BITTIGER

CLASS_7
OOD DESIGN

Ben Gong

Content of Class_7

Design
295. Find Median from Data Stream
460. LFU Cache
362. Design Hit Counter
Load Limit

Design 类题目小综合

运用各种数据结构解决实际问题

LinkedList

HashMap

HashSet

Trie Tree

Circular Array

Heap

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:

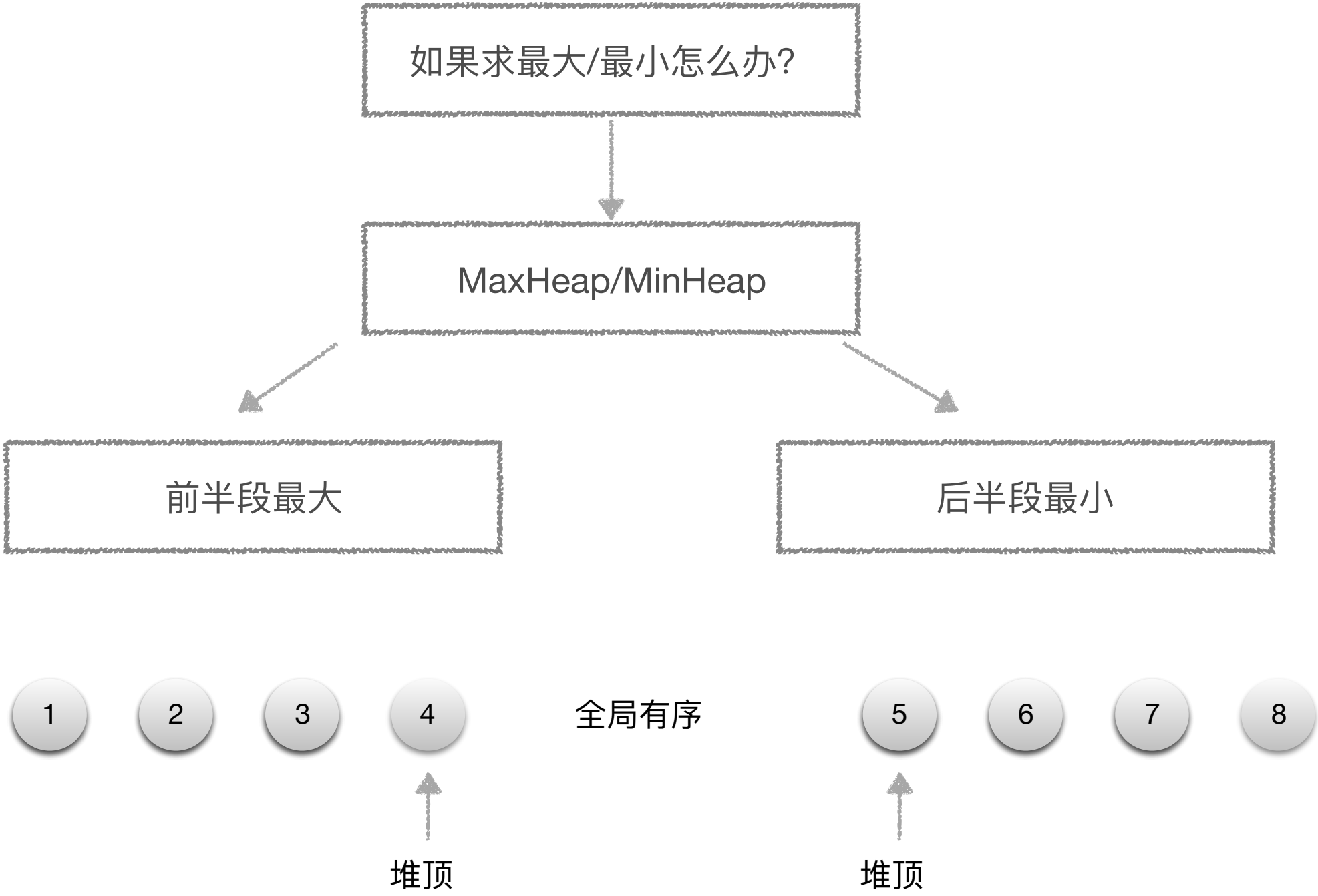
- void addNum(int num) - Add a integer number from the data stream to the data structure.
- double findMedian() - Return the median of all elements so far.

