
- The total number of calls to `addRange` in a single test case is at most `1000`.
- The total number of calls to `queryRange` in a single test case is at most `5000`.
- The total number of calls to `removeRange` in a single test case is at most `1000`.

`self.X` is a sorted list of x-coordinates used by add/remove, where the tracking might start/stop. The corresponding `self.track` values tell whether tracking is on at the coordinate and to its right. Removing is really just adding a range of False, so I reuse addRange for it.

```
class RangeModule(object):

    def __init__(self):
        self.X = [0, 10**9]
        self.track = [False] * 2

    def addRange(self, left, right, track=True):
        def index(x):
            i = bisect.bisect_left(self.X, x)
            if self.X[i] != x:
                self.X.insert(i, x)
                self.track.insert(i, self.track[i-1])
            return i
        i = index(left)
        j = index(right)
        self.X[i:j] = [left]
        self.track[i:j] = [track]

    def queryRange(self, left, right):
        i = bisect.bisect(self.X, left) - 1
        j = bisect.bisect_left(self.X, right)
        return all(self.track[i:j])

    def removeRange(self, left, right):
        self.addRange(left, right, False)
```

```python
class RangeModule(object):

    def __init__(self):
        self.X = [0, 10**9]
        self.track = [False] * 2

    def addRange(self, left, right, track=True):
        def index(x):
            i = bisect.bisect_left(self.X, x)
            if self.X[i] != x:
                self.X.insert(i, x)
                self.track.insert(i, self.track[i-1])
            return i
        i = index(left)
        j = index(right)
        self.X[i:j] = [left]
        self.track[i:j] = [track]

    def queryRange(self, left, right):
        i = bisect.bisect(self.X, left) - 1
        j = bisect.bisect_left(self.X, right)
        return all(self.track[i:j])

    def removeRange(self, left, right):
        self.addRange(left, right, False)


# Your RangeModule object will be instantiated and called as such:
# obj = RangeModule()
# obj.addRange(left,right)
# param_2 = obj.queryRange(left,right)
# obj.removeRange(left,right)
```

Sqrt(x)

Go to Discuss

Implement `int sqrt(int x)`.

Compute and return the square root of *x*, where *x* is guaranteed to be a non-negative integer.

Since the return type is an integer, the decimal digits are truncated and only the integer part of the result is returned.

Example 1:

```
Input: 4
Output: 2
```

Example 2:

```
Input: 8
Output: 2
Explanation: The square root of 8 is 2.82842..., and since
             the decimal part is truncated, 2 is returned.
```

```python
class Solution:
    def mySqrt(self, x):
        """
        :type x: int
        :rtype: int
        """
        if x==1: return 1 #deal with exception
        l, r = 0, x
        while l <= r:
            mid = (r+l)//2
            if mid * mid <= x < (mid+1)*(mid+1):
                return mid
            elif x < mid * mid:
                r = mid
            else:
                l = mid
```

Insert Interval

Go to Discuss

Given a set of *non-overlapping* intervals, insert a new interval into the intervals (merge if necessary).

You may assume that the intervals were initially sorted according to their start times.

Example 1:

```
Input: intervals = [[1,3],[6,9]], newInterval = [2,5]
Output: [[1,5],[6,9]]
```

Example 2:

```
Input: intervals = [[1,2],[3,5],[6,7],[8,10],[12,16]], newInterval = [4,8]
Output: [[1,2],[3,10],[12,16]]
Explanation: Because the new interval [4,8] overlaps with [3,5],[6,7],[8,10].
```

Sort Transformed Array

Go to Discuss

Given a **sorted** array of integers *nums* and integer values *a*, *b* and *c*. Apply a quadratic function of the form $f(x) = ax^2 + bx + c$ to each element x in the array.

The returned array must be in **sorted order**.

