基础知识

Java :

1. Abstract Class vs Interface:

Both abstract class and interface are used for abstraction, which means hiding the implementation of the feature and only shows the functionality to the users.

Different:

1. **Type of methods: Interface can have only abstract methods. Abstract class can have abstract and non-abstract methods. From Java 8, it can have default and static methods also.**
2. **Final Variables: Variables declared in a Java interface are by default final. An abstract class may contain non-final variables.**
3. **Type of variables: Abstract class can have final, non-final, static and non-static variables. Interface has only static and final variables.**
4. **Implementation: Abstract class can provide the implementation of interface. Interface can’t provide the implementation of abstract class.**
5. **Inheritance vs Abstraction: A Java interface can be implemented using keyword “implements” and abstract class can be extended using keyword “extends”.**
6. **Multiple implementation: An interface can extend another Java interface only, an abstract class can extend another Java class and implement multiple Java interfaces.**

7. **Accessibility of Data Members: Members of a Java interface are public by default. A Java abstract class can have class members like private, protected, etc.**

2. Java Static Variable

A **static variable** is common to all the instances (or objects) of the class because it is a class level **variable**. In other words you can say that only a single copy of **static variable** is created and shared among all the instances of the class. ... **Static variables** are also known as Class **Variables**.

3. Final, finally, finalize

Final is a keyword, used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden and final variable value can't be changed.

Finally is a block, is used to place important code, it will be executed whether an exception is handled or not.

Finalize is a method, Finalize is used to perform clean up processing just before object is garbage collected.

4. **Primitive types** are the most basic data **types** available within the Java language. There are 8: boolean , byte (8) , char(16) , short(16) , int(32) , long(64) , float(32) and double(64) . These **types** serve as the building blocks of data manipulation in Java.

5. Java OOD

1. ***Abstraction*** is the process of exposing the essential details of an entity, while ignoring the irrelevant details, to reduce the complexity for the users.
2. ***Encapsulation*** is the process of bundling data and operations on the data together in an entity.
3. ***Inheritance*** is used to derive a new type from an existing type, thereby establishing a parent-child relationship.
4. ***Polymorphism*** lets an entity take on different meanings in different contexts.

6. Overloading vs Overriding

Overloading occurs when two or more methods in one class have the same method name but different parameters.

Overriding means having two methods with the same method name and parameters (i.e., *method signature*). One of the methods is in the parent class and the other is in the child class. Overriding allows a child class to provide a specific implementation of a method that is already provided its parent class.

7. String, StringBuilder, StringBuffer

In summary here is list of difference between StringBuffer, String, and StringBuilder in Java :

1) The String object is immutable in Java but StringBuffer and StringBuilder are mutable objects.

2) StringBuffer is synchronized while StringBuilder is not which makes StringBuilder faster than StringBuffer.

3) Concatenation operator "+" is internally implemented using either StringBuffer or StringBuilder.

4) Use String if you require immutability, use StringBuffer in java if you need mutable + thread-safety and use StringBuilder in Java if you require mutable + without thread-safety.

8. Process vs Thread

The similarity between thread and process is that they are all independent sequences of execution.

The typical difference is that threads (of the same process) run in a shared memory space, while processes run in separate memory spaces.

Process:
Process is a program in execution. Each process provides the resources needed to execute a program. A process has a virtual address space, executable code, open handles to system objects, context, a unique process identifier, variables, and at least one thread of execution.
Each process is started with a single thread, often called the primary thread, but can create additional threads from any of its threads. Each process has a (PCB) Process control block contains the information about processes, for example: Process priority, process id, process state, CPU, register etc. Process is isolated means **it does not share memory with any other process.**
Thread:
Thread is an entity within a process that can be scheduled for execution. A process starts with one thread, and then they can create multiple threads. All threads of a process share the virtual address space and system resources. However, it has its register and stack.

9. TCP vs UDP

The short answer: TCP is a transport-layer protocol, and HTTP is an application-layer protocol that runs over TCP

HTTP HTTPs In fact, **HTTPS** is basically an **HTTP** protocol with additional security. ... But in addition to adding that extra layer of security, **HTTPS** is also secured via TLS (Transport Layer Security) protocol.

10.

Mutex and Semaphore both provide synchronization services but they are not the same. Details about both Mutex and Semaphore are given below:

Mutex

Mutex is a mutual exclusion object that synchronizes access to a resource. It is created with a unique name at the start of a program. The Mutex is a locking mechanism that makes sure only one thread can acquire the Mutex at a time and enter the critical section. This thread only releases the Mutex when it exits the critical section.

This is shown with the help of the following example:

```
wait (mutex);
      …..
Critical Section
      …..
signal (mutex);
```

Semaphore

A semaphore is a signalling mechanism and a thread that is waiting on a semaphore can be signaled by another thread. This is different than a mutex as the mutex can be signaled only by the thread that called the wait function.

A semaphore uses two atomic operations, wait and signal for process synchronization.

The wait operation decrements the value of its argument S, if it is positive. If S is negative or zero, then no operation is performed.

There are mainly two types of semaphores i.e. counting semaphores and binary semaphores.

Counting Semaphores are integer value semaphores and have an unrestricted value domain. These semaphores are used to coordinate the resource access, where the semaphore count is the number of available resources.

The binary semaphores are like counting semaphores but their value is restricted to 0 and 1. The wait operation only works when the semaphore is 1 and the signal operation succeeds when semaphore is 0.

口头过了三四个test case，最后一道题只过了一个test case没时间了问了我其他可能的test case是什么

1) ✔️ First Duplicated Word ⇒ 正则split + Set O(N)

给一个String of words. 每个单词可能是被空格，逗号，分号等给分割开，让你找到第一个重复的词。我直接用了split加正则表达式把每个单词拿到放到数组里， 然后loop加个set解决。

```java
    public static String findFirstDuplicate(String words) {
        // In order to find the first duplicates, we need to use a data structure
to help us to remeber what we have seen before, and the most appropriate data
structure is Set.
        // Step 1 split the words into a String list of words with regular
expression
        String[] wordList = words.split("[, ;.]");
        Set<String> seen = new HashSet<>();
        for (String word : wordList) {
            // cos we might find the "", so we need to ignore it.
            if (word.equals("")) { continue; }
            if (seen.contains(word)) {
                return word;
            } else {
                seen.add(word);
            }
        }
        return "";
```

Time complexity O(n) Space complexity O(n)
Test Cases:

1. “This, is a sentence.This apple is good; that is not bad.”
2. “, ;.”
3. “Sentence”
4. “This sentence has no duplicate”

2) ✔️ 387. First Unique Character ⇒ int[26] as Map

```java
    public static int findFirstUniqueChar(String s) {
        // In order to find the first unique character, we need to use a data
structure to record what we have seen before, so the most appropriate data
structure is Map. Because the string only has English alphabet, so we could use a
int array to represents the map
        int[] seen = new int[26];
        for (int i = 0; i < s.length(); i++) {
            seen[s.charAt(i) - 'a']++;
        }
        for (int i = 0; i < s.length(); i++) {
            if (seen[s.charAt(i) - 'a'] == 1) {
                return i;
            }
        }
        return -1;
    }
```

This solution will pass the whole string two times.
We can do one pass

Follow up: One Pass

```
    public static int findFirstUniqueChar(String s) {
        int[] seen = new int[26];
        Arrays.fill(seen, -1); // fill with -1
        for (int i = 0; i < s.length(); i++) {
            char c = s.charAt(i);
            if (seen[c - 'a'] == -1) {
                // seen c first time, set the value as its index number
                seen[c - 'a'] = i;
            } else {
                // seen c more than first time, marked it as -2, means it not
possible be the answer.
                seen[c - 'a'] = -2;
            }
        }
        int first = Integer.MAX_VALUE;
        for (int index : seen) {
            if (index >= 0 && index < first) { first = index; }
        }
        return first == Integer.MAX_VALUE? -1 : first;
    }
```

Test cases:

1. “a”
2. “firstunique”
3. “today”

Linux的相关东西?

`ArithmeticException` is for

✔️Thrown when an exceptional arithmetic condition has occurred.

`IllegalArgumentException` is for

Thrown to indicate that a method has been passed an illegal or inappropriate argument.

