**java Collections数据结构**

LinkedHashSet 与 HashSet的区别：

* 这两者的insert, remove, 和查找的时间都是O(1)
* 内部实现：LinkedHashSet是LinkedHashMap, HashSet是HashMap
* **order of elements: LinkedHashSet用iterator()的顺序和insert的顺序一致**，reinsert已经存在的数据不改变顺序；HashSet无序；
* Memory occupation: LinkedHashSet 需要更多memory，因为需要保持linkedList

**关于顺序**

LinkedHashSet<Integer> set = new LinkedHashSet<>();

int nextNum = set.iterator().next(); // 这里取出的顺序就和放入的顺序一致，先取的一定是最先放入的

可以看这篇文章：

<http://javaconceptoftheday.com/hashset-vs-linkedhashset-vs-treeset-in-java/>

**数据结构设计**

146. LRU Cache

Design and implement a data structure for Least Recently Used (LRU) cache. It should support the following operations: get and put.

get(key) - Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.

put(key, value) - Set or insert the value if the key is not already present, if present, update the value. When the cache reached its capacity, it should invalidate the least recently used item before inserting a new item.

Follow up: Could you do both operations in O(1) time complexity?

**S1 DlinkedList**

这道题是非常经典的，用双端链表来做的题目。有一个global variable head, head.next是最近刚刚被调用过的node，head.prev是LRU node,

1. 每次put 新的node，直接加在head后面；
2. 每次到capacity之后，就删除head.prev；
3. 用map 存储<key, DlinedList>的配对

public class LRUCache {

HashMap<Integer, DLinkedList> cache;

DLinkedList head;

int capacity;

public LRUCache(int capacity) {

cache = new HashMap<>();

this.capacity = capacity;

head = new DLinkedList(0, 0);

head.prev = null;

head.next = null;

}

public int get(int key) {

DLinkedList node = cache.get(key);

if (node != null) {

head.updateNode(node);

return node.value;

} else return -1;

}

public void put(int key, int value) {

if (cache.containsKey(key)) {

DLinkedList node = cache.get(key);

head.deleteThisNode(node);

cache.remove(key);

} else {

if (cache.size() == capacity) {

int lrukey = head.deleteNode();

cache.remove(lrukey);

}

}

DLinkedList newNode = new DLinkedList(key, value);

cache.put(key, newNode);

head.addNode(newNode);

}

class DLinkedList {

int key;

int value;

DLinkedList prev;

DLinkedList next;

public DLinkedList(int key, int value) {

this.key = key;

this.value = value;

}

*// always add after head*

public void addNode(DLinkedList node) {

if (head.prev != null) {

node.next = head.next;

node.prev = head;

head.next = node;

node.next.prev = node;

} else {

head.next = node;

head.prev = node;

node.prev = head;

node.next = head;

}

}

*// always delete the one before head, return key of node deleted.*

public int deleteNode() {

int key = head.prev.key;

head.prev.prev.next = head;

head.prev = head.prev.prev;

return key;

}

public void deleteThisNode(DLinkedList node) {

node.prev.next = node.next;

node.next.prev = node.prev;

}

*// when a node is mentioned by get(), move its place to*

*// the one after head*

public void updateNode(DLinkedList node) {

if (node.prev != head) {

deleteThisNode(node);

addNode(node);

}

}

}

}

460. LFU Cache

Design and implement a data structure for Least Frequently Used (LFU) cache. It should support the following operations: get and put.

get(key) - Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.

put(key, value) - Set or insert the value if the key is not already present. When the cache reaches its capacity, it should invalidate the least frequently used item before inserting a new item. For the purpose of this problem, when there is a tie (i.e., two or more keys that have the same frequency), the least recently used key would be evicted.

Follow up: Could you do both operations in O(1) time complexity?

1. 需要保存每个key对应的value，count, 需要知道min count, 用Map, <key, value>, <key, count>

2. 并且能根据min count找出对应的key并把相关的数据evict出去，用Map, <count, set of keys>

3. 在min count对应多个key时，evict 的要是最早的那一个 – 需要set中的数据保持有序，顺序和insert顺序一致，用LinkedHashSet

4. 关于min count, 在每次放入新数据时，初始化为1，在每次调用get时检查，是否增加到min + 1.

class LFUCache {

Map<Integer, Integer> vals;

Map<Integer, Integer> freqs;

Map<Integer, LinkedHashSet<Integer>> lists; *//<count, <keys of this count>>*

int capacity;

int min = -1;

public LFUCache(int capacity) {

if (capacity <= 0) throw new Exception("...");

vals = new HashMap<>();

freqs = new HashMap<>();

lists = new HashMap<>();

this.capacity = capacity;

lists.put(1, new LinkedHashSet<>());

}

public int get(int key) {

if (!vals.containsKey(key)) return -1; *// not exist*

int freq = freqs.get(key);

freqs.put(key, freq + 1);

// 此时key 对应的 freq改变，增加了1，如果min == freq, 需要查看此时的min还是否有效，

// 也就是说需要查看<count, set of keys> 中是否还包含其他的key，如果不包含，只能min跟着++

lists.get(freq).remove(key);