Expected time complexity: **O(*n*)**

Example 1:

```
Input: nums = [-4,-2,2,4], a = 1, b = 3, c = 5
Output: [3,9,15,33]
```

Example 2:

```
Input: nums = [-4,-2,2,4], a = -1, b = 3, c = 5
Output: [-23,-5,1,7]
```

```python
class Solution(object):
    def sortTransformedArray(self, nums, a, b, c):
        """
        :type nums: List[int]
        :type a: int
        :type b: int
        :type c: int
        :rtype: List[int]
        """

        res = [None] * len(nums)
        res = [None] * len(nums)
        if a == 0 and b >=0:
            return [b * x + c for x in nums]
        if a == 0 and b < 0:
            return ([b * x + c for x in nums])[::-1]
        pivot = (float(b) / 2 / float(a)) * (-1)
        left, right, tail = 0, len(nums) - 1, len(nums) - 1
        while left <= right:
            if abs(nums[left] - pivot) > abs(nums[right] - pivot):
                res[tail] = a * nums[left] ** 2 + b * nums[left] + c
                left += 1
            else:
                res[tail] = a * nums[right] ** 2 + b * nums[right] + c
                right -= 1
            tail -= 1
        if a > 0: return res
        return res[::-1]
```

Merge Intervals

Go to Discuss

Given a collection of intervals, merge all overlapping intervals.

Example 1:

```
Input: [[1,3],[2,6],[8,10],[15,18]]
Output: [[1,6],[8,10],[15,18]]
Explanation: Since intervals [1,3] and [2,6] overlaps, merge them into [1,6].
```

Example 2:

```
Input: [[1,4],[4,5]]
Output: [[1,5]]
Explanation: Intervals [1,4] and [4,5] are considered overlapping.
```

```python
class Solution(object):
    def merge(self, intervals):
        """
        :type intervals: List[Interval]
        :rtype: List[Interval]
        """
        if len(intervals) == 0:
            return []
        sorted_intervals = sorted(intervals, key=lambda x: x.start)
        min_temp = sorted_intervals[0].start
        max_temp = sorted_intervals[0].end
        if len(intervals) == 1:
            return [[min_temp, max_temp]]
        processed = []
        for i in range(1, len(sorted_intervals)):
            if sorted_intervals[i].start <= max_temp:
                max_temp = max(sorted_intervals[i].end, max_temp)
            else:
                processed.append([min_temp, max_temp])
                min_temp = sorted_intervals[i].start
                max_temp = sorted_intervals[i].end
        processed.append([min_temp, max_temp])
        return processed
```

Longest Palindromic Substring

Go to Discuss

Given a string **s**, find the longest palindromic substring in **s**. You may assume that the maximum length of **s** is 1000.

Example 1:

```
Input: "babad"
Output: "bab"
Note: "aba" is also a valid answer.
```

Example 2:

```
Input: "cbbd"
Output: "bb"
```

```python
class Solution(object):
    def longestPalindrome(self, s):
        A = '&#' + '#'.join(s) + '#@'
        dp = [0]*len(A)
        id = mx = 0
        maxl = center = 0

        for i in xrange(1, len(A)-1):
            if mx>i:
                dp[i] = min(mx-i, dp[2*id-i])

            while A[i+dp[i]+1] == A[i-dp[i]-1]:
                dp[i] += 1

            if dp[i]+i > mx:
                id = i
                mx = dp[i]+i

            if dp[i] > maxl:
                maxl = dp[i]
                center = i
        return s[(center-maxl)//2:(center+maxl)//2]
```

Diagonal Traverse

Go to Discuss

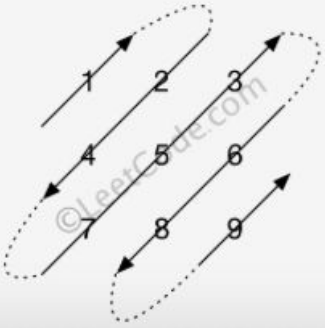
Given a matrix of M x N elements (M rows, N columns), return all elements of the matrix in diagonal order as shown in the below image.