Thrown when an exceptional arithmetic condition has occurred. For example, an integer "divide by zero" throws an instance of this class.

4) ✔️ 53. Maximum Subarray use a SUM variable

Print the subarray:

```java
public static int maxSubArray(int[] nums) {
    int sum = 0;
    int max = Integer.MIN_VALUE;
    int length = 0;
    int start = 0;
    int end = 0;
    for (int i = 0; i < nums.length; i++) {
        if (sum < 0) {
            sum = 0;
            start = i;
        }
        sum += nums[i];
        if (max < sum) {
            end = i;
            length = end - start + 1;
            max = sum;
        }
    }
    for (int i = end + 1 - length; i <= end; i++) {
        System.out.println(nums[i]);
    }
    return max;
```

Test cases:
[1,1,1,1,1,1,1], [-1,-1,-1,-1,-1], [1,-2,3,-4,5,6,7], [1,2,3,8,7,-15,-2, 10, 11]

Length >= 2 and print it ??? 【 size < nums.length的情况】
https://www.geeksforgeeks.org/largest-sum-subarray-least-k-numbers/
Test case:
[1,2,3,4,5] ⇒ 15
[-1,-1,-2,-3,-4] ⇒ -2
[7,9, -3,-4,-8,16] ⇒ 17
[-1, 32] ⇒ 31

```java
    public static int maxSubArrayK(int[] nums, int k) {
        // my idea is for each elem in nums, we can compute the max continues
array end at current element which length is at least k. Then we can find the
maxSubArray with at least k elements in whole array.
        // in order to compute the max continues array end at current element
which length is at least k. We decompose this question into two parts ==>
        // the max continues array end at current element which length is at
least k = the sum of nums[curr - k + 1] ~ nums[curr]
        //  + the max continues array end at nums[curr - k] at least one elem.
        int[] dp = new int[nums.length];
        dp[0] = nums[0];
        for (int i = 1; i < nums.length; i++) {
            dp[i] = Math.max(dp[i - 1] + nums[i], nums[i]);
        }

        int twoSum = 0;
        for (int i = 0; i < k; i++) { twoSum += nums[i]; }
        int maxSum = twoSum;
        for (int i = k; i < nums.length; i++) {
            twoSum += nums[i];
            twoSum -= nums[i - k];
            maxSum = Math.max(twoSum + Math.max(0, dp[i - k]), maxSum);
        }
        return maxSum;
    }
```

```
        int end = 0;
        int max = nums[0] + nums[1];
        for (int i = 2; i < nums.length; i++) {
            int maxEndAtHere = Math.max(nums[i] + nums[i - 1], nums[i] + nums[i -
1] + dp[i - 2]);
            if (max < maxEndAtHere) {
                max = maxEndAtHere;
                end = i;
            }
        }
        int sum = 0;
        int start = 0;
        for (int i = end; i >= 0; i--) {
            sum += nums[i];
            if (sum == max) {
                start = i;
                break;
            }
        }
        for (int i = start; i <= end; i++) { System.out.println(nums[i]); }
        return max;
    }
```

5) ✔️ 152. Maximum Product Subarray

```java
    public static int maxProduct(int[] nums) {
        // not only we need to record the previous max value end at current
        // but also we need to record the previous minimum value
        // cos a negative number multiply another negative number might be a
very large number
        int prevMax = nums[0];
        int prevMin = nums[0];
        int max = nums[0];
        for (int i = 1; i < nums.length; i++) {
            int tempMax = Math.max(nums[i], Math.max(nums[i] * prevMax, nums[i]
* prevMin));
            int tempMin = Math.min(nums[i], Math.min(nums[i] * prevMax, nums[i]
* prevMin));
            prevMax = tempMax;
            prevMin = tempMin;
            max = Math.max(max, prevMax);
        }
        return max;
    }
```

规定结果不能超过一个范围，然后返回最优解。

Length

If all positive, we can use sliding window to solve, otherwise

6) ✔️ 36. Valid Sudoku 一遍循环

```
class Solution {
    public boolean isValidSudoku(char[][] board) {
        if (board.length != board[0].length) { return false; }
        Set<Character>[] col = new Set[board.length];
        Set<Character>[] row = new Set[board.length];
        Set<Character>[] cube = new Set[(board.length / 3) * (board.length / 3)];
        for (int i = 0; i < col.length; i++) {
            col[i] = new HashSet<>();
            row[i] = new HashSet<>();
        }
        for (int i = 0; i < 9; i++) {
            cube[i] = new HashSet<>();
        }
        for (int i = 0; i < board.length; i++) {
            for (int j = 0; j < board.length; j++) {
                char c = board[i][j];
                if (c == '.') { continue; }
                if (row[i].contains(c) || col[j].contains(c) || cube[(i / 3) * 3 + j /
3].contains(c)) {
                    return false;
                } else {
                    row[i].add(c);
                    col[j].add(c);
                    cube[(i / 3) * 3 + j / 3].add(c);
                }
            }
        }
        return true;
    }
}
```

题目就是Validate Sudok。
第一次把check row, check column, check block分开写的。
后来要求写在一个循环里，说他们做validate credit card的时候就是这么去优化代码的。

```
public boolean isValidSudoku(char[][] board) {
    Set seen = new HashSet();
    for (int i=0; i<9; ++i) {
        for (int j=0; j<9; ++j) {
            char number = board[i][j];
            if (number != '.')
                if (!seen.add(number + " in row " + i) ||
                    !seen.add(number + " in column " + j) ||
                    !seen.add(number + " in block " + i/3 + "-" + j/3))
                    return false;
        }
    }
    return true;
}
```

7) ✔️ 68. Text Justification (word > maxWidth?) (简化)

Justify OneLine :

```
public static String justifyOneLine(String text, int maxWidth) {
    // all words can be put into one line
    String[] words = text.split(" ");
    if (words.length == 1) {
        return words[0] + fillWithSpace(maxWidth - words[0].length());
    }
    int totalSpace = maxWidth - countCharInWords(words);
    int numSpace = words.length - 1;
    String aveSpace = fillWithSpace(totalSpace / numSpace);
    int remain = totalSpace % numSpace;
    StringBuilder line = new StringBuilder();
    for (int i = 0; i < words.length - 1; i++) {
        line.append(words[i]);
        line.append(aveSpace);
        line.append(remain-- > 0 ? " ":"");
    }
    line.append(words[words.length - 1]);
    line.append(fillWithSpace(maxWidth - line.length()));
    return line.toString();
}
```

```
public static String fillWithSpace(int n) {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < n; i++) {
        sb.append(" ");
    }
    return sb.toString();
}

public static int countCharInWords(String[] words) {
    int cnt = 0;
    for (String word : words) {
        cnt += word.length();
    }
    return cnt;
}
```

完整版

```java
    public static List<String> fullJustify(String text, int maxWidth) {
        String[] words = text.split(" ");
        int left = 0;
        List<String> res = new ArrayList<>();
        while (left < words.length) {
            // find the last word int this line
            int right = findRight(left, words, maxWidth);
            res.add(justify(left, right, words, maxWidth));
            left = right + 1;
        }
        return res;
    }
```

```java
    public static int findRight(int left, String[] words, int maxWidth) {
        int right = left;
        int totalChar = words[right++].length();
        while (right < words.length && (totalChar + 1 +
words[right].length()) <= maxWidth) {
            totalChar += 1 + words[right++].length();
        }
        return right - 1;
    }
```

```
    public static String justify(int left, int right, String[] words, int
maxWidth) {
        // only have one word in this line, left justify
        if (right - left == 0) {
            return words[left] + generateSpace(maxWidth -
words[left].length());
        }
        boolean isLastLine = right == words.length - 1;
        int spaceNum = right - left;
        int totalSpace = maxWidth - countWordsLength(words, left, right);
        String space = isLastLine ? " " : generateSpace(totalSpace /
spaceNum);
        int remain = isLastLine ? 0 : totalSpace % spaceNum;
        StringBuilder result = new StringBuilder();
        for (int i = left; i < right; i++) {
            result.append(words[i]);
            result.append(space);
            result.append(remain-- > 0 ? " " : "");
        }
        result.append(words[right]);
        result.append(generateSpace(maxWidth - result.length()));
        return result.toString();
    }
```

```
    public static String generateSpace(int n) {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < n; i++) { sb.append(" "); }
        return sb.toString();
    }

    public static int countWordsLength(String[] words, int left, int right)
{
        int count = 0;
        for (int i = left; i <= right; i++) {
            count += words[i].length();
        }
        return count;
    }
}
```