*// we already know freq + 1 exist for sure.*

if (freq == min && lists.get(min).size() == 0) min++;

if (!lists.containsKey(freq + 1)) lists.put(freq + 1, new LinkedHashSet<>());

lists.get(freq + 1).add(key);

return vals.get(key);

}

public void put(int key, int value) {

if (vals.containsKey(key)) { *// still need to update freq*

vals.put(key, value);

get(key);

return;

}

if (vals.size() >= capacity) {

int evictKey = lists.get(min).iterator().next();

lists.get(min).remove(evictKey);

vals.remove(evictKey);

}

vals.put(key, value);

freqs.put(key, 1);

min = 1;

lists.get(1).add(key);

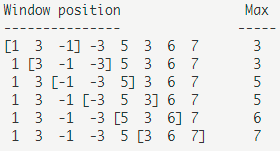
}

}

239. Sliding Window Maximum

Given an array nums, there is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position.

For example, Given nums = [1,3,-1,-3,5,3,6,7], and k = 3.



Therefore, return the max sliding window as [3,3,5,5,6,7].

Note: You may assume k is always valid, ie: 1 ≤ k ≤ input array's size for non-empty array.

S1 Deque解法，O(n), 真的太神了

1. 用一个deque, 从前面add，从后面remove，在不考虑max的情况下，就可以保持window:

public int[] maxSlidingWindow(int[] nums, int k) {

Deque<Integer> deque = new LinkedList<>();

for (int i = 0;i < nums.length;i++) {

deque.offerFirst(nums[i]);

*// ....*

if (i - k >= 0) deque.pollLast();

}

}

1. 接下来考虑怎么取得每个window中的max值。我们可以考虑让deque中的最大值始终保持在head或者tail中的一侧。既然我们add number是从head做，那么可以在每次加之前，都从head看一下head对应数字的大小是不是比现有的num小，如果小的话，就pollFirst，这样保证deque尾部是window中的最大值

public int[] maxSlidingWindow(int[] nums, int k) {

Deque<Integer> deque = new LinkedList<>();

for (int i = 0;i < nums.length;i++) {

while (!deque.isEmpty() && nums[i] > deque.peekFirst()) {

deque.pollFirst();

}

deque.offerFirst(nums[i]);

*// ....*

if (i - k >= 0) deque.pollLast();

}

}

（3）但这样就导致了一个问题，之前是在除了开头deque没有满的时候，每次都pollLast, 现在因为之前pollFirst,不能再直接pollLast了，有可能把下个window中的数字也拿掉。

解决这个问题的思路就是：存的不是数值，是nums的数组下标。比如说当i = 5, k = 3的时候，要维持window的大小是3，所以为了保证下一轮正常，i = 3,4,5对应的数字如果还在deque中的话，都不应该poll出 – 也就是只有在i = 2的时候需要poll出。而每次加入数字都从head加，所以可以保证deque 偏向head的数字的index一定靠后，最先加入deque的数字一定在tail，所以换一下pollLast的判断：

public int[] maxSlidingWindow(int[] nums, int k) {

Deque<Integer> deque = new LinkedList<>();

for (int i = 0;i < nums.length;i++) {

while (!deque.isEmpty() && nums[i] > deque.peekFirst()) {

deque.pollFirst();

}

deque.offerFirst(nums[i]);

*// ....*

if (deque.peekLast() == i - k) deque.pollLast();

}

}

总的代码如下：

public int[] maxSlidingWindow(int[] nums, int k) {

if(nums == null || nums.length == 0) return new int[0];

int[] rst = new int[nums.length - k + 1];

*// Deque stores indexes of elements i, in descending order of nums[i]*

*// First : smallet element in current window*

*// Last : biggest element in current window*

Deque<Integer> deque = new LinkedList<>();

for(int i = 0; i < nums.length; i++){

while(!deque.isEmpty() && nums[i] > nums[deque.peekFirst()]) {

deque.pollFirst();

}

deque.offerFirst(i);

*// element to remove from sliding window*

if(deque.peekLast() == i - k) deque.pollLast();

if(i - k + 1 >= 0) rst[i - k + 1] = nums[deque.peekLast()];

}

return rst;

}

**Data Stream**

295. Find Median from Data Stream

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**.

Examples: [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.

**S1. PriorityQueue**

用两个PQ实现，一个存<= median的数字（从大到小），

一个存> median的数字（从小到大），保证peek都是可以用于取median值的，

并且每次addNum时rebalance，保持两个PQ的大小差值不超过1.