Example:

```
Input:
[
 [ 1, 2, 3 ],
 [ 4, 5, 6 ],
 [ 7, 8, 9 ]
]

Output:  [1,2,4,7,5,3,6,8,9]

Explanation:
```




```python
class Solution(object):
    def findDiagonalOrder(self, matrix):
        """
        :type matrix: List[List[int]]
        :rtype: List[int]
        """
        result = []

        # '0' is dl, '1' is ur
        direction = 1
        lenR = len(matrix)
        lenC = 0
        if lenR > 0 and type(matrix[0]) == list:
            lenC = len(matrix[0])
        items = lenR * lenC
        (r,c) = (0,0)
```

```
        for _ in xrange(items):
            # Append current element
            result.append(matrix[r][c])

            # Compute next element
            if direction == 1:
                if r > 0 and (c+1) < lenC:
                    (r,c) = (r-1,c+1)
                elif c+1 < lenC:
                    (r,c) = (r,c+1)
                    direction = 0
                else:
                    (r,c) = (r+1,c)
                    direction = 0
            else:
                if c > 0 and (r+1) < lenR:
                    (r,c) = (r+1,c-1)
                elif r+1 < lenR:
                    (r,c) = (r+1,c)
                    direction = 1
                else:
                    (r,c) = (r,c+1)
                    direction = 1

        return result
```

Next Greater Element I

Go to Discuss

You are given two arrays **(without duplicates)** `nums1` and `nums2` where `nums1` 's elements are subset of `nums2` . Find all the next greater numbers for `nums1` 's elements in the corresponding places of `nums2` .

The Next Greater Number of a number **x** in `nums1` is the first greater number to its right in `nums2` . If it does not exist, output -1 for this number.

Example 1:

```
Input: nums1 = [4,1,2], nums2 = [1,3,4,2].
Output: [-1,3,-1]
Explanation:
    For number 4 in the first array, you cannot find the next greater number for it in the se
cond array, so output -1.
    For number 1 in the first array, the next greater number for it in the second array is 3.
    For number 2 in the first array, there is no next greater number for it in the second arr
ay, so output -1.
```

Example 2:

```
Input: nums1 = [2,4], nums2 = [1,2,3,4].
Output: [3,-1]
Explanation:
    For number 2 in the first array, the next greater number for it in the second array is 3.
    For number 4 in the first array, there is no next greater number for it in the second arr
ay, so output -1.
```

Note:

1. All elements in `nums1` and `nums2` are unique.
2. The length of both `nums1` and `nums2` would not exceed 1000.

```python
class Solution(object):
    def nextGreaterElement(self, findNums, nums):
        greater,stack = {},[]
        for n in nums:
            while stack and n > stack[-1]:
                greater[stack.pop()] = n
            stack.append(n)
        return [greater[n] if n in greater else -1 for n in findNums]
```

Pacific Atlantic Water Flow

Go to Discuss

Given an `m x n` matrix of non-negative integers representing the height of each unit cell in a continent, the "Pacific ocean" touches the left and top edges of the matrix and the "Atlantic ocean" touches the right and bottom edges.

Water can only flow in four directions (up, down, left, or right) from a cell to another one with height equal or lower.

Find the list of grid coordinates where water can flow to both the Pacific and Atlantic ocean.

Note:

1. The order of returned grid coordinates does not matter.
2. Both *m* and *n* are less than 150.

Example:

```
Given the following 5x5 matrix:

  Pacific ~   ~   ~   ~   ~
       ~  1   2   2   3  (5) *
       ~  3   2   3  (4) (4) *
       ~  2   4  (5)  3   1  *
       ~ (6) (7)  1   4   5  *
       ~ (5)  1   1   2   4  *
          *   *   *   *   * Atlantic
```

```
Return:

[[0, 4], [1, 3], [1, 4], [2, 2], [3, 0], [3, 1], [4, 0]] (positions with parentheses in above
 matrix).
```

```python
# Definition for an interval.
# class Interval(object):
#     def __init__(self, s=0, e=0):
#         self.start = s
#         self.end = e

class Solution:
    def insert(self, intervals, newInterval):
        start = newInterval.start
        end = newInterval.end
        right = left = 0
        while right < len(intervals):
            if start <= intervals[right].end:
                if end < intervals[right].start:
                    break
                start = min(start, intervals[right].start)
                end = max(end, intervals[right].end)
            else:
                left += 1
            right += 1
        return intervals[:left] + [Interval(start, end)] + intervals[right:]
```

```python
class Solution(object):
    def pacificAtlantic(self, matrix):
        if not matrix: return []
        m, n = len(matrix), len(matrix[0])
        def bfs(reachable_ocean):
            q = collections.deque(reachable_ocean)
            while q:
                (i, j) = q.popleft()
                for (di, dj) in [(0,1), (0, -1), (1, 0), (-1, 0)]:
                    if 0 <= di+i < m and 0 <= dj+j < n and (di+i, dj+j) not in
reachable_ocean \
                            and matrix[di+i][dj+j] >= matrix[i][j]:
                            q.append( (di+i,dj+j) )
                            reachable_ocean.add( (di+i, dj+j) )
            return reachable_ocean
        pacific  =set ( [ (i, 0) for i in range(m)]   + [(0, j) for j  in range(1, n)])
        atlantic =set ( [ (i, n-1) for i in range(m)] + [(m-1, j) for j in range(n-1)])
        return list( bfs(pacific) & bfs(atlantic) )
```

Evaluate Reverse Polish Notation

Go to Discuss

Evaluate the value of an arithmetic expression in Reverse Polish Notation.

Valid operators are `+`, `-`, `*`, `/`. Each operand may be an integer or another expression.

Note:

- Division between two integers should truncate toward zero.
- The given RPN expression is always valid. That means the expression would always evaluate to a result and there won't be any divide by zero operation.

Example 1:

```
Input: ["2", "1", "+", "3", "*"]
Output: 9
Explanation: ((2 + 1) * 3) = 9
```

Example 2:

```
Input: ["4", "13", "5", "/", "+"]
Output: 6
Explanation: (4 + (13 / 5)) = 6
```

Example 3:

```
Input: ["10", "6", "9", "3", "+", "-11", "*", "/", "*", "17", "+", "5", "+"]
Output: 22
Explanation:
  ((10 * (6 / ((9 + 3) * -11))) + 17) + 5
= ((10 * (6 / (12 * -11))) + 17) + 5
= ((10 * (6 / -132)) + 17) + 5
= ((10 * 0) + 17) + 5
= (0 + 17) + 5
= 17 + 5
= 22
```

```python
class Solution(object):
    def evalRPN(self, tokens):
        """
        :type tokens: List[str]
        :rtype: int
        """
        #tokens = ["10","6","9","3","+","-11","*","/","*","17","+","5","+"]
        ops = {
                '+': lambda x, y: x + y,
                '-': lambda x, y: x - y,
                '*': lambda x, y: x * y,
                '/': lambda x, y: x // y if x * y > 0 else -(-x//y)
            }
        q = collections.deque()
        for x in tokens:
            if x in ops:
                b = q.pop()
                a = q.pop()
                q.append(ops[x](a, b))
                #print(a, b, x, ops[x](a, b))
            else:
                q.append(int(x))
        return q[-1]
```

Dynamic Programming

Decode Ways

It can be tricky to identify the subproblems and connecting them, which is essential in solving Dynamic Programming problems. Dynamic programming is not that scary as you might think, and you can improve your dynamic programming skills greatly by practicing a lot of these problems.

According to our user survey, some of the most frequently asked Dynamic Programming Google interview questions include: Sentence Screen Fitting, Bomb Enemy, and Paint Fence.

- Decode Ways
- Word Break
  Editor's choice: Frequently asked in Google phone interview.
- Sentence Screen Fitting
- Maximum Vacation Days
  Editor's choice: Frequently asked in Google onsite interview.
- Edit Distance
- Minimum Path Sum
- House Robber II

418. Sentence Screen Fitting

Medium 300 168 Favorite Share

Given a `rows x cols` screen and a sentence represented by a list of **non-empty** words, find **how many times** the given sentence can be fitted on the screen.