刘巴 这次的要求是如果有某个词的长度超过每行的规定长度，就把这个词原封不动加上去就行，不用分割词来满足规定长度. 这题也给了no. 原因是写的不clean. 他们希望中间能**尽量用helper function**使得code 更clean…

❓🏁 followup: 单词长度超过了 maxwidth 需要分割

```java
public List<String> fullJustify(String[] words, int maxWidth) {
    //String[] words = text.split(" ");
    int left = 0;
    List<String> res = new ArrayList<>();
    while (left < words.length) {
        // find the last word int this line
        int right = findRight(left, words, maxWidth);
        // handle the case a word's length is bigger than maxwidth
        if (right == left && words[left].length() > maxWidth) {
            res.add(words[left].substring(0, maxWidth - 1) + "-");
            words[left] = words[left].substring(maxWidth - 1);
        } else {
            res.add(justify(left, right, words, maxWidth));
            left = right + 1;
        }
    }
    return res;
```

lc陆拾捌，先写一行的，follow up是原题

text justify 输入是一个string不是string[] 并且输入里面有换行符\n 碰到换行符需要立刻换行

所以不能用split弄成string[]
一道题，justify string:
1. 给一个string s和长度n
2. 把这个s按word分开，组成新string, 插空格进去保证长度为n
3. 空格要插的尽量均匀，最后一行可以不插

好问题！那你解决一下用“-”分割单词吧！

Example:
justify(25, "This is some sample text, just enough for the justify function to have a few lines to work with to show what it is supposed to do.")

Might return:
"This is some sample text,"
"just enough for the"
"justify function to have"
"a few lines to work with"

"to show what it is"
"supposed to do."

1. 是否为三角形 给定三个整数数组 a b c确保a[i] b[i] c[i] 三个整数是否可以形成三角形
如果是在 返回的字串数组中 String[i] = "Yes" 反之 "No"

2. 找出整数数组中
两两最小的差值
并把所有具有最小差值的对找出
并以ascending order的方式 "列印"出来

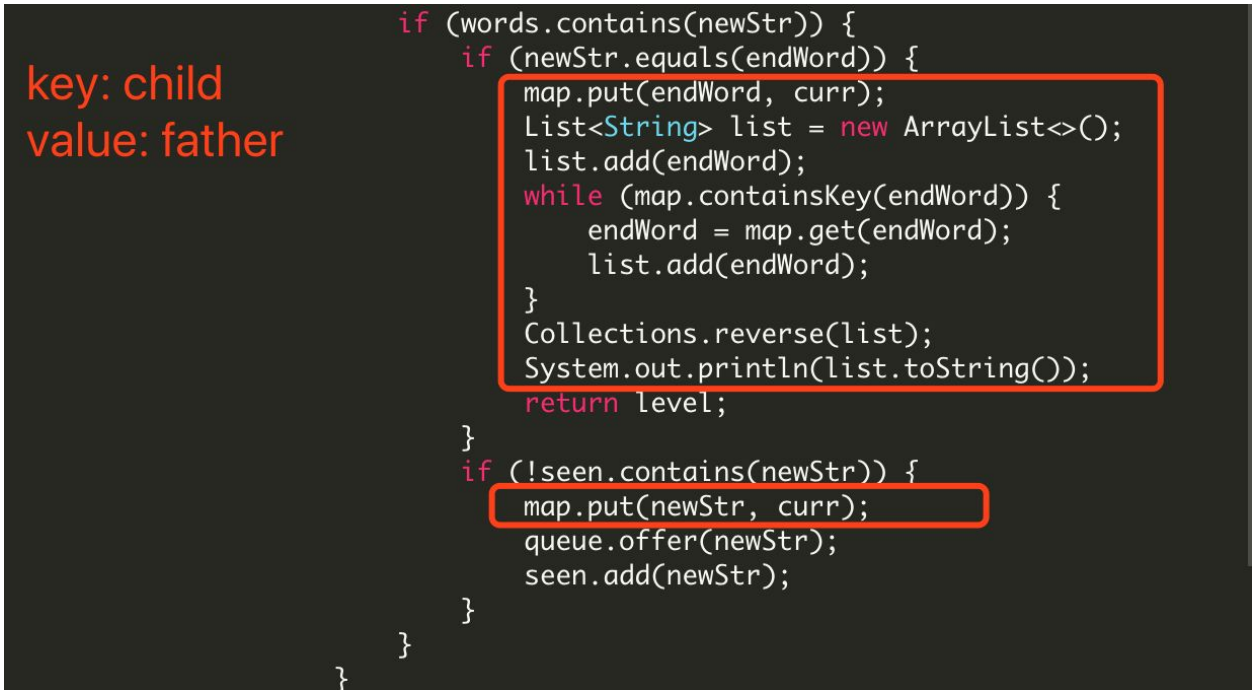
8) ✔️ 127. Word Ladder (Onsite)

word ladder I 的变形，**要求空间复杂度O(n), 用map，方法自己想，我就不提示了，然后是一堆讨论.**

```java
class Solution {
    public int ladderLength(String beginWord, String endWord, List<String> wordList) {
        Set<String> wordSet = new HashSet<>(wordList);
        if (!wordSet.contains(endWord)) { return 0; }
        Queue<String> queue = new LinkedList<>();
        Set<String> visited = new HashSet<>();
        queue.offer(beginWord);
        visited.add(beginWord);
        int level = 1;
        while (!queue.isEmpty()) {
            level++;
            int size = queue.size();
            while (size > 0) {
                size--;
                String curr = queue.poll();
                for (int i = 0; i < curr.length(); i++) {
                    char[] charArray = curr.toCharArray();
                    for (char c = 'a'; c <= 'z'; c++) {
                        charArray[i] = c;
                        String newString = new String(charArray);
                        if (!wordSet.contains(newString)) {
                            continue;
                        }
                        if (newString.equals(endWord)) {
                            return level;
                        }
                        else if (!visited.contains(newString)) {
                            queue.offer(newString);
                            visited.add(newString);
                        }
                    }
                }
            }
        }
        return 0;
    }
}
```

对于每个节点构图需要的时间复杂度是`O(26*L)`，BFS不重复地至多走过n个节点，故算法复杂度`O(26*L*n)`

Use the Map<child, father> Return any path



- Time Complexity: $O(M \times N)$, where M is the length of words and N is the total number of words in the input word list. Finding out all the transformations takes M iterations for each of the N words. Also, breadth first search in the worst case might go to each of the N words.
- Space Complexity: $O(M \times N)$, to store all M transformations for each of the N words, in the `all_combo_dict` dictionary. Visited dictionary is of N size. Queue for BFS in worst case would need space for all N words.

9) ✔️ 抄 126. Word Ladder2 (Onsite)