PriorityQueue<Integer> minHeap; *// 从小到大，存 > median的值*

PriorityQueue<Integer> maxHeap; *// 从大到小，存 <= median的值*

public MedianFinder() {

maxHeap = new PriorityQueue<>(Collections.reverseOrder());

minHeap = new PriorityQueue<>();

}

public void addNum(int num) {

if (minHeap.isEmpty() || num > minHeap.peek()) {

minHeap.offer(num);

} else {

maxHeap.offer(num);

}

*// diff between heaps should <=1, rebalance if not.*

while (Math.abs(maxHeap.size() - minHeap.size()) > 1) {

if (maxHeap.size() > minHeap.size()) {

minHeap.offer(maxHeap.poll());

} else {

maxHeap.offer(minHeap.poll());

}

}

}

public double findMedian() {

if (minHeap.isEmpty() && maxHeap.isEmpty()) return 0.0;

int minSize = minHeap.size(), maxSize = maxHeap.size();

if (maxSize == minSize)

return (double)(minHeap.peek() + maxHeap.peek()) / 2;

else if (maxSize < minSize) return (double)minHeap.peek();

else return (double)maxHeap.peek();

}

346. Moving Average from Data Stream

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

For example,

MovingAverage m = new MovingAverage(3);

m.next(1) = 1

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

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

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

**S1. Deque**

这种fixed size window, 非常适合用deque做

class MovingAverage {

Deque<Integer> deque;

double sum;

int size;

*/\*\* Initialize your data structure here. \*/*

public MovingAverage(int size) {

deque = new LinkedList<>();

this.size = size;

this.sum = 0;

}

public double next(int val) {

if (deque.size() == size) sum -= deque.pollLast();

deque.offerFirst(val);

sum += val;

return sum / deque.size();

}

}

352. Data Stream as Disjoint Intervals

Given a data stream input of non-negative integers a1, a2, ..., an, ..., summarize the numbers seen so far as a list of disjoint intervals.

For example, suppose the integers from the data stream are 1, 3, 7, 2, 6, ..., then the summary will be:

[1, 1]

[1, 1], [3, 3]

[1, 1], [3, 3], [7, 7]

[1, 3], [7, 7]

[1, 3], [6, 7]

What if there are lots of merges and the number of disjoint intervals are small compared to the data stream's size?

**S1 TreeMap**

主要利用了TreeMap中的两个method：lowerKey, higherKey, 找到树中与num最接近的 >, < num的节点。

还有一个优化点：判断中会用到多次tree.get(l).end, 考虑到查找需要logn, 可以一开始就把数值存起来，注意判断null.

public class SummaryRanges {

TreeMap<Integer, Interval> tree;

public SummaryRanges() {

tree = new TreeMap<>();

}

public void addNum(int val) {

if (tree.containsKey(val)) return;

Integer l = tree.lowerKey(val);

Integer h = tree.higherKey(val);

int lend = l != null ? tree.get(l).end : -1;

if (l != null && h != null && lend + 1 == val && h == val + 1) {

*// [1..3], [5,...8] and insert 4.*

tree.get(l).end = tree.get(h).end;

tree.remove(h);

} else if (l != null && lend + 1 >= val) {

*// [1 ... 3] and insert 2 or 3, which is not key,*

*// but still exists or 1 bigger than tree.get(l).*

if (lend + 1 == val) tree.get(l).end = val;

} else if (h != null && h == val + 1) {

*// [1], [4...8], insert 3*

tree.put(val, new Interval(val, tree.get(h).end));

tree.remove(h);

} else {

tree.put(val, new Interval(val, val));

}

}

public List<Interval> getIntervals() {

return new ArrayList<>(tree.values());

}

}

**Design/OOD类题目**

359. Logger Rate Limiter

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

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

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

思路：

用map，用<message, latest time used + 10>来保存timestamp, 然后每次查有没有超出即可。

public class Logger {

Map<String, Integer> latestTime; *// <massage, the latest time used>*

*/\*\* Initialize your data structure here. \*/*

public Logger() {

latestTime = new HashMap<>();

}

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

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

*The timestamp is in seconds granularity. \*/*

public boolean shouldPrintMessage(int timestamp, String message) {

if (timestamp < latestTime.getOrDefault(message, 0)) {

return false;

}

latestTime.put(message, timestamp + 10);

return true;

}

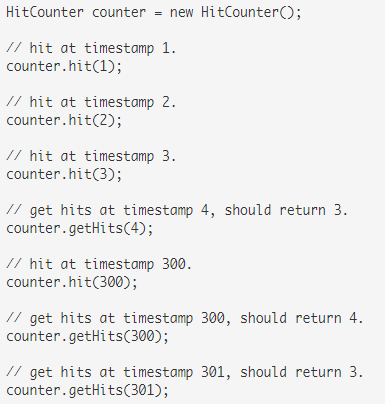
}

362. Design Hit Counter

Design a hit counter which counts the number of hits received in the past 5 minutes.

Each function accepts a timestamp parameter (in seconds granularity) and you may assume that calls are being made to the system in chronological order (ie, the timestamp is monotonically increasing). You may assume that the earliest timestamp starts at 1.

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



思路：

通过timestamp和hits分别记录hit的时间，和同一个时间hit的count。

public class HitCounter {

private int[] time;

private int[] hits;

*/\*\* Initialize your data structure here. \*/*

public HitCounter() {

time = new int[300];

hits = new int[300];

}

*/\*\* Record a hit.*

***@param*** *timestamp - The current timestamp (in seconds granularity). \*/*

public void hit(int timestamp) {

int idx = timestamp % 300;

if (timestamp != time[idx]) {

time[idx] = timestamp;

hits[idx] = 1;

} else { *// there are 1+ arrive at the same time*

hits[idx]++;

}

}

*/\*\* Return the number of hits in the past 5 minutes.*

***@param*** *timestamp - The current timestamp (in seconds granularity). \*/*

public int getHits(int timestamp) {

int count = 0;

for (int i = 0;i < 300;i++) {

if (timestamp - time[i] < 300)

count += hits[i];

}

return count;

}

}

355. Design Twitter

Design a simplified version of Twitter where users can post tweets, follow/unfollow another user and is able to see the 10 most recent tweets in the user's news feed. Your design should support the following methods:

* postTweet(userId, tweetId): Compose a new tweet.
* getNewsFeed(userId): Retrieve the 10 most recent tweet ids in the user's news feed. Each item in the news feed must be posted by users who the user followed or by the user herself. Tweets must be ordered from most recent to least recent.
* follow(followerId, followeeId): Follower follows a followee.
* unfollow(followerId, followeeId): Follower unfollows a followee.