For example:

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

10 min

```
39    /** initialize your data structure here. */
40    public MedianFinder() {
41
42    }
43
44    public void addNum(int num) {
45
46    }
47
48    public double findMedian() {
49
50    }
```


如果求最大/最小怎么办?
MaxHeap/MinHeap
前半段最大
后半段最小
1
2
3
4
全局有序
5
6
7
8
堆顶
堆顶

```java
public class MyComparator implements Comparator<Integer> {
    @Override
    public int compare(Integer l1, Integer l2){
        // return l2 - l1;
        if(l2 > l1){
            return 1;
        } else if(l2 < l1){
            return -1;
        } else {
            return 0;
        }
        // return l2.compareTo(l1);
    }
}
public PriorityQueue<Integer> minHeap = new PriorityQueue<>();
public PriorityQueue<Integer> maxHeap = new PriorityQueue<>(new MyComparator());
/** initialize your data structure here. */
public MedianFinder() {

}

public void addNum(int num) {
    minHeap.add(num);
    maxHeap.add(minHeap.poll());
    if(maxHeap.size() > minHeap.size()){
        minHeap.add(maxHeap.poll());
    }
}

public double findMedian() {
    if(maxHeap.size() == minHeap.size()){
        return (double)(maxHeap.peek() + minHeap.peek()) / 2;
    }else{
        return (double)minHeap.peek();
    }
}
```

3种compare写法

Java中用priorityqueue实现heap

必须确保全局有序

460. LFU Cache

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

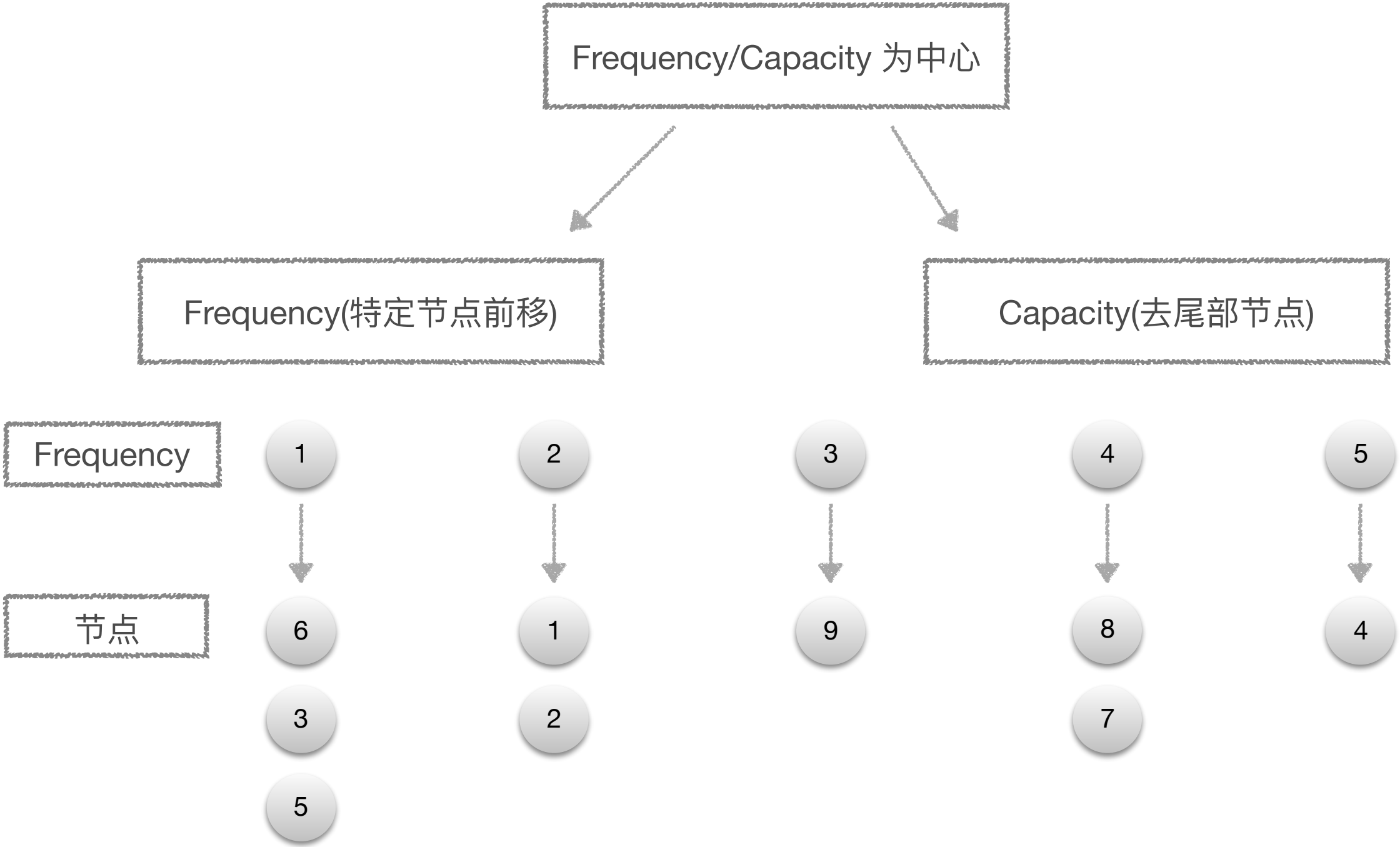
`get(key)` - Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.