word ladder ii 但只需要返回一个最短路径 然后只用bfs做。。。**hashmap key是这层的词 value是上一层的词 这样你找到target就直接一路返回路径就好**

```java
class Solution {
    public List<List<String>> findLadders(String beginWord, String endWord, List<String>
wordList) {
        // step1 build the graph using BFS
        // use map to represent a graph ==> key is the father, a set of strings represents its
children
        Map<String, Set<String>> graph = new HashMap<>();
        Map<String, Integer> steps = new HashMap<>(); // steps represents how many steps tranform
to current string
        Set<String> wordSet = new HashSet<>(wordList);
        Set<String> seen = new HashSet<>();
        Queue<String> queue = new LinkedList<>();
        queue.offer(beginWord);
        seen.add(beginWord);
        boolean findEndWord = false;
        int level = 1;
        while (!queue.isEmpty()) {
            int size = queue.size();
            level++;
            while (size > 0) {
                size--;
                String currWord = queue.poll();
                for (int i = 0; i < currWord.length(); i++) {
                    char[] currWordArray = currWord.toCharArray();
                    for (char k = 'a'; k <= 'z'; k++) {
                        currWordArray[i] = k;
                        String newWord = new String(currWordArray);
                        if (wordSet.contains(newWord)) {
                            if (!steps.containsKey(newWord) || level <= steps.get(newWord)) {
                                if (!graph.containsKey(currWord)) { graph.put(currWord, new
HashSet<>()); }
                                graph.get(currWord).add(newWord);
                                steps.put(newWord, level);
                            }

                            if (newWord.equals(endWord)) { findEndWord = true; }
                            else if (!seen.contains(newWord)) {
                                queue.offer(newWord);
                                seen.add(newWord);
                            }
                        }
                    }
                }
            }
            if (findEndWord) { break; }
        }

        // step2 searching with DFS
        List<List<String>> res = new ArrayList<>();
        List<String> list = new ArrayList<>();
        list.add(beginWord);
        dfs(beginWord, endWord, res, list, graph, new HashSet<>(), 1, level);
        return res;
    }
```

```
    private void dfs(String curr, String endWord, List<List<String>> res, List<String> list,
Map<String, Set<String>> graph, Set<String> visited, int currDepth, int targetDepth) {
        if (currDepth > targetDepth) { return; }
        if (currDepth == targetDepth && curr.equals(endWord)) {
            res.add(new ArrayList<>(list)); }
        else {
            if (!graph.containsKey(curr)) { return; }
            for (String next : graph.get(curr)) {
                if (visited.contains(next)) { continue; }
                list.add(next);
                visited.add(next);
                dfs(next, endWord, res, list, graph, visited, currDepth + 1, targetDepth);
                list.remove(list.size() - 1);
                visited.remove(next);
            }
        }
    }
```

10) ❓ 给一个list，先按list的顺序构建出BST，再给一个值问插入这个值的话返回插入在第几层。follow up如果只有list不直接构建BST再返回插入到第几层。

11) ✔️ 65. Valid Number 看到数字 看到小数点 看到E

```java
class Solution {
    public boolean isNumber(String s) {
        s = s.trim();
        boolean seenDecimal = false;
        boolean seenE = false;
        boolean seenNumber = false;
        boolean seenNumberAfterE = false;
        for (int i = 0; i < s.length(); i++) {
            if (s.charAt(i) >= '0' && s.charAt(i) <= '9') {
                seenNumber = true;
                if (seenE) { seenNumberAfterE = true; }
            } else if (s.charAt(i) == '.') {
                if (seenE || seenDecimal) { return false; }
                seenDecimal = true;
            } else if (s.charAt(i) == 'e') {
                if (seenE || !seenNumber) { return false; }
                seenE = true;
            } else if (s.charAt(i) == '+' || s.charAt(i) == '-') {
                if (i != 0 && s.charAt(i - 1) != 'e') { return false; }
            } else { // invalid character
                return false;
            }
        }
        return seenNumber && (seenE && seenNumberAfterE || !seenE && !seenNumberAfterE);
    }
}
```

(do not need to consider 'e')

```java
    public static boolean isValidNumber(String s) {
        s = s.trim();
        boolean seenNum = false;
        boolean seenPoint = false;
        for (int i = 0; i < s.length(); i++) {
            char c = s.charAt(i);
            if (c >= '0' && c <= '9') {
                seenNum = true;
            } else if (c == '.') {
                if (seenPoint) { return false; }
                seenPoint = true;
            } else if (c == '+' || c == '-') {
                if (i != 0) { return false; }
            } else {
                return false;
            }
        }
        return seenNum;
    }
```

12) ✔️ 1. 给一列数，打印第一个不是null的数字。

1. 给一列数，打印第一个不是null的数字。
2. 从第一个不是null的数字开始，打印整个list。比如[null, 1, null, 2, 3] -> 1,,2,3,
3. 蠢口雾散。要求subarray的长度至少是2，打印subarray。

13) ✔️ 38. Count And Say

```
class Solution {
    public String countAndSay(int n) {
        int i = 1;
        String curr = "1";
        while (i < n) {
            StringBuilder sb = new StringBuilder();
            int begin = 0;
            for (int k = 0; k < curr.length(); k++) {
                if (k != 0 && curr.charAt(k - 1) != curr.charAt(k)) {
                    sb.append("" + (k - begin) + curr.charAt(k - 1));
                    begin = k;
                }
            }
            sb.append("" + (curr.length() - begin) + curr.charAt(begin));
            curr = sb.toString();
            i++;
        }
        return curr;

    }
}
```

14) ✔️ 76. Minimum Window Substring 简化版 set

```
    public static String minSubWin(String s, Set<Character> set) {
        Map<Character, Integer> seen = new HashMap<>();
        // using sliding window method to solve this problem
        int left = 0;
        int right = 0;
        int length = 0;
        int begin = 0;
        int minLen = Integer.MAX_VALUE;
        while (right < s.length()) {
            char c = s.charAt(right);
            if (set.contains(c)) {
                seen.put(c, seen.getOrDefault(c, 0) + 1);
            }
            while (seen.size() == set.size()) {
                char leftChar = s.charAt(left);
                if (set.contains(leftChar)) {
                    if (minLen > right - left + 1) {
                        minLen = right - left + 1;
                        begin = left;
                    }
                    seen.put(leftChar, seen.get(leftChar) - 1);
                    if (seen.get(leftChar) == 0) { seen.remove(leftChar); }
                }
                left++;
            }
        }
        return minLen == Integer.MAX_VALUE ? "" : s.substring(begin, begin +
minLen);
    }
}
```

完整版

```java
class Solution {
    public String minWindow(String s, String t) {
        Map<Character, Integer> need = new HashMap<>();
        Map<Character, Integer> seen = new HashMap<>();
        for (Character c : t.toCharArray()) { need.put(c, need.getOrDefault(c, 0) + 1); }
        int matched = 0;
        int left = 0;
        int right = 0;
        int minLength = Integer.MAX_VALUE;
        int start = 0;
        while (right < s.length()) {
            if (need.containsKey(s.charAt(right))) {
                seen.put(s.charAt(right), seen.getOrDefault(s.charAt(right), 0) + 1);
                if (seen.get(s.charAt(right)) <= need.get(s.charAt(right))) { matched++; }
            }
            while (left <= right && matched == t.length()) {
                if (need.containsKey(s.charAt(left))) {
                    int tempLength = (right - left + 1);
                    if (tempLength < minLength) {
                        start = left;
                        minLength = tempLength;
                    }
                    seen.put(s.charAt(left), seen.get(s.charAt(left)) - 1);
                    if (seen.get(s.charAt(left)) < need.get(s.charAt(left))) { matched--; }
                }
                left++;
            }
            right++;
        }
        return minLength == Integer.MAX_VALUE ? "" : s.substring(start, start + minLength);
    }
}
```

15) ✔️ 102. Binary Tree Level Order Traversal

```java
class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
        if (root == null) { return new ArrayList<>(); }
        List<List<Integer>> result = new ArrayList<>();
        Queue<TreeNode> queue = new LinkedList<>();
        queue.offer(root);
        while (!queue.isEmpty()) {
            int size = queue.size();
            List<Integer> list = new ArrayList<>();
            while (size > 0) {
                size--;
                TreeNode node = queue.poll();
                list.add(node.val);
                if (node.left != null) { queue.offer(node.left); }
                if (node.right != null) { queue.offer(node.right); }
            }
            result.add(list);
        }
        return result;
    }
}
```

2. 打印一颗二叉树

1

 2 3
  4 5
打印结果为
1
2 3
. 4 . 5

✔️ **D&C找到树的最大深度 ⇒ 然后在用BFS**

```
public static List<String> printTree(TreeNode root) {
    // find the depth of this binary tree
    int depth = getDepth(root);
    // BFS
    List<String> result = new ArrayList<>();
    Queue<TreeNode> queue = new LinkedList<>();
    queue.offer(root);
    int level = 1;
    while (!queue.isEmpty()) {
        int size = queue.size();
        StringBuilder sb = new StringBuilder();
        while (size > 0) {
            size--;
            TreeNode curr = queue.poll();
            if (curr == null) {
                sb.append(". ");
                queue.offer(null);
                queue.offer(null);
            } else {
                sb.append(curr.val + " ");
                queue.offer(curr.left);
                queue.offer(curr.right);
            }
```

```
            }
            result.add(sb.toString());
            level++;
            if (level > depth) { break; }
        }
        return result;

    }
    private static int getDepth(TreeNode root) {
        if (root == null) { return 0; }
        // using D & C
        int leftDepth = getDepth(root.left);
        int rightDepth = getDepth(root.right);
        return 1 + Math.max(leftDepth, rightDepth);
    }
}
class TreeNode {
    int val;
    TreeNode right;
    TreeNode left;
    public TreeNode(int val) {
        this.val = val;
    }
}
```