每个User维护自己的tweets, post只是post到自己的列表里面，刷新时到follow列表抓。

因为需要最新的10条，所以tweets需要加timestamp, 自己维护全局变量，是典型merge k sorted list。

新建Tweet class, 为了便于按timestamp排序，实现了comparable接口，重写compareTo函数，使最新的tweets放在最前；

为了便于在merge的时候迅速拿出更早一个Tweet, 加入prev变量。

corner case：follow自己，取关不存在的人，取关没有关注的人，自己和关注的人post数 < 10等。

public class Twitter {

HashMap<Integer, LinkedList<Tweet>> tweetsMap; *// <userId, list of tweet>*

Map<Integer, Set<Integer>> follows; *// <follower A, followee B,C,D> A follows them*

int timestamp;

private class Tweet implements Comparable<Tweet>{

int timestamp;

int tweetId;

Tweet prev;

public Tweet(int timestamp, int tweetId, Tweet prev){

this.timestamp = timestamp;

this.tweetId = tweetId;

this.prev = prev;

}

public int compareTo(Tweet a){ *// in decrease order*

return a.timestamp - this.timestamp;

}

}

public Twitter() {

tweetsMap = new HashMap<>();

follows = new HashMap<>();

timestamp = 0;

}

*/\*\* Compose a new tweet. \*/*

public void postTweet(int userId, int tweetId) {

if (!tweetsMap.containsKey(userId)) {

tweetsMap.put(userId, new LinkedList<>());

}

int size = tweetsMap.get(userId).size();

Tweet prev = size == 0 ? null : tweetsMap.get(userId).peekFirst();

tweetsMap.get(userId).offerFirst(new Tweet(timestamp, tweetId, prev));

timestamp++;

}

*/\*\* Rqueue.etrieve the 10 most recent tweet ids in the user's news feed. Each item in the news feed must be posted by users who the user followed or by the user herself.*

*Tweets must be ordered from most recent to least recent. \*/*

public List<Integer> getNewsFeed(int userId) {

List<Integer> res = new ArrayList<>();

PriorityQueue<Tweet> queue = new PriorityQueue<>();

if (tweetsMap.containsKey(userId) && !tweetsMap.get(userId).isEmpty()) {

queue.offer(tweetsMap.get(userId).peekFirst());

}

if (follows.containsKey(userId)) {

for (int followee:follows.get(userId)) {

if (tweetsMap.containsKey(followee)

&& !tweetsMap.get(followee).isEmpty()) {

queue.offer(tweetsMap.get(followee).peekFirst());

}

}

}

for (int i = 0;i < 10 && !queue.isEmpty();i++) {

Tweet cur = queue.poll();

res.add(cur.tweetId);

if (cur.prev != null) queue.offer(cur.prev);

}

return res;

}

*/\*\* Follower follows a followee. If the operation is invalid, it should be a no-op. \*/*

public void follow(int followerId, int followeeId) {

if (followerId == followeeId) return;

if (!follows.containsKey(followerId)) follows.put(followerId, new HashSet<>());

follows.get(followerId).add(followeeId);

}

*/\*\* Follower unfollows a followee. If the operation is invalid, it should be a no-op. \*/*

public void unfollow(int followerId, int followeeId) {

if (!follows.containsKey(followerId) || !follows.get(followerId).contains(followeeId)) return;

follows.get(followerId).remove(followeeId);

}

}

348. Design Tic-Tac-Toe

Design a Tic-tac-toe game that is played between two players on a n x n grid.

You may assume the following rules:

1.A move is guaranteed to be valid and is placed on an empty block.

2.Once a winning condition is reached, no more moves is allowed.

3.A player who succeeds in placing n of their marks in a horizontal, vertical, or diagonal row wins the game.