Note:

1. A word cannot be split into two lines.
2. The order of words in the sentence must remain unchanged.
3. Two consecutive words **in a line** must be separated by a single space.
4. Total words in the sentence won't exceed 100.
5. Length of each word is greater than 0 and won't exceed 10.
6. 1 ≤ rows, cols ≤ 20,000.

Example 1:

```
Input:
rows = 2, cols = 8, sentence = ["hello", "world"]

Output:
1

Explanation:
hello---
world---

The character '-' signifies an empty space on the screen.
```

Example 2:

```
Input:
rows = 3, cols = 6, sentence = ["a", "bcd", "e"]

Output:
2

Explanation:
a-bcd-
e-a---
bcd-e-

The character '-' signifies an empty space on the screen.
```

Example 3:

```
Input:
rows = 4, cols = 5, sentence = ["I", "had", "apple", "pie"]

Output:
1

Explanation:
I-had
apple
pie-I
had--

The character '-' signifies an empty space on the screen.
```

```python
class Solution(object):
    def wordsTyping(self, sentence, rows, cols):
        """
        :type sentence: List[str]
        :type rows: int
        :type cols: int
        :rtype: int
        """
        s = ' '.join(sentence) + ' '
        start = 0
        for i in xrange(rows):
            start += cols - 1
            if s[start % len(s)] == ' ':
                start += 1
            elif s[(start + 1) % len(s)] == ' ':
                start += 2
            else:
                while start > 0 and s[ (start - 1) % len(s) ] != ' ':
                    start -= 1
        return start/ len(s)
```

361. Bomb Enemy

Medium 256 54 Favorite Share

Given a 2D grid, each cell is either a wall `'W'`, an enemy `'E'` or empty `'0'` (the number zero), return the maximum enemies you can kill using one bomb.
The bomb kills all the enemies in the same row and column from the planted point until it hits the wall since the wall is too strong to be destroyed.
Note: You can only put the bomb at an empty cell.

Example:

```
Input: [["0","E","0","0"],["E","0","W","E"],["0","E","0","0"]]
Output: 3
Explanation: For the given grid,

0 E 0 0
E 0 W E
0 E 0 0

Placing a bomb at (1,1) kills 3 enemies.
```

Compute a matrix for row-wise hits and one for column-wise hits. Then find the maximum.

```python
class Solution(object):
    def maxKilledEnemies(self, grid):
        def hits(grid):
            return [[h
                     for block in ''.join(row).split('W')
                     for h in [block.count('E')] * len(block) + [0]]
                    for row in grid]
        rowhits = hits(grid)
        colhits = zip(*hits(zip(*grid)))
        return max([rh + ch
                    for row in zip(grid, rowhits, colhits)
                    for cell, rh, ch in zip(*row)
                    if cell == '0'] or [0])
```

zip() in Python

The purpose of zip() is to **map the similar index of multiple containers** so that they can be used just using as single entity.

Syntax :
*zip(*iterators)*
Parameters :
Python iterables or containers (list, string etc)
Return Value :
Returns a single iterator object, having mapped values from all the containers.

```python
# Python code to demonstrate the working of
# zip()

# initializing lists
name = [ "Manjeet", "Nikhil", "Shambhavi", "Astha" ]
roll_no = [ 4, 1, 3, 2 ]
marks = [ 40, 50, 60, 70 ]

# using zip() to map values
mapped = zip(name, roll_no, marks)

# converting values to print as set
mapped = set(mapped)

# printing resultant values
print ("The zipped result is : ",end="")
print (mapped)
```

Output:

```
The zipped result is : {('Shamb-
havi', 3, 60), ('Astha', 2, 70),
('Manjeet', 4, 40), ('Nikhil', 1, 50)}
```

How to unzip?