`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.

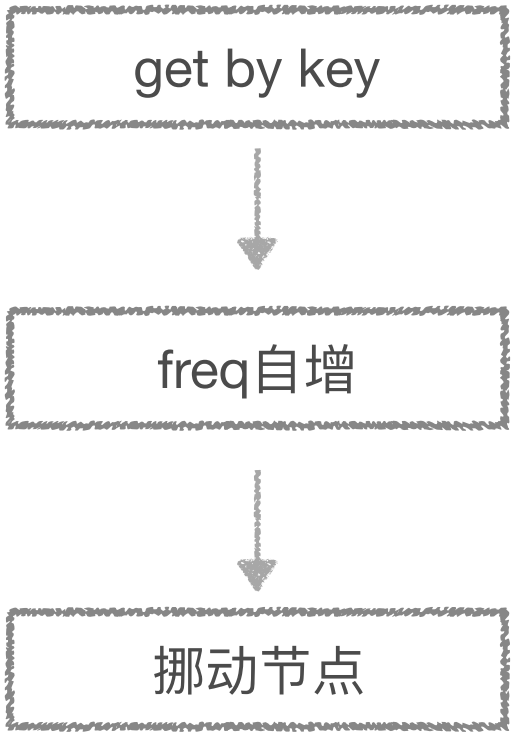
Follow up:

Could you do both operations in **O(1)** time complexity?

```
3  public class LFUCache {
4
5      public LFUCache(int capacity) {
6
7      }
8
9      public int get(int key) {
10
11     }
12
13     public void put(int key, int value) {
14
15     }
16 }
```


Frequency/Capacity 为中心
Frequency(特定节点前移)
Capacity(去尾部节点)
Frequency
节点
1
2
3
4
5
6
1
9
8
4
3
2
7
5

```java
HashMap<Integer, Integer> vals; // store key -- value
HashMap<Integer, Integer> freqs; // store frequency
HashMap<Integer, LinkedHashSet<Integer>> lists;  // store freq -- key
int capacity;
int minFreq = 1;

public LFUCache(int capacity) {
    this.capacity = capacity;
    this.vals = new HashMap<>();
    this.freqs = new HashMap<>();
    this.lists = new HashMap<>();
    this.lists.put(1, new LinkedHashSet<>());
}

public int get(int key) {

    if(!vals.containsKey(key)){
        return -1;
    }

    int freq = freqs.get(key);
    freqs.put(key, freq + 1);

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

    if(freq == minFreq && lists.get(freq).size() == 0){
        minFreq++;
    }

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

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

    return vals.get(key);
}
```



```
40  public void put(int key, int value) {
41      if(this.capacity <= 0){
42          return;
43      }
44
45      if(vals.containsKey(key)) {
46          vals.put(key, value);
47          get(key);
48          return;
49      }
50
51      if(vals.size() >= this.capacity) {
52          int evit = lists.get(minFreq).iterator().next();
53          lists.get(minFreq).remove(evit);
54          vals.remove(evit);
55      }
56
57      vals.put(key, value);
58      freqs.put(key, 1);
59      minFreq = 1;
60      lists.get(1).add(key);
61
62  }
```