**思路：**

用rows[i] 表示第i行的sum，用cols[j]表示第j行的sum。

如果是player1, 就+1，如果是play2, 就是-1，如果到size或者 -size, 就说明行/列已经满了，已经有人赢了。

public class TicTacToe {

private int[] rows;

private int[] cols;

private int diagonal;

private int antiDiagonal;

*/\*\* Initialize your data structure here. \*/*

public TicTacToe(int n) {

rows = new int[n];

cols = new int[n];

}

*/\*\* Player {player} makes a move at ({row}, {col}).*

***@param*** *row The row of the board.*

***@param*** *col The column of the board.*

***@param*** *player The player, can be either 1 or 2.*

***@return*** *The current winning condition, can be either:*

*0: No one wins.*

*1: Player 1 wins.*

*2: Player 2 wins. \*/*

public int move(int row, int col, int player) {

int val = player == 1 ? 1 : -1;

int size = rows.length;

rows[row] += val;

cols[col] += val;

if (row == col) diagonal += val;

if (row + col == size - 1) antiDiagonal += val;

if (Math.abs(rows[row]) == size ||

Math.abs(cols[col]) == size ||

Math.abs(diagonal) == size ||

Math.abs(antiDiagonal) == size) {

return player;

}

return 0;

}

}

379. Design Phone Directory

Design a Phone Directory which supports the following operations:

get: Provide a number which is not assigned to anyone.

check: Check if a number is available or not.

release: Recycle or release a number.

635. Design Log Storage System

You are given several logs that each log contains a unique id and timestamp. Timestamp is a string that has the following format: Year:Month:Day:Hour:Minute:Second, for example, 2017:01:01:23:59:59. All domains are zero-padded decimal numbers.

Design a log storage system to implement the following functions:

void Put(int id, string timestamp): Given a log's unique id and timestamp, store the log in your storage system.

int[] Retrieve(String start, String end, String granularity): Return the id of logs whose timestamps are within the range from start to end. Start and end all have the same format as timestamp. However, granularity means the time level for consideration. For example, start = "2017:01:01:23:59:59", end = "2017:01:02:23:59:59", granularity = "Day", it means that we need to find the logs within the range from Jan. 1st 2017 to Jan. 2nd 2017.

**S1 List**

这道题写的简洁不太容易，巧妙的点在于，这里units, 和indices的处理，两者对应，indice对应的是units中对应位所需要的位数。

然后通过string的compareTo方法来确定in range.

class LogSystem {

List<String[]> timestamps = new LinkedList<>();

List<String> units = Arrays.asList("Year", "Month", "Day", "Hour", "Minute", "Second");

int[] indices = new int[]{4,7,10,13,16,19}; *// index of timestamp*

public void put(int id, String timestamp) {

timestamps.add(new String[]{Integer.toString(id), timestamp});

}

public List<Integer> retrieve(String s, String e, String gra) {

List<Integer> res = new LinkedList<>();

int idx = indices[units.indexOf(gra)]; *// [0, idx) need to be considered*

for (String[] timestamp : timestamps) {

if (timestamp[1].substring(0, idx).compareTo(s.substring(0, idx)) >= 0 &&

timestamp[1].substring(0, idx).compareTo(e.substring(0, idx)) <= 0)

res.add(Integer.parseInt(timestamp[0]));

}

return res;

}

}

Houzz版本-Design Excel Formula

Houzz – Calendar Add Day

design calendar class 有 day year month 实现 add (days) return calendar

input是一个日期，即年月日，要求返回加上n天以后得到的日期。

public class Calendar {

int year;

int month;

int day;

static final int[] MONTH = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

static final int[] LEAP\_MONTH = {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

public Calendar(int year, int month, int day) {

this.year = year;

this.month = month;

this.day = day;

}

public static void main(String[] args) {

*// month start from 0, 0-based.*

GregorianCalendar gregorianCalendar = new GregorianCalendar(2000, 0, 1);

Calendar calendar = new Calendar(2000, 1, 1);

int[] days = {31, 365, 366, 1000, 3000};

for (int i = 0; i < days.length; i++) {

calendar.addDays(days[i]);

gregorianCalendar.add(java.util.Calendar.DAY\_OF\_YEAR, days[i]);

compareResult(calendar, gregorianCalendar);

}

}

public static void compareResult(Calendar calendar, GregorianCalendar gregorianCalendar) {

System.out.printf("Current Calendar Date: %d/%d/%d\n", calendar.month, calendar.day, calendar.year);

System.out.printf("Current GregorianCalendar Date: %d/%d/%d\n", gregorianCalendar.get(java.util.Calendar.MONTH),

gregorianCalendar.get(java.util.Calendar.DAY\_OF\_MONTH), gregorianCalendar.get(java.util.Calendar.YEAR));

}

public void addDays(int n) {

*// add year*

int leftDaysThisYear = leftDaysThisYear();

if (n > leftDaysThisYear) {

year++;

month = day = 1;

n -= leftDaysThisYear + 1;

}

int daysThisYear = isLeapYear(year) ? 366 : 365;

while (n > daysThisYear) {

n -= daysThisYear;

year++;

daysThisYear = isLeapYear(year) ? 366 : 365;

}

boolean isLeapYear = isLeapYear(year);

*// add month*

int leftDaysThisMonth = leftDaysThisMonth();

if (n > leftDaysThisMonth) {

month++;

day = 1;

n -= leftDaysThisMonth + 1;

}

int daysThisMonth = leftDaysThisMonth();

while (n > daysThisMonth && month + 1 <= 12) {

n -= isLeapYear ? LEAP\_MONTH[month] : MONTH[month];

month++;

daysThisMonth = leftDaysThisMonth();

}

*// add days*

day += n;

}

private int leftDaysThisMonth() {

boolean isLeapYear = isLeapYear(year);

return (isLeapYear ? LEAP\_MONTH[month] : MONTH[month]) - day;