16) ✔️ 139 & 140. Word Break

139 word break I
Using dynamic programming to solve this problem
We can use a boolean array, for dp[i] = true means for the substring(0,i)(exclude i) can break into words.

```java
class Solution {
    public boolean wordBreak(String s, List<String> wordDict) {
        if (s == null || s.length() == 0) { return false; }
        // using dynamic programming to solve
        // convert list to Set
        Set<String> set = new HashSet<>(wordDict);
        boolean dp[] = new boolean[s.length() + 1];
        dp[0] = true;
        for(int i = 1; i < s.length() + 1; i++) {
            for (int k = i - 1; k >= 0; k--) {
                if (dp[k] && set.contains(s.substring(k, i))) {
                    dp[i] = true;
                    break;
                }
            }
        }
        return dp[s.length()];
    }
}
```

140 word break II

Recursion brute force

```java
class Solution {
    public List<String> wordBreak(String s, List<String> wordDict) {
        // brute force
        Set<String> wordSet = new HashSet<>(wordDict);
        return wordBreakHelper(s, wordSet, 0);
    }
    private List<String> wordBreakHelper(String s, Set<String> wordSet, int start)
{
        List<String> res = new ArrayList<>();
        if (start == s.length()) {
            res.add("");
            return res;
        }
        for (int i = start + 1; i <= s.length(); i++) {
            String curr = s.substring(start, i);
            if (wordSet.contains(curr)) {
                List<String> rest = wordBreakHelper(s, wordSet, i);
                for (String ss : rest) {
                    res.add(curr + (ss.equals("") ? "":" ") + ss);
                }
            }
        }
        return res;
    }
}
```

Use recursion with memoization

```java
class Solution {
    public List<String> wordBreak(String s, List<String> wordDict) {
        // brute force
        Set<String> wordSet = new HashSet<>(wordDict);
        Map<Integer, List<String>> map = new HashMap<>();
        return wordBreakHelper(s, wordSet, 0, map);
    }
    private List<String> wordBreakHelper(String s, Set<String> wordSet, int start,
Map<Integer, List<String>> map) {
        if (map.containsKey(start)) { return map.get(start); }

        List<String> res = new ArrayList<>();
        if (start == s.length()) {
            res.add("");
        }
        for (int i = start + 1; i <= s.length(); i++) {
            String curr = s.substring(start, i);
            if (wordSet.contains(curr)) {
                List<String> rest = wordBreakHelper(s, wordSet, i, map);
                for (String ss : rest) {
                    res.add(curr + (ss.equals("") ? "":" ") + ss);
                }
            }
        }
        map.put(start, res);
        return res;
    }
}
```

Time complexity ⇒ 2^n Space complexity is 2^n

```java
public List<String> wordBreak(String s, Set<String> wordDict) {
    return DFS(s, wordDict, new HashMap<String, LinkedList<String>>());
}

// DFS function returns an array including all substrings derived from s.
List<String> DFS(String s, Set<String> wordDict, HashMap<String, LinkedList<String>>map) {
    if (map.containsKey(s))
        return map.get(s);

    LinkedList<String>res = new LinkedList<String>();
    if (s.length() == 0) {
        res.add("");
        return res;
    }
    for (String word : wordDict) {
        if (s.startsWith(word)) {
            List<String>sublist = DFS(s.substring(word.length()), wordDict, map);
            for (String sub : sublist)
                res.add(word + (sub.isEmpty() ? "" : " ") + sub);
        }
    }
    map.put(s, res);
    return res;
}
```

17) implement a max Queue using this interface :

max que 实现**offer, poll, pollMax**, 要求所有操作log(n). 类似max stack的做法用treemap和double linkedlist来做.
所有的保证 O(logn) 复杂度

```
public interface MaxQueue {
    public void add(Long l);
    public Long poll();
    public Long pollMax();
}
```

实用 double LinkedList && PriorityQueue
follow up :
如果这个queue 要在多线程的环境下做 producer consumer 要怎么实现？

https://leetcode.com/playground/WUUCz5hL

18) ✔️ 186. Reverse Words in a String II

```java
    public static String reverse(String str) {
        char[] strArray = str.toCharArray();
        // step 1 reverse whole
        reverseAll(strArray, 0, strArray.length - 1);
        int left = 0;
        int right = 0;
        int length = strArray.length;
        // find the first word
        while (right < length) {
            // find the word's left bound
            while (left < length && strArray[left] == ' ') { left++; }
            right = left;
            // find the word's right bound
            while (right < length && strArray[right] != ' ') { right++; }
            reverseAll(strArray, left, right - 1);
            left = right + 1;
        }
        return new String(strArray);
    }
```

```java
    private static void reverseAll(char[] arr, int begin, int end) {
        int left = begin;
        int right = end;
        while (left < right) {
            char temp = arr[left];
            arr[left] = arr[right];
            arr[right] = temp;
            left++;
            right--;
        }
    }
```

19) ✔️ 101. Mirror Tree

recursion

```java
class Solution {
    public boolean isSymmetric(TreeNode root) {
        if (root == null) { return true; }
        return isSymmetricHelper(root.left, root.right);
    }
    private boolean isSymmetricHelper(TreeNode left, TreeNode right) {
        if (left == null && right == null) { return true; }
        if (left == null || right == null) { return false; }
        if (left.val != right.val) { return false; }
        return isSymmetricHelper(left.left, right.right) && isSymmetricHelper(left.right, right.left);
    }
}
```

Time complexity O(n) Space Complexity ⇒ worst O(n)

Iterate

```
public boolean isSymmetric(TreeNode root) {
        if(root == null)
            return true;
        Queue<TreeNode> q = new LinkedList();

        q.add(root.left);
        q.add(root.right);
        while(!q.isEmpty()){
            TreeNode left = q.poll();
            TreeNode right = q.poll();
            if(left == null && right == null)
                continue;
            if(left == null || right == null ||left.val != right.val )
                return false;
            q.add(left.left);
            q.add(right.right);
            q.add(left.right);
            q.add(right.left);

        }
        return true;

    }
```

21) ✔️ 443. String Compression

```
class Solution {
    public int compress(char[] chars) {
        int indexToFill = 0;
        int curr = 0;
        while (curr < chars.length) {
            int count = 0;
            char currChar = chars[curr];
            while (curr < chars.length && chars[curr] == currChar) {
                count++;
                curr++;
            }
            chars[indexToFill++] = currChar;
            if (count > 1) {
                for (Character c : Integer.toString(count).toCharArray()) {
                    chars[indexToFill++] = c;
                }
            }
        }
        return indexToFill;
    }
}
```

22) ✔️ 1.给一个字符串，输出所有不重复的字符并且按原字符串的顺序。用一个set搞定

```
public static void findUnique(String s) {
    Map<Character, Integer> map = new HashMap<>();
    char[] chars = s.toCharArray();
    for (Character c : chars) {
        map.put(c, map.getOrDefault(c, 0) + 1);
    }
    for (Character c : chars) {
        if (map.get(c) == 1) { System.out.print(c + ","); }
    }
}
```

23) ✔️ 341 nested list iterator

Give you a nested list, write a function to flat. eg. x = [1,2,[3,[4],5]], output = 1,2,3,4,5
Use the stack to solve this problem ⇒
First we push the whole nested list into the stack
Then every time, we pop the top element of the stack. Then, there are two situation,

⇒ 1. This element is a nested list.
⇒ 2. This element is a integer.

```
public class NestedIterator implements Iterator<Integer> {

    private Stack<ListIterator<NestedInteger>> stack;

    public NestedIterator(List<NestedInteger> nestedList) {
        stack = new Stack<>();
        stack.push(nestedList.listIterator());
    }

    @Override
    public Integer next() {
        if (hasNext()) {
            return stack.peek().next().getInteger();
        } else {
            return null;
        }
    }
```

```
    @Override
    public boolean hasNext() {
        while (!stack.isEmpty()) {
            if (!stack.peek().hasNext()) {
                stack.pop();
            } else {
                NestedInteger x = stack.peek().next();
                if (x.isInteger()) {
                    stack.peek().previous();
                    return true;
                } else {
                    stack.push(x.getList().listIterator());
                }
            }
        }
        return false;
    }
}
```