利用get来自增freq 挪动节点

删除节点

362. Design Hit Counter

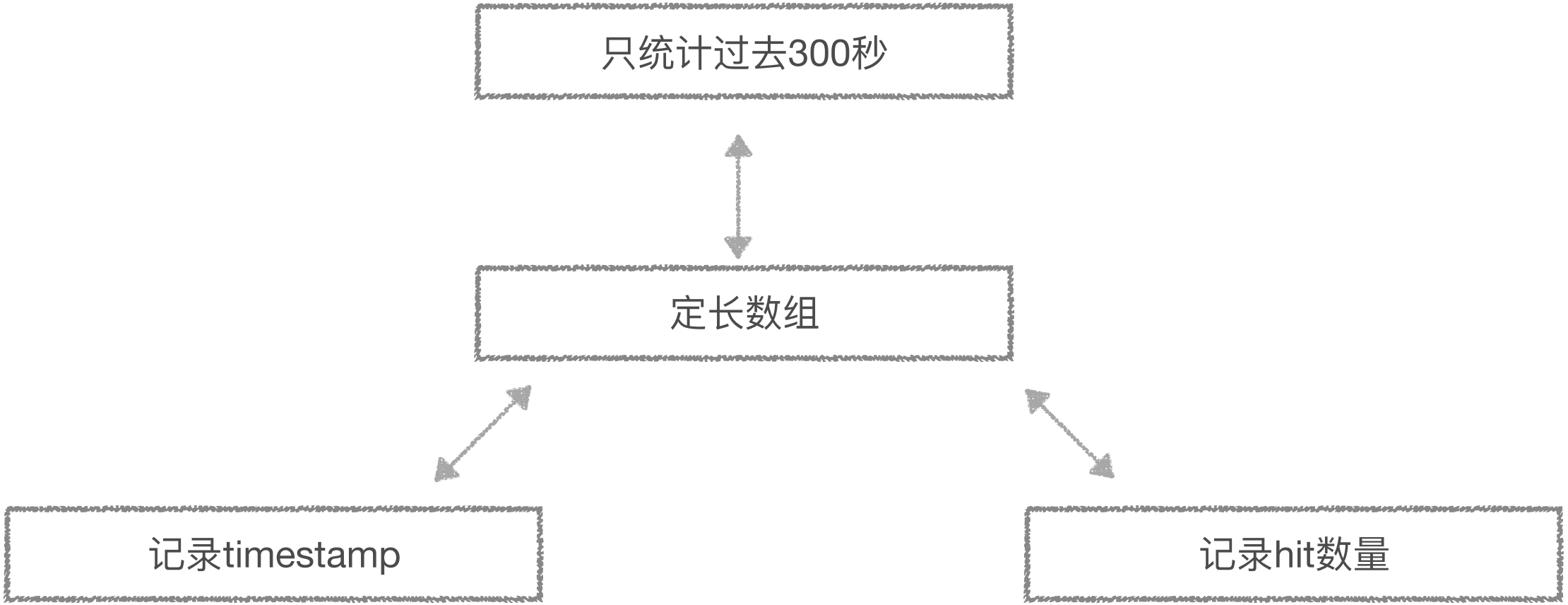
Design a hit counter which counts the number of hits received in the past 5 minutes.

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.

10 min

```
34  public class HitCounter {
35
36      /** Initialize your data structure here. */
37      public HitCounter() {
38
39      }
40
41      /** Record a hit.
42          @param timestamp - The current timestamp (in seconds granularity). */
43      public void hit(int timestamp) {
44
45      }
46
47      /** Return the number of hits in the past 5 minutes.
48          @param timestamp - The current timestamp (in seconds granularity). */
49      public int getHits(int timestamp) {
50
51      }
52  }
```


只统计过去300秒
定长数组
记录timestamp
记录hit数量

```java
int[] times;
int[] hits;
/** Initialize your data structure here. */
public HitCounter() {
    times = new int[300];
    hits = new int[300];
}

/** Record a hit.
    @param timestamp - The current timestamp (in seconds granularity). */
public void hit(int timestamp) {
    int index = timestamp % 300;
    if (times[index] != timestamp) {
        times[index] = timestamp;
        hits[index] = 1;
    } else {
        hits[index]++;
    }
}

/** Return the number of hits in the past 5 minutes.
    @param timestamp - The current timestamp (in seconds granularity). */
public int getHits(int timestamp) {
    int total = 0;
    for (int i = 0; i < 300; i++) {
        if (timestamp - times[i] < 300) {
            total += hits[i];
        }
    }
    return total;
}
```

刷新 timestamps 记录

统计hit 数量

Load Limit

Design a data structure that has the ability to limit N hits in last M seconds.