}

private int leftDaysThisYear() {

int thisMonth = leftDaysThisMonth();

int nextMonths = 0;

boolean isLeapYear = isLeapYear(year);

for (int i = month + 1;i < MONTH.length;i++) {

nextMonths += isLeapYear ? LEAP\_MONTH[i] : MONTH[i];

}

return thisMonth + nextMonths;

}

public boolean isLeapYear(int year) {

if (year % 4 == 0 && year % 100 != 0 || year % 400 == 0) {

return true;

}

return false;

}

}

**随机算法**

380. Insert Delete GetRandom O(1)

Design a data structure that supports all following operations in average O(1) time.

insert(val): Inserts an item val to the set if not already present.

remove(val): Removes an item val from the set if present.

getRandom: Returns a random element from current set of elements. Each element must have the same probability of being returned.

**S1. HashMap + List**

map: <number, id of this number in list>

list: 下标是id，存的值是insert的数字。

remove的时候需要注意，如果直接remove，会导致后面的id 都不准，所以和list的最后一个swap一下。

class RandomizedSet {

Map<Integer, Integer> map;

List<Integer> nums;

Random rand;

*/\*\* Initialize your data structure here. \*/*

public RandomizedSet() {

map = new HashMap<>();

nums = new ArrayList<>();

rand = new Random();

}

*/\*\* Inserts a value to the set. Returns true if the set did not already contain the specified element. \*/*

public boolean insert(int val) {

if (map.containsKey(val)) return false;

map.put(val, nums.size());

nums.add(val);

return true;

}

*/\*\* Removes a value from the set. Returns true if the set contained the specified element. \*/*

public boolean remove(int val) {

if (!map.containsKey(val)) return false;

int idx = map.get(val);

if (idx != nums.size() - 1) {

int tmp = nums.get(nums.size() - 1);

nums.set(idx, tmp);

map.put(tmp, idx);

}

nums.remove(nums.size() - 1);

map.remove(val);

return true;

}

*/\*\* Get a random element from the set. \*/*

public int getRandom() {

return nums.get(rand.nextInt(nums.size()));

}

}

381. Insert Delete GetRandom O(1) - Duplicates allowed

Design a data structure that supports all following operations in average O(1) time.

Note: Duplicate elements are allowed.

**S1.HashMap + List**

和上一题的区别就是可以加入duplicate，所以map改用<number, <set of id>>来表示。

注意remove的时候需要用iterator来删除。

class RandomizedCollection {

List<Integer> nums;

HashMap<Integer, Set<Integer>> map;

Random rand;

*/\*\* Initialize your data structure here. \*/*

public RandomizedCollection() {

nums = new ArrayList<Integer>();

map = new HashMap<Integer, Set<Integer>>();

rand = new Random();

}

*/\*\* Inserts a value to the collection. Returns true if the collection did not already contain the specified element. \*/*

public boolean insert(int val) {

boolean contain = map.containsKey(val);

if (!contain) map.put(val, new LinkedHashSet<Integer>());

map.get(val).add(nums.size());

nums.add(val);

return !contain ;

}

*/\*\* Removes a value from the collection. Returns true if the collection contained the specified element. \*/*

public boolean remove(int val) {

boolean contain = map.containsKey(val);

if (!contain) return false;

int idx = map.get(val).iterator().next();

map.get(val).remove(idx);

if (idx < nums.size() - 1 ) {

int last = nums.get(nums.size() - 1);

nums.set(idx , last);

map.get(last).remove(nums.size() - 1);

map.get(last).add(idx);

}

nums.remove(nums.size() - 1);

if (map.get(val).isEmpty()) map.remove(val);

return true;

}

*/\*\* Get a random element from the collection. \*/*

public int getRandom() {

return nums.get(rand.nextInt(nums.size()) );