Unzipping means converting the zipped values back to the individual self as they were. This is done with the help of "*" operator.

```
# Python code to demonstrate the working of
# unzip

# initializing lists

name = [ "Manjeet", "Nikhil", "Shambhavi", "Astha" ]
roll_no = [ 4, 1, 3, 2 ]
marks = [ 40, 50, 60, 70 ]

# using zip() to map values
mapped = zip(name, roll_no, marks)

# converting values to print as list
mapped = list(mapped)

# printing resultant values
print ("The zipped result is : ",end="")
print (mapped)

print("\n")

# unzipping values
namz, roll_noz, marksz = zip(*mapped)
```

```
print ("The unzipped result: \n",end="")

# printing initial lists
print ("The name list is : ",end="")
print (namz)

print ("The roll_no list is : ",end="")
print (roll_noz)

print ("The marks list is : ",end="")
print (marksz)
```

Output:

```
The zipped result is : [('Man-
jeet', 4, 40), ('Nikhil', 1, 50),
('Shambhavi', 3, 60), ('Astha', 2, 70)]


The unzipped result:
The name list is : ('Manjeet', 'Nikhil', 'Shamb-
havi', 'Astha')
The roll_no list is : (4, 1, 3, 2)
The marks list is : (40, 50, 60, 70)
```

Practical Applications : There are many possible applications that can be said to be exected using zip, be it **student database or scorecard** or any other utility that requires mapping of groups. A small example of score-card is demonstrated below.

```
# Python code to demonstrate the application of
# zip()

# initializing list of players.
players = [ "Sachin", "Sehwag", "Gambhir", "Dravid", "

# initializing their scores
scores = [100, 15, 17, 28, 43 ]

# printing players and scores.
for pl, sc in zip(players, scores):
    print ("Player :  %s     Score : %d" %(pl, sc))
```

Output:

```
Player :  Sachin     Score : 100
Player :  Sehwag     Score : 15
Player :  Gambhir     Score : 17
```

‹ Back | Python solution with explanation

orbuluh ★ 877 Last Edit: October 5, 2018 1:58 PM 10.2K VIEWS

154

If n == 1, there would be k-ways to paint.

if n == 2, there would be two situations:

- 2.1 You paint same color with the previous post: k*1 ways to paint, named it as `same`
- 2.2 You paint differently with the previous post: k*(k-1) ways to paint this way, named it as `dif`

So, you can think, if n >= 3, you can always maintain these two situations,
`You either paint the same color with the previous one, or differently.`

Since there is a rule: "no more than two adjacent fence posts have the same color."

We can further analyze:

- from 2.1, since previous two are in the same color, next one you could only paint differently, and it would form one part of "paint differently" case in the n == 3 level, and the number of ways to paint this way would equal to `same*(k-1)`.
- from 2.2, since previous two are not the same, you can either paint the same color this time (`dif*1`) ways to do so, or stick to paint differently (`dif*(k-1)`) times.

Here you can conclude, when seeing back from the next level, ways to paint the same, or variable `same` would equal to `dif*1 = dif`, and ways to paint differently, variable `dif`, would equal to `same*(k-1)+dif*(k-1) = (same + dif)*(k-1)`

So we could write the following codes:

```python
class Solution(object):
    def numWays(self, n, k):
        """
        :type n: int
        :type k: int
        :rtype: int
        """
        if n == 0:
            return 0
        if n == 1:
            return k
        same, dif = k, k*(k-1)
        for i in range(3, n+1):
            same, dif = dif, (same+dif)*(k-1)
        return same + dif
```

Decode Ways

Go to Discuss

A message containing letters from `A-Z` is being encoded to numbers using the following mapping:

```
'A' -> 1
'B' -> 2
...
'Z' -> 26
```

Given a **non-empty** string containing only digits, determine the total number of ways to decode it.