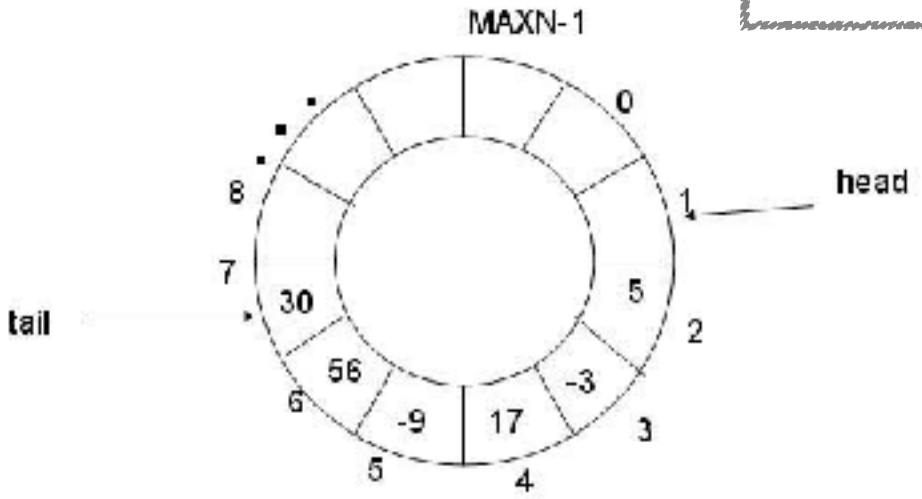
hit() if exceeds limit return false else return true

digest() if empty return false else return true and digest the first hit.

10 min

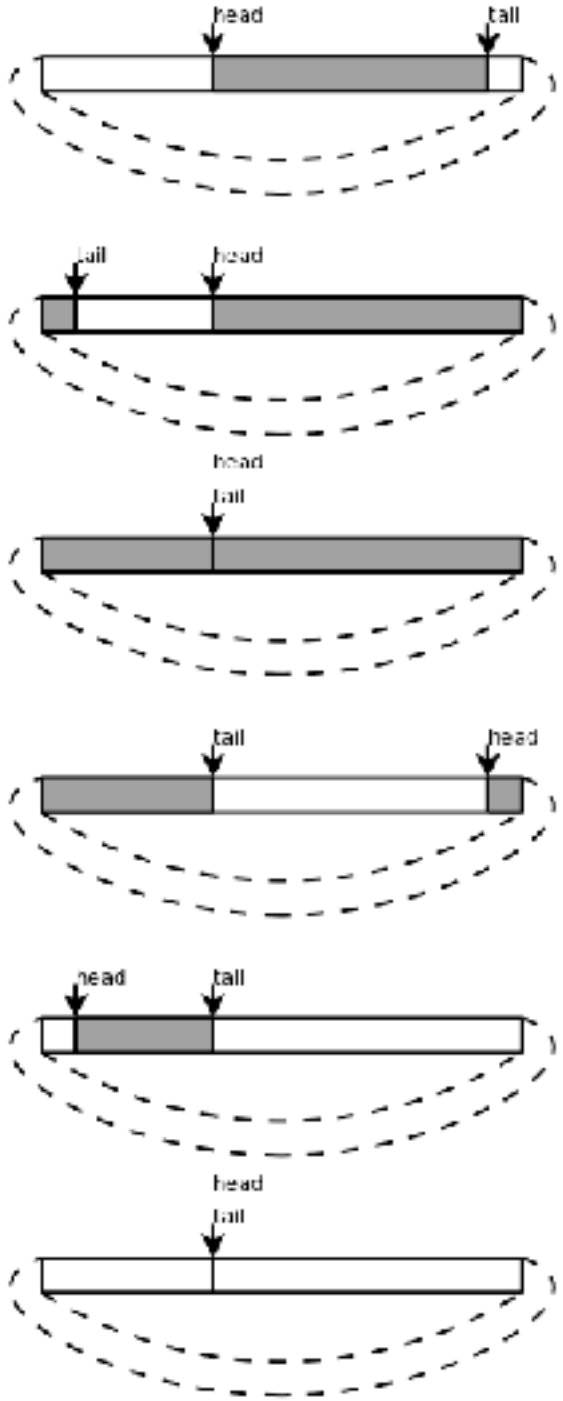
```
3   public LoadLimit(int timeLimit, int hitLimit){
4   }
5
6   public boolean hit(Token token){
7   }
8
9   public boolean digest(){
10  }
```