}

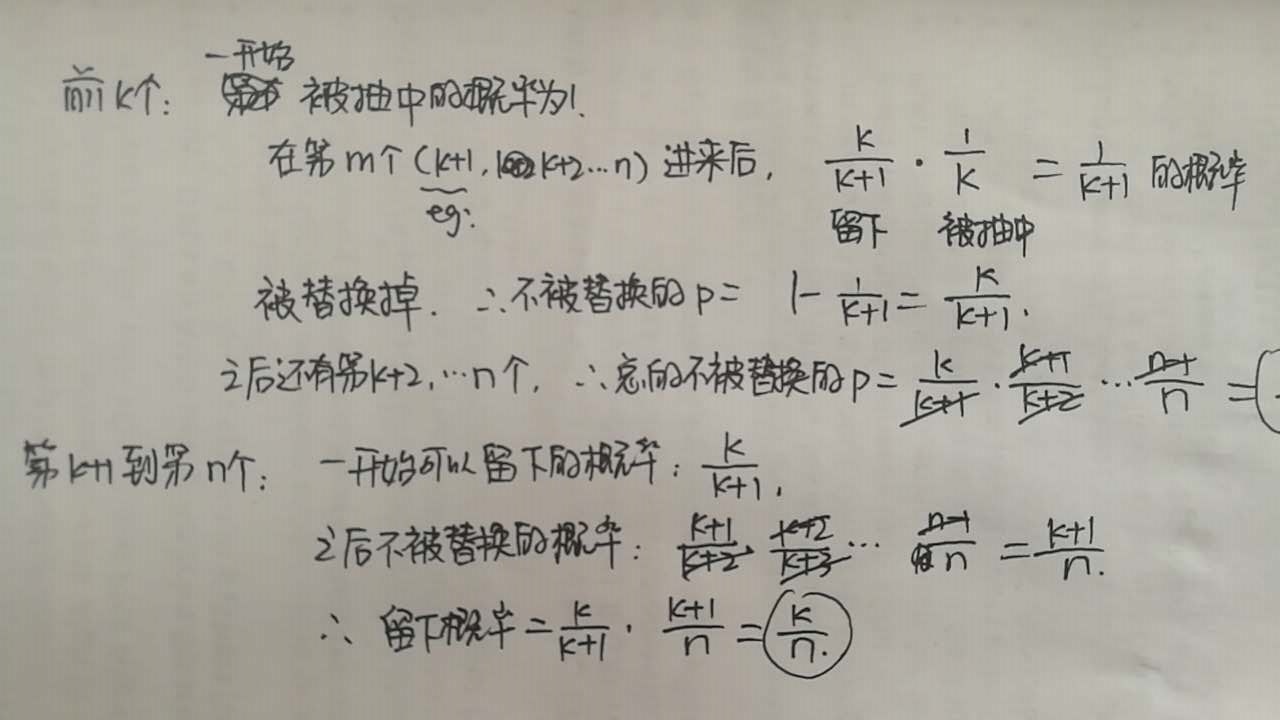
}

**Reservoir Sampling 蓄水池抽样**

面对长度未知的data stream, 随机抽取k个数据。

1. 选取前k个元素，保存在集合A中
2. 从第m = k + 1个元素开始，每次先以k/m的概率，选择是否让这个元素留下。
   1. 如果被选中留下，则从A中随机选择一个元素被它替换
   2. 没被选中留下，则continue
3. 重复2直到结束，集合中剩下的就是随机抽取的K个元素。

证明：



382. Linked List Random Node

Given a singly linked list, return a random node's value from the linked list. Each node must have the same probability of being chosen.

Follow up:

What if the linked list is extremely large and its length is unknown to you? Could you solve this efficiently without using extra space?

**S1. Reservoir Sampling**

这里的k为1

class Solution {

Random rand;

ListNode head;

*/\*\** ***@param*** *head The linked list's head.*

*Note that the head is guaranteed to be not null, so it contains at least one node. \*/*

public Solution(ListNode head) {

this.head = head;

rand = new Random();

}

*/\*\* Returns a random node's value. \*/*

public int getRandom() {

int count = 1;

ListNode cur = head;

ListNode chosen = head;

while (cur != null) {

if (rand.nextInt(count++) == 0) chosen = cur;

cur = cur.next;

}

return chosen.val;

}

}

398. Random Pick Index

Given an array of integers with possible duplicates, randomly output the index of a given target number. You can assume that the given target number must exist in the array.

Note: The array size can be very large. Solution that uses too much extra space will not pass the judge.

**S1 Reservoir Sampling**

一开始count = 1，所以nextInt的范围只有0，第一个数字一定会被选到，相当于是初始值。

之后会再根据概率来判断后面的数字是否留下并替换原来的数。

class Solution {

Random rand;

int[] nums;

public Solution(int[] nums) {

rand = new Random();

this.nums = nums;

}

public int pick(int target) {

int chosen = -1;

int size = nums.length;

int count = 1;

for (int i = 0;i < size;i++) {

if (nums[i] == target) {

if (rand.nextInt(count++) == 0) {

chosen = i;

}

}

}

return chosen;

}

}

Data Structure

232. Implement Queue using Stacks

Implement the following operations of a queue using stacks.

push(x) -- Push element x to the back of queue.

pop() -- Removes the element from in front of queue.

peek() -- Get the front element.

empty() -- Return whether the queue is empty.

很久以前做过的题目，发现原来有新的非常棒的方法，beat 96%+

**S1. 2 stacks 倒换**

每次要加入新的元素，就从input stack往里面加

如果要取出元素，就从output stack往外取。

把元素从input stack倒到output stack的过程，刚好就是调换顺序的过程，

所以每次output空的时候，就做一次倒换。

class MyQueue {

*// input: 1,2,3,4,5, output: --*

*// input: --, output: 5,4,3,2,1*

*// input: 6,7 output:*

Stack<Integer> input;

Stack<Integer> output;

*/\*\* Initialize your data structure here. \*/*

public MyQueue() {

input = new Stack();

output = new Stack();

}

public void push(int x) {

input.push(x);

}

public int pop() {

if (output.empty()) inputToOutput();

return output.pop();

}

public int peek() {

if (output.empty()) inputToOutput();

return output.peek();

}

private void inputToOutput() {

while (!input.empty()) {

output.push(input.pop());

}

}

public boolean empty() {

return input.empty() && output.empty();

}

}

225. Implement Stack using Queues

Implement the following operations of a stack using queues.

push(x) -- Push element x onto stack.

pop() -- Removes the element on top of the stack.

top() -- Get the top element.

empty() -- Return whether the stack is empty.