Example 1:

```
Input: "12"
Output: 2
Explanation: It could be decoded as "AB" (1 2) or "L" (12).
```

Example 2:

```
Input: "226"
Output: 3
Explanation: It could be decoded as "BZ" (2 26), "VF" (22 6), or "BBF" (2 2 6).
```

```python
class Solution:
    # @param s, a string
    # @return an integer
    def numDecodings(self, s):
        #dp[i] = dp[i-1] if s[i] != "0"
        #       +dp[i-2] if "09" < s[i-1:i+1] < "27"
        if s == "": return 0
        dp = [0 for x in range(len(s)+1)]
        dp[0] = 1
        for i in range(1, len(s)+1):
            if s[i-1] != "0":
                dp[i] += dp[i-1]
            if i != 1 and s[i-2:i] < "27" and s[i-2:i] > "09":  #"01"ways = 0
                dp[i] += dp[i-2]
        return dp[len(s)]
```

Word Break

Go to Discuss

Given a **non-empty** string *s* and a dictionary *wordDict* containing a list of **non-empty** words, determine if *s* can be segmented into a space-separated sequence of one or more dictionary words.

Note:

- The same word in the dictionary may be reused multiple times in the segmentation.
- You may assume the dictionary does not contain duplicate words.

Example 1:

```
Input: s = "leetcode", wordDict = ["leet", "code"]
Output: true
Explanation: Return true because "leetcode" can be segmented as "leet code".
```

Example 2:

```
Input: s = "applepenapple", wordDict = ["apple", "pen"]
Output: true
Explanation: Return true because "applepenapple" can be segmented as "apple pen apple".
             Note that you are allowed to reuse a dictionary word.
```

Example 3:

```
Input: s = "catsandog", wordDict = ["cats", "dog", "sand", "and", "cat"]
Output: false
```

```python
class Solution(object):
    def wordBreak(self, s, wordDict):
        """
        :type s: str
        :type wordDict: Set[str]
        :rtype: bool
        """
        dp = [False] * (len(s) + 1) # dp[i] means s[:i+1] can be segmented into words in the wordDicts
        dp[0] = True
        for i in range(len(s)):
            for j in range(i, len(s)):
                if dp[i] and s[i: j+1] in wordDict:
                    dp[j+1] = True

        return dp[-1]
```

Sentence Screen Fitting

Go to Discuss

Given a `rows x cols` screen and a sentence represented by a list of **non-empty** words, find **how many times** the given sentence can be fitted on the screen.

Note:

1. A word cannot be split into two lines.
2. The order of words in the sentence must remain unchanged.
3. Two consecutive words **in a line** must be separated by a single space.
4. Total words in the sentence won't exceed 100.
5. Length of each word is greater than 0 and won't exceed 10.
6. 1 ≤ rows, cols ≤ 20,000.

Example 1:

```
Input:
rows = 2, cols = 8, sentence = ["hello", "world"]

Output:
1

Explanation:
hello---
world---

The character '-' signifies an empty space on the screen.
```

Example 2:

```
Input:
rows = 3, cols = 6, sentence = ["a", "bcd", "e"]

Output:
2

Explanation:
a-bcd-
e-a---
bcd-e-

The character '-' signifies an empty space on the screen.
```

Example 3:

```
Input:
rows = 4, cols = 5, sentence = ["I", "had", "apple", "pie"]

Output:
1

Explanation:
I-had
apple
pie-I
had--

The character '-' signifies an empty space on the screen.
```

```python
class Solution(object):
    def wordsTyping(self, sentence, rows, cols):
        """
        :type sentence: List[str]
        :type rows: int
        :type cols: int
        :rtype: int
        """
        s = ' '.join(sentence) + ' '
        start = 0
        for i in xrange(rows):
            start += cols - 1
            if s[start % len(s)] == ' ':
                start += 1
            elif s[(start + 1) % len(s)] == ' ':
                start += 2
            else:
                while start > 0 and s[ (start - 1) % len(s) ] != ' ':
                    start -= 1
        return start/ len(s)
```

Maximum Vacation Days

Go to Discuss

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 Mo
day.)
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.
```

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.