Circular Array/Buffer



当head赶上tail，队列空，则令 isFull = false

当tail赶上head，队列满，则令 isFull = true



```java
public static class Token{
    int timeStamp;
    int val;
    public Token(int timeStamp, int val){
        this.timeStamp = timeStamp;
        this.val = val;
    }
}

Token[] circularArray;
int beg;
int end;
boolean isFull;
int timeLimit;
int hitLimit;

public LoadLimit(int timeLimit, int hitLimit){
    this.circularArray = new Token[hitLimit];
    this.hitLimit = hitLimit;
    this.timeLimit = timeLimit;
    this.beg = 0;
    this.end = 0;
}
```

simulate Token

define circular array

```java
public boolean hit(Token token){
    if(isFull){
        //beg == end
        Token begToken = circularArray[beg];
        if(token.timeStamp - begToken.timeStamp >= timeLimit){
            circularArray[end] = token;
            end = (end + 1) % hitLimit;
            beg = end;
            isFull = true;
            return true;
        }else{
            return false;
        }
    }else{
        if(beg == end){
          // empty
            circularArray[beg] = token;
            end = (end + 1) % hitLimit;
            return true;
        }else{
          // normal case
            circularArray[end] = token;
            end = (end + 1) % hitLimit;
            if(beg == end){
                isFull = true;
            }
            return true;
        }

    }
}
```

exceed limit

rewrite previous token

```
57      public boolean digest(){
58          if(isFull){
59              circularArray[beg] = null;
60              beg = (beg + 1) % hitLimit;
61              isFull = false;
62              return true;
63          }else{
64              if(beg == end){
65                  // empty array
66                  return false;
67              }else{
68                  // normal case
69                  circularArray[beg] = null;
70                  beg = (beg + 1) % hitLimit;
71                  return true;
72              }
73          }
74      }
```

```java
public static void main(String[] args){

    LoadLimit loadLimit = new LoadLimit(4, 5);

    System.out.println("==== Test Case 1 ====");
    System.out.println(loadLimit.hit(new Token(1, 1)));
    System.out.println(loadLimit.hit(new Token(2, 2)));
    System.out.println(loadLimit.hit(new Token(3, 3)));
    System.out.println(loadLimit.hit(new Token(3, 4)));
    System.out.println(loadLimit.hit(new Token(4, 5)));
    // hit limit return false
    System.out.println(loadLimit.hit(new Token(4, 6)));
    // overwrite pre token return true
    System.out.println(loadLimit.hit(new Token(10, 7)));

    System.out.println("==== Test Case 2 ====");
    loadLimit = new LoadLimit(4, 5);
    System.out.println(loadLimit.hit(new Token(1, 1)));
    System.out.println(loadLimit.hit(new Token(2, 2)));
    System.out.println(loadLimit.hit(new Token(3, 3)));
    System.out.println(loadLimit.hit(new Token(3, 4)));
    System.out.println(loadLimit.digest());
    // un-hit limit return true
    System.out.println(loadLimit.hit(new Token(4, 6)));
    System.out.println(loadLimit.hit(new Token(4, 7)));
    // hit limit return false
    System.out.println(loadLimit.hit(new Token(4, 8)));
}
```

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_25.j
==== Test Case 1 ====
true
true
true
true
true
false
true
==== Test Case 2 ====
true
true
true
true
true
true
true
false

Process finished with exit code 0
```

Homework

Q & A

Thank you