**题解：**

这里的思路就是，把element放入之后，把内部元素翻转一遍。

public class MyStack {

Queue<Integer> s;

*/\*\* Initialize your data structure here. \*/*

public MyStack() {

s = new LinkedList<>();

}

*/\*\* Push element x onto stack. \*/*

public void push(int x) {

s.add(x);

int size = s.size() - 1;

for (int i = 0;i < size;i++) {

s.add(s.poll());

}

}

*/\*\* Removes the element on top of the stack and returns that element. \*/*

public int pop() {

return s.poll();

}

*/\*\* Get the top element. \*/*

public int top() {

return s.peek();

}

*/\*\* Returns whether the stack is empty. \*/*

public boolean empty() {

return s.isEmpty();

}

}

155. Min Stack + 用List来implement Stack，（LinkedList, ArrayList）

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

push(x) -- Push element x onto stack.

pop() -- Removes the element on top of the stack.

top() -- Get the top element.

getMin() -- Retrieve the minimum element in the stack.

题解：

这里用了LinkedList, 因为有一系列API可以用，比如peekFirst, offerFirst, pollFirst, 而且stack只会对某一端的数据做处理，不会directly access中间某处的数据，所以linkedList比arraylist更合适。

通过Element class来实现min

public class MinStack {

LinkedList<Element> stack;

static class Element {

final int val;

final int min;

Element(int val, int min) {

this.val = val;

this.min = min;

}

}

*/\*\* initialize your data structure here. \*/*

public MinStack() {

stack = new LinkedList<>();

}

public void push(int x) {

int min = stack.isEmpty() ? x : Math.min(x, stack.peekFirst().min);

stack.offerFirst(new Element(x, min));

}

public void pop() {

stack.pollFirst();

}

public int top() {

return stack.peekFirst().val;

}

public int getMin() {

return stack.peekFirst().min;

}

}

S2. 用ArrayList来做

public class MinStack {

ArrayList<Element> stack;

static class Element {

final int val;

final int min;

Element(int val, int min) {

this.val = val;

this.min = min;

}

}

*/\*\* initialize your data structure here. \*/*

public MinStack() {

stack = new ArrayList<>();

}

public void push(int x) {

int min = stack.isEmpty() ? x : Math.min(x, stack.get(0).min);

stack.add(0, new Element(x, min));

}

public void pop() {

stack.remove(0);

}

public int top() {

return stack.get(0).val;

}

public int getMin() {

return stack.get(0).min;

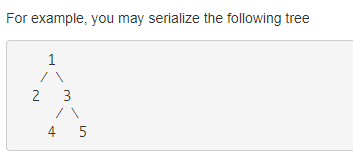
}

}

297. Serialize and Deserialize Binary Tree

Serialization is the process of converting a data structure or object into a sequence of bits so that it can be stored in a file or memory buffer, or transmitted across a network connection link to be reconstructed later in the same or another computer environment.

Design an algorithm to serialize and deserialize a binary tree. There is no restriction on how your serialization/deserialization algorithm should work. You just need to ensure that a binary tree can be serialized to a string and this string can be deserialized to the original tree structure.

as "[1,2,3,null,null,4,5]"

**S1. 就按照example的方法,**

通过Level order traversal来serialize, 先找到height, 在最后一行4,5这里不再加入null.

*// Encodes a tree to a single string.*

public static String serialize(TreeNode root) {

if (root == null) return "";

int height = getHeight(root);

StringBuilder sb = new StringBuilder();

Queue<TreeNode> q = new LinkedList<>();

q.add(root);

while (!q.isEmpty()) {

height--;

int curSize = q.size();

for (int i = 0;i < curSize;i++) {

TreeNode cur = q.poll();

if (cur == null) {

sb.append("null,");

} else {

sb.append(cur.val + ",");

// if at last level, not add “null” into sb, save some space

if (height != 0 || cur.left != null) q.add(cur.left);

if (height != 0 || cur.right != null) q.add(cur.right);

}

}

}

return sb.toString();

}

public static int getHeight(TreeNode root) {

if (root == null) return 0;

return Math.max(getHeight(root.left), getHeight(root.right)) + 1;

}

**Deserilize:**

首先通过”,” 来split

也用Queue，每次i = i+2, handle left, right，并把非null的node继续加入queue处理

*// Decodes your encoded data to tree.*

public static TreeNode deserialize(String data) {

if (data == null || data.length() == 0) return null;

String[] nodes = data.split(",");

TreeNode root = new TreeNode(Integer.parseInt(nodes[0]));

Queue<TreeNode> q = new LinkedList<>();

q.add(root);

*// handle left, right each time, so i+2*

for (int i = 1;i < nodes.length;i += 2) {

TreeNode parent = q.poll();

if (!nodes[i].equals("null")) {

TreeNode leftNode = new TreeNode(Integer.parseInt(nodes[i]));

parent.left = leftNode;

q.add(leftNode);

}

if (!nodes[i + 1].equals("null")){

TreeNode rightNode = new TreeNode(Integer.parseInt(nodes[i + 1]));

parent.right = rightNode;

q.add(rightNode);

}

}

return root;

}