24) ✔️ 给一个binary tree 和一个target node, construct root到target的双向链表

input tree t: 1,2,3,4,5,null,null target: 5 要求返回root 1到target 5的路径的doubly linked list,即 1<->2<->5
第二题是在一个bst里面找root到target node的path，一开始我没看清，直接写了general tree里面找node，后来他提醒了，就改过来了。写完了还早，他就让我完成之前general tree的代码。自己给自己加难度算是。然后run了几个test case，问了点问题就结束了。

BST

We can use the property of BST to search the target node and build the linked list while searching.

```
public static Node getDlinkedListFromBST(TreeNode root, TreeNode target) {
    //
    Node dummy = new Node(-1);
    Node curr = dummy;
    while (root != null) {
        Node node = new Node(root.val);
        curr.next = node;
        node.prev = curr;
        curr = curr.next;
        if (root.val > target.val) {
            root = root.left;
        } else if (root.val < target.val) {
            root = root.right;
        } else {
            curr.next = dummy.next;
            dummy.next.prev = curr;
            break;
        }
    }
    return dummy.next;
}
```

General tree

For a general tree, we use preorder traversal to find the target node. If the node has found in this recursive path, we just return true, else we return false.

```
public static Node getDLinkedListFromTree(TreeNode root, TreeNode target) {
    Node dummy = new Node(-1);
    traverse(root, target, dummy);
    return dummy.next;

}
```

```java
private static boolean traverse(TreeNode root, TreeNode target, Node dummy) {
    if (root == null) {
        return false;
    }
    if (root.val == target.val) {
        dummy.next = new Node(root.val);
        dummy.next.next = dummy.next;
        dummy.next.prev = dummy.next;
        return true;
    }
    if (traverse(root.left, target, dummy)) {
        addAtFirst(root.val, dummy.next);
        return true;
    }
    if (traverse(root.right, target, dummy)) {
        addAtFirst(root.val, dummy.next);
        return true;
    }
    return false;
}
```

```java
private static void addAtFirst(int val, Node head) {
    Node node = new Node(val);
    node.next = head;
    node.prev = head.prev;
    head.prev.next = node;
    head.prev = node;
}
```

In Place

```java
public static TreeNode findRount(TreeNode root, TreeNode target) {
    traverse(root, target);
    TreeNode curr = root;
    while (curr.right != null) { curr = curr.right; }
    root.left = curr;
    curr.right = root;
    return root;
}
```

```java
  private static boolean traverse(TreeNode root, TreeNode target) {
      if (root == null) { return false; }
      if (root == target) {
          root.left = null;
          root.right = null;
          return true;
      }
      else {
          boolean findAtLeft = traverse(root.left, target);
          if (findAtLeft) {
              root.right = root.left;
              root.left.left = root;
              root.left = null;
              return true;
          }
          boolean findAtRight = traverse(root.right, target);
          if (findAtRight) {
              root.right.left = root;
              root.left = null;
              return true;
          }
          return false;
      }
  }
```

25) ✔️ TODO 224 && 227

2. 计算器的实现，参考利口贰贰柒，但是不完全一样，特别是输入部分，重点问了如何处理异常输入，我是口述的。

227

```java
class Solution {
    public int calculate(String s) {
        // in order to compute the expression, we need to compute all the multiply and
division first and store its answer
        // then sum all numbers
        // so I think we can use a stack, when we a number need to be added or minus, we just
put it into stack
        // when we see a number that need to be operate with operator multiply or division, we
pop the number on the top of stack, and then operate them, then add the result into the stack.
        // so how do we know what we should do for a number, we can use a variable to remember
the operator before it.
```

```
        char prevOperator = '+';
        Stack<Integer> stack = new Stack<>();
        int num = 0;
        int sum = 0;
        for (int i = 0; i < s.length(); i++) {
            char c = s.charAt(i);
            if (Character.isDigit(c)) {
                num = num * 10 + (c - '0');
            }
            if (!Character.isDigit(c) && c != ' ' || i == s.length() - 1) {
                if (prevOperator == '+') {
                    stack.push(num);
                } else if (prevOperator == '-') {
                    stack.push(-num);
                } else if (prevOperator == '*') {
                    int res = stack.pop() * num;
                    stack.push(res);
                } else if (prevOperator == '/') {
                    int res = stack.pop() / num;
                    stack.push(res);
                }
                prevOperator = c;
                num = 0;
            }
        }
        while (!stack.isEmpty()) {
            sum += stack.pop();
        }
        return sum;
    }
```

```
public int calculate(String s) {
    Stack<Integer> stack = new Stack<Integer>();
    int result = 0;
    int number = 0;
    int sign = 1;
    for(int i = 0; i < s.length(); i++){
        char c = s.charAt(i);
        if(Character.isDigit(c)){
            number = 10 * number + (int)(c - '0');
        }else if(c == '+'){
            result += sign * number;
            number = 0;
            sign = 1;
        }else if(c == '-'){
            result += sign * number;
            number = 0;
            sign = -1;
        }else if(c == '('){
            //we push the result first, then sign;
            stack.push(result);
            stack.push(sign);
            //reset the sign and result for the value in the parenthesis
            sign = 1;
            result = 0;
        }else if(c == ')'){
            result += sign * number;
            number = 0;
            result *= stack.pop();    //stack.pop() is the sign before the parenthesis
            result += stack.pop();   //stack.pop() now is the result calculated before the parenthesis

        }
    }
    if(number != 0) result += sign * number;
    return result;
}
```

26) ✔️ leaf doubly linked list

就是把一颗二叉树上所有的叶子节点连接成一个双向链表，注意要求有序！也就是按照他们各自的位置顺序。比如：

```
      1
   2     3
4     5
```

```
struct TreeNode {
    int val;
```

```
    TreeNode* left;
    TreeNode* right;
};
struct ListNode {
    int val;
    ListNode* next;
    ListNode* prev;
};
```

返回 4 <- >5 <-> 3 需要自己定义ListNode和TreeNode

```
    public static Node linkLeaf(TreeNode root) {
        // use preorder traversal
        Node tail = null;
        tail = traverse(root, tail);
        Node curr = tail;
        while (curr.prev != null) { curr = curr.prev; }
        curr.prev = tail;
        tail.next = curr;
        return curr;
    }
```

```
    private static Node traverse(TreeNode root, Node tail) {
        if (root == null) { return tail; }
        if (root.left == null && root.right == null) {
            if (tail == null) {
                tail = new Node(root.val);
            } else {
                Node node = new Node(root.val);
                tail.next = node;
                node.prev = tail;
                tail = node;
            }
            return tail;
        }
        tail = traverse(root.left, tail);
        tail = traverse(root.right, tail);
        return tail;
    }
```

In place

```
public static TreeNode linkLeaf(TreeNode root) {
    TreeNode prev = null;
    TreeNode tail = traverse(root, prev);
    prev = tail;
    while (prev.left != null) {
        prev = prev.left;
    }
    prev.left = tail;
    tail.right = prev;
    return prev;
}
```

```
private static TreeNode traverse(TreeNode root, TreeNode prev) {
    if (root == null) {
        return prev;
    } else {
        if (root.left == null && root.right == null) {
            if (prev == null) { prev = root; }
            else {
                prev.right = root;
                root.left = prev;
                prev = root;
            }
            return prev;
        }
        prev = traverse(root.left, prev);
        prev = traverse(root.right, prev);
        return prev;
    }
}
```

27) ✔️ 9. Palindrome Number

1. Convert number to string

```java
class Solution {
    public boolean isPalindrome(int x) {
        String number = "" + x;
        int left = 0;
        int right = number.length() - 1;
        while (left < right) {
            if (number.charAt(left) != number.charAt(right)) {
                return false;
            }
            left++;
            right--;
        }
        return true;
    }
}
```