```python
class Solution(object):
    def maxVacationDays(self, flights, days):
        """
        :type flights: List[List[int]]
        :type days: List[List[int]]
        :rtype: int
        """
        last = [(0, 0)]*len(days[0])

        for week in range(len(days[0])):
            tmp = []
            for currCity in range(len(flights)):
                for prev in last:
                    if (prev[1] == currCity or flights[prev[1]][currCity] == 1):
                        tmp += [(prev[0]+days[currCity][week], currCity)]
                        break
            last = sorted(tmp, reverse=True)
        return last[0][0]
```

Edit Distance

Go to Discuss

Given two words *word1* and *word2*, find the minimum number of operations required to convert *word1* to *word2*.

You have the following 3 operations permitted on a word:

1. Insert a character
2. Delete a character
3. Replace a character

Example 1:

```
Input: word1 = "horse", word2 = "ros"
Output: 3
Explanation:
horse -> rorse (replace 'h' with 'r')
rorse -> rose (remove 'r')
rose -> ros (remove 'e')
```

Example 2:

```
Input: word1 = "intention", word2 = "execution"
Output: 5
Explanation:
intention -> inention (remove 't')
inention -> enention (replace 'i' with 'e')
enention -> exention (replace 'n' with 'x')
exention -> exection (replace 'n' with 'c')
exection -> execution (insert 'u')
```

```python
class Solution(object):
    def minDistance(self, word1, word2):
        """
        :type word1: str
        :type word2: str
        :rtype: int
        """
        l1, l2 = len(word1)+1, len(word2)+1
        dp = range(l2)
        pre = 0
        for i in xrange(1, l1):
            pre, dp[0] = i-1, i
            for j in xrange(1, l2):
                buf = dp[j]
                dp[j] = min(dp[j]+1, dp[j-1]+1, pre+(word1[i-1]!=word2[j-1]))
                pre = buf
        return dp[-1]
```

Minimum Path Sum

Go to Discuss

Given a *m* x *n* grid filled with non-negative numbers, find a path from top left to bottom right which *minimizes* the sum of all numbers along its path.

Note: You can only move either down or right at any point in time.

Example:

```
Input:
[
  [1,3,1],
  [1,5,1],
  [4,2,1]
]
Output: 7
Explanation: Because the path 1→3→1→1→1 minimizes the sum.
```

```python
class Solution(object):
    def minPathSum(self, grid):
        """
        :type grid: List[List[int]]
        :rtype: int
        """
     # O(m*n) space

        if not grid:
            return
        r, c = len(grid), len(grid[0])
        dp = [[0 for _ in xrange(c)] for _ in xrange(r)]
        dp[0][0] = grid[0][0]
        for i in xrange(1, r):
            dp[i][0] = dp[i-1][0] + grid[i][0]
        for i in xrange(1, c):
            dp[0][i] = dp[0][i-1] + grid[0][i]
        for i in xrange(1, len(grid)):
            for j in xrange(1, len(grid[0])):
                dp[i][j] = min(dp[i-1][j], dp[i][j-1]) + grid[i][j]
        return dp[-1][-1]
```

House Robber II

Go to Discuss

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed. All houses at this place are **arranged in a circle.** That means the first house is the neighbor of the last one. Meanwhile, adjacent houses have security system connected and **it will automatically contact the police if two adjacent houses were broken into on the same night**.

Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight **without alerting the police**.

Example 1:

```
Input: [2,3,2]
Output: 3
Explanation: You cannot rob house 1 (money = 2) and then rob house 3 (money = 2),
             because they are adjacent houses.
```

Example 2:

```
Input: [1,2,3,1]
Output: 4
Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).
             Total amount you can rob = 1 + 3 = 4.
```

```python
class Solution(object):
    def rob(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        n = len(nums)
        if n == 0: return 0
        if n < 4: return max(nums)

        first, second = 0, 0
        for i in nums[:-1]: first, second = second, max(first + i, second)
        result = second

        first, second = 0, 0
        for i in nums[1:]: first, second = second, max(first + i, second)
        return max(result, second)
```