2. O(1) space method
   Reverse the half number:

```java
class Solution {
    public boolean isPalindrome(int x) {
        // edge case
        if (x < 0 || x != 0 && x % 10 == 0) { return false; }
        int reverse = 0;
        while (x > reverse) {
            reverse = reverse * 10 + x % 10;
            x /= 10;
        }
        return (reverse == x || reverse / 10 == x);
    }
}
```

28) ✔️ 51. N-Queens

```java
class Solution {
    public List<List<String>> solveNQueens(int n) {
        List<List<String>> ans = new ArrayList<>();
        solveNQueensHelper(n, 0, new ArrayList<>(), ans);
        return ans;
    }

    private void solveNQueensHelper(int n, int row, List<String> list, List<List<String>> ans) {
        if (row == n) {
            ans.add(new ArrayList<>(list));
        } else {
            for (int i = 0; i < n; i++) {
                if (validPos(i, row, list, n)) {
                    StringBuilder sb = new StringBuilder();
                    for (int k = 0; k < n; k++) {
                        if (k == i) { sb.append('Q'); }
                        else { sb.append('.'); }
                    }
                    list.add(sb.toString());
                    solveNQueensHelper(n, row + 1, list, ans);
                    list.remove(list.size() - 1);
                }
            }
        }
    }
```

```java
    private boolean validPos(int col, int row, List<String> list, int n) {
        for (int i = row - 1; i >= 0; i--) {
            String curr = list.get(i);
            int leftInd = col - (row - i);
            int rightInd = col + (row - i);
            if (curr.charAt(col) == 'Q' ||
                (leftInd >= 0 && curr.charAt(leftInd) == 'Q') ||
                (rightInd < n && curr.charAt(rightInd) == 'Q')) {
                return false;
            }
        }
        return true;
    }
}
```

29) ✔️ 54. Spiral Matrix

```java
public List<Integer> spiralOrder(int[][] matrix) {
    List<Integer> res = new ArrayList<Integer>();
    if (matrix.length == 0) { return res; }
    int rowBegin = 0;
    int rowEnd = matrix.length-1;
    int colBegin = 0;
    int colEnd = matrix[0].length - 1;
    while (rowBegin <= rowEnd && colBegin <= colEnd) {
        for (int j = colBegin; j <= colEnd; j ++) {
            res.add(matrix[rowBegin][j]);
        }
        rowBegin++;
        for (int j = rowBegin; j <= rowEnd; j ++) {
            res.add(matrix[j][colEnd]);
        }
        colEnd--;
        if (rowBegin <= rowEnd) {
            for (int j = colEnd; j >= colBegin; j --) {
                res.add(matrix[rowEnd][j]);
            }
        }
        rowEnd--;
        if (colBegin <= colEnd) {
            for (int j = rowEnd; j >= rowBegin; j --) {
                res.add(matrix[j][colBegin]);
            }
        }
        colBegin ++;
    }
    return res;
```

```cpp
class Solution {
public:
    vector<int> spiralOrder(vector<vector<int>> &matrix) {
        vector<int> result;
        if (matrix.empty()) return result;
        result = matrix[0];  // no need to check the first row
        int dirs[4][2] = {{1, 0}, {0, -1}, {-1, 0}, {0, 1}};  // direction offsets
        int d = 0;  // direction index
        int m = matrix.size();
        int n = matrix[0].size();
        int pos[2] = {0, n - 1};  // start from the top right corner
        int i = (m - 1) * n;  // number of the rest numbers
        while (i > 0) {
            for (int j = 1; j < m; j++) {
                i--;
                pos[0] += dirs[d][0]; pos[1] += dirs[d][1];
                result.push_back(matrix[pos[0]][pos[1]]);
            }
            m--;  // decrease the size of row or column
            swap(m, n);  // switch between horizontal and vertical mode
            d = (d + 1) % 4;  // loop between direction offsets
        }
        return result;
    }
};
```

30) ✔️ 找到list中第一个大于给定字符串的字符串

题目：找到list中第一个大于给定字符串的字符串
输入: List<String>, String word
输出： String word

eg: 给你一个list：{hello, cow, cat, dog}，让你找出"cat"的下一个字典序最大。

美国小哥哥Jason面的，上来就是自我介绍一下，并没有问基础知识。之后就是coding，不是地里之前说的word ladder 2的变形了，题如下：一个file里一行一个单词（unsorted），比如：dog
Zebra
cow
monkey
donkey
...
input 是 dog， output是alphabetically的下一个也就是donkey。开始说的是别的解法，小哥要求先优化时间再优化空间复杂度，最后提示我让我用trie tree。一个并不熟悉trie tree的我被考到了T-T，最后没写完。。。肯定是要跪了。。。大家加油！

TreeSet

```java
public class FindNext {
    public static void main(String[] args) throws Exception {
        System.out.println(findNext( filePath: "/Users/mac126/19fall/udemy/spring/wepay/test",  target: "dog"));
    }
    public static String findNext(String filePath, String target) throws Exception {

        TreeSet<String> treeSet = new TreeSet(new Comparator<String>() {
            public int compare(String o1, String o2) {
                int curr1 = 0;
                int curr2 = 0;
                if (curr1 < o1.length() && curr2 < o2.length()) {
                    char c1 = o1.charAt(curr1);
                    char c2 = o2.charAt(curr2);
                    if (c1 < c2) {
                        return -1;
                    } else if (c1 > c2) {
                        return 1;
                    }
                }
                if (o1.length() < o2.length()) {
                    return -1;
                } else if (o1.length() > o2.length()) {
                    return 1;
                } else {
                    return 0;
                }
            }
        });
        File file = new File(filePath);
        if (!file.isFile() || !file.exists()) {
            throw new Exception("filePath invalid");
        }
        FileReader fileReader = new FileReader(file);
        BufferedReader bufferedReader = new BufferedReader(fileReader);
        String line = null;

        while ((line = bufferedReader.readLine()) != null) {
            treeSet.add(line);
        }
        return treeSet.higher(target);


    }
}
```

TrieTree

```java
class WordsList {
  private TrieNode root;
  public WordsList() {
    root = new TrieNode();
  }
  public void insert(String word) {
    TrieNode curr = root;
    for (int i = 0; i < word.length(); i++) {
      char c = word.charAt(i);
      if (curr.children[c - 'a'] == null) {
        curr.children[c - 'a'] = new TrieNode();
      }
      curr = curr.children[c - 'a'];
    }
    curr.word = word;
  }
  public String findNext(String word) {
    Stack<TrieNode> stack = new Stack<>();
    TrieNode curr = root;
    stack.push(curr);
    for (int i = 0; i < word.length(); i++) {
      char c = word.charAt(i);
```

```
            if (curr.children[c - 'a'] == null) { break; }
            curr = curr.children[c - 'a'];
            stack.push(curr);
        }
        while (!stack.isEmpty()) {
            TrieNode node = stack.pop();
            for (char c = 'a'; c <= 'z'; c++) {
                if (node.children[c - 'a'] != null) {
                    node = node.children[c - 'a'];
                    if (node.word != null) { return node.word; }
                    stack.push(node);
                    break;
                }
            }
        }
        return null;
    }
}
```

31) ✔️ 268 Missing Numbers

The basic idea is to use XOR operation. We all know that a^b^b =a, which means two xor operations with the same number will eliminate the number and reveal the original number.
In this solution, I apply XOR operation to both the index and value of the array. In a complete array with no missing numbers, the index and value should be perfectly corresponding(nums[index] = index), so in a missing array, what left finally is the missing number.

```
public int missingNumber(int[] nums) {

    int xor = 0, i = 0;
    for (i = 0; i < nums.length; i++) {
        xor = xor ^ i ^ nums[i];
    }

    return xor ^ i;
}
```

I guess this is more easier to understand:

public int missingNumber(int[] nums) {

```
    int res = nums.length;
    for(int i=0; i<nums.length; i++){
        res = res ^ i ^ nums[i]; // a^b^b = a
    }
    return res;
```

}

2.SUM

```
public int missingNumber(int[] nums) { //sum
    int len = nums.length;
    int sum = (0+len)*(len+1)/2;
    for(int i=0; i<len; i++)
        sum-=nums[i];
    return sum;
}
```

3.Binary Search

```
public int missingNumber(int[] nums) { //binary search
    Arrays.sort(nums);
    int left = 0, right = nums.length, mid= (left + right)/2;
    while(left<right){
        mid = (left + right)/2;
        if(nums[mid]>mid) right = mid;
        else left = mid+1;
    }
    return left;
}
```

32) ✔️ 285 inorder successor

Test case:
 2 find 1
1 3

1
 2
  5 find 5 , 4, 6
 4 6

Inorder traversal
Without parent

```java
class Solution {
    public TreeNode inorderSuccessor(TreeNode root, TreeNode p) {
        // using in-order traversal
        Stack<TreeNode> stack = new Stack<>();
        TreeNode curr = root;
        TreeNode succ = null;
        boolean find = false;

        while (curr != null || !stack.isEmpty()) {
            while (curr != null) {
                stack.push(curr);
                curr = curr.left;
            }
            curr = stack.pop();
            if (find) {
                return curr;
            }
            if (curr == p) { find = true; }
            curr = curr.right;
        }
        return null;
    }
}
```

```java
public TreeNode inorderSuccessor(TreeNode root, TreeNode p) {
    TreeNode succ = null;
    while (root != null) {
        if (p.val < root.val) {
            succ = root;
            root = root.left;
        }
        else
            root = root.right;
    }
    return succ;
}

// 29 / 29 test cases passed.
// Status: Accepted
// Runtime: 5 ms
```

```
public Node inorderSuccessor(Node x) {
        if(x == null) return null;

        if(x.right != null) {
            x = x.right;

            while(x != null && x.left != null) {
                x = x.left;
            }

            return x;
        }

        while(x != null)
        {
            if(x.parent == null)
                return null;

            if(x.parent.left == x)
                return x.parent;

            else {
                x = x.parent;
            }

        }

        return x;
    }
```

二、寻找前驱结点

1. 若一个结点有左子树，那么该结点的前驱节点是其左子树中val值最大的结点（即左子树中最右边的结点）
2. 若一个结点没有左子树
    1. 若该结点是其父结点的右孩子，那么该结点的前驱结点即为其父结点。
    2. 若该结点是其父结点的左孩子，那么需要沿着其父结点一直向树的顶端寻找，直到找到一个结点P，P结点是其父结点Q的右孩子，那么Q就是该结点的前驱结点。

三、寻找后继结点

1. 若一个结点有右子树，那么该结点的后继节点是其右子树中val值最小的结点（即右子树中最左边的结点）
2. 若一个结点没有右子树
    1. 若该结点是其父结点的左孩子，那么该节点的后继结点即为其父节点。
    2. 若该结点是其父结点的右孩子，那么需要沿着其父结点一直向树的顶端寻找，直到找到一个结点P，P结点是其父结点Q的左孩子，那么Q就是该结点的前驱结点。

举例分析：

以紫色方框中的子树（5，3，2，4）为例。

- 从一个角度看，要找到结点4的后继结点就需要沿着3寻找，此时找到结点3，发现结点3是其父结点5的右孩子，那么5就是结点4的后继结点。
- 从另一个角度看，结点4是子树（3，2，4）中val值最大的结点，而子树（3，2，4）是结点5的左子树，由左子树的结点总比当前结点的val值小，我们可以知道5就是结点4的后继结点。

33) ✔️ 785 二分图

```java
class Solution {
    public boolean isBipartite(int[][] graph) {
        if (graph == null || graph.length == 0) { return false; }

        int[] colors = new int[graph.length];
        for (int i = 0; i < colors.length; i++) {
            if (colors[i] == 0 && !dfs(graph, colors, i, 1)) {
                return false;
            }
        }
        return true;
    }
    private boolean dfs(int[][] graph, int[] colors, int curr, int color) {
        if (colors[curr] != 0) {
            return colors[curr] == color;
        } else {
            colors[curr] = color;
            for (int next : graph[curr]) {
                if (!dfs(graph, colors, next, -color)) { return false; }
            }
            return true;
        }
    }
}
```

- Time Complexity: $O(N + E)$, where N is the number of nodes in the graph, and E is the number of edges. We explore each node once when we transform it from uncolored to colored, traversing all its edges in the process.
- Space Complexity: $O(N)$ the space used to store the `color`.

34) ✔️ 通讯录 TrieTree 给定一个通讯录。在搜索栏中输入几个字母，就可以的到所有以这几个字母开头的联系人列表。问用什么数据结构存通讯录 ⇒ trie tree

```java
class TrieNode {
    public boolean isEnd;
    public TrieNode[] children;
    public TrieNode() {
        children = new TrieNode[26];
    }
}
```

```
class AddressBook {
    private TrieNode root;
    public AddressBook() {
        root = new TrieNode();
    }
    public void insert(String name) {
        TrieNode curr = root;
        for (int i = 0; i < name.length(); i++) {
            char c = name.charAt(i);
            if (curr.children[c - 'a'] == null) {
                curr.children[c - 'a'] = new TrieNode();
            }
            curr = curr.children[c - 'a'];
        }
        curr.isEnd = true;
    }
```

```
    public List<String> findWithPrefix(String prefix) {
        if (prefix == null || prefix.length() == 0) { return new ArrayList<>(); }
        List<String> res = new ArrayList<>();
        TrieNode curr = root;
        StringBuilder sb = new StringBuilder(prefix);
        for (int i = 0; i < prefix.length(); i++) {
            char c = prefix.charAt(i);
            if (curr.children[c - 'a'] == null) { return new ArrayList<>(); }
            curr = curr.children[c - 'a'];
        }
        dfs(curr, sb, res);
        return res;
    }
    private void dfs(TrieNode curr, StringBuilder sb, List<String> res) {
        if (curr.isEnd) { res.add(sb.toString()); }
        for (char c = 'a'; c <= 'z'; c++) {
            if(curr.children[c - 'a'] == null) { continue; }
            sb.append(c);
            dfs(curr.children[c - 'a'], sb, res);
            sb.deleteCharAt(sb.length() - 1);
        }
    }
}
```

35) ✔️ 再来三题coding,实际上是一个一个follow up：1.BST find root node to goal node path.2.Binary Tree find root node to goal node path3. Graph find from one node to goal node path

36) ✔️ double sqrt

```java
public static double sqrt(double n) throws Exception {
    if (n < 0) {
        throw new Exception("the Number is Negative");
    }
    double eps = 0.0000001;
    double last = 0;
    double left = 0;
    double right = n;
    double mid = left + (right - left) / 2;
    while (Math.abs(mid-last) > eps) {
        if (mid * mid > n) {
            right = mid;
        } else {
            left = mid;
        }
        last = mid;
        mid = left + (right - left) / 2;
    }
    return mid;
}
```

2. Merge two sorted array

```
class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
        int point1 = m - 1;
        int point2 = n - 1;
        int insertPos = m + n - 1;
        while(point1 >= 0 && point2 >= 0) {
            if (nums1[point1] >= nums2[point2]) {
                nums1[insertPos] = nums1[point1];
                point1--;
            } else {
                nums1[insertPos] = nums2[point2];
                point2--;
            }
            insertPos--;
        }
        // System.out.println(insertPos);
        // System.out.println(point1);
        // System.out.println(point2);
        if (point1 >= 0) {
            for (; point1 >= 0; point1--) {
                nums1[insertPos] = nums1[point1];
                insertPos--;
            }
        }
        if (point2 >= 0) {
            for (; point2 >= 0; point2--) {
                nums1[insertPos] = nums2[point2];
                insertPos--;
            }
        }
    }
}
```

37) ✔️ distinct palindromic substrings for a given string.

```
    public static Set<String> findAllDistinctPalin(String s) {
        Set<String> set = new HashSet<>();
        int axis = 0;
        while (axis < s.length()) {
            // find old
            find(s, set, axis, axis);
            find(s, set, axis, axis + 1);
            axis++;
        }
        return set;
    }
```

```java
private static void find(String s, Set<String> set, int left, int right) {
    while (left >= 0 && right < s.length()) {
        if (s.charAt(left) == s.charAt(right)) {
            set.add(s.substring(left, right + 1));
            left--;
            right++;
        } else {
            break;
        }
    }
}
```

2. 给一个map，map是英文名字和对应的昵称。写一个函数，把一个句子（String）中在map中出现过的名字替换成对应的昵称。

Use TrieTree