128. Hash Function ☆

Description | Notes | >_ Testcase | Judge

In data structure Hash, hash function is used to convert a string(or any other type) into an integer smaller than hash size and bigger or equal to zero. The objective of designing a hash function is to "hash" the key as unreasonable as possible. A good hash function can avoid collision as less as possible. A widely used hash function algorithm is using a magic number 33, consider any string as a 33 based big integer like follow:

hashcode("abcd") = (ascii(a) * 33^3 + ascii(b) * 33^2 + ascii(c) *33 + ascii(d)) % HASH_SIZE

= (97* 33^3 + 98 * 33^2 + 99 * 33 +100) % HASH_SIZE

= 3595978 % HASH_SIZE

here HASH_SIZE is the capacity of the hash table (you can assume a hash table is like an array with index 0 ~ HASH_SIZE-1).

Given a string as a key and the size of hash table, return the hash value of this key.f

Clarification

For this problem, you are not necessary to design your own hash algorithm or consider any collision issue, you just need to implement the algorithm as described.

Example

For key="abcd" and size=100, return 78

Tags ▾

```
public class Solution {
    /*
     * @param key: A string you should hash
     * @param HASH_SIZE: An integer
     * @return: An integer
     */
    public int hashCode(char[] key, int HASH_SIZE) {
        // write your code here
                long ans = 0;
        for(int i = 0; i < key.length;i++) {
            ans = (ans * 33 + (int)(key[i])) % HASH_SIZE;
        }
    return (int)ans;
    }
}
```

613. High Five ☆

Description | Notes | >_ Testcase | Judge

There are two properties in the node student `id` and `scores`, to ensure that each student will have at least 5 points, find the average of 5 highest scores for each person.

Have you met this question in a real interview? Yes

Example

```
Given results = [[1,91],[1,92],[2,93],[2,99],[2,98],[2,97],[1,60],[1,58],
[2,100],[1,61]]

Return
```

Tags

```
/**
 * Definition for a Record
 * class Record {
 *     public int id, score;
 *     public Record(int id, int score){
 *         this.id = id;
 *         this.score = score;
 *     }
 * }
 */
public class Solution {
    /**
     * @param results a list of <student_id, score>
     * @return find the average of 5 highest scores for each person
     * Map<Integer, Double> (student_id, average_score)
     */
    public Map<Integer, Double> highFive(Record[] results) {
        Map<Integer, Double> answer = new HashMap<Integer, Double>();
        Map<Integer, PriorityQueue<Integer>> hash = new HashMap<Integer,
PriorityQueue<Integer>>();

        for (Record r : results) {
            if (!hash.containsKey(r.id)){
                hash.put(r.id, new PriorityQueue<Integer>());
            }
            PriorityQueue<Integer> pq=hash.get(r.id);
            if (pq.size() < 5) {
                pq.add(r.score);
            } else {
                if (pq.peek() < r.score){
                    pq.poll();
                    pq.add(r.score);
                }
            }
        }

        for (Map.Entry<Integer, PriorityQueue<Integer>> entry : hash.entrySet()) {
            int id = entry.getKey();
            PriorityQueue<Integer> scores = entry.getValue();
            double average = 0;
            for (int i = 0; i < 5; ++i)
                average += scores.poll();
            average /= 5.0;
            answer.put(id, average);
        }
        return answer;
    }
}
```

612. K Closest Points ☆

Description | Notes | >_ Testcase | Judge

Given some `points` and a point `origin` in two dimensional space, find `k` points out of the some points which are nearest to `origin`.
Return these points sorted by distance, if they are same with distance, sorted by x-axis, otherwise sorted by y-axis.

Have you met this question in a real interview? Yes

Example

Given points = `[[4,6],[4,7],[4,4],[2,5],[1,1]]`, origin = `[0, 0]`, k = `3`
return `[[1,1],[2,5],[4,4]]`

Tags

```
/**
 * Definition for a point.
 * class Point {
 *     int x;
 *     int y;
 *     Point() { x = 0; y = 0; }
 *     Point(int a, int b) { x = a; y = b; }
 * }
 */


public class Solution {
    /**
     * @param points a list of points
     * @param origin a point
     * @param k an integer
     * @return the k closest points
     */
    private Point global_origin = null;
    public Point[] kClosest(Point[] points, Point origin, int k) {
        // Write your code here
        global_origin = origin;
        PriorityQueue<Point> pq = new PriorityQueue<Point> (k, new Comparator<Point> () {
            @Override
            public int compare(Point a, Point b) {
                int diff = getDistance(b, global_origin) - getDistance(a, global_origin);
                if (diff == 0)
                    diff = b.x - a.x;
                if (diff == 0)
                    diff = b.y - a.y;
                return diff;
            }
        });

        for (int i = 0; i < points.length; i++) {
            pq.offer(points[i]);
            if (pq.size() > k)
                pq.poll();
        }

        k = pq.size();
        Point[] ret = new Point[k];
        while (!pq.isEmpty())
            ret[--k] = pq.poll();
        return ret;
    }

    private int getDistance(Point a, Point b) {
```

```
        return (a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y);
    }
}
```

606. Kth Largest Element II ☆

Description | Notes | >_ Testcase | Judge

Find K-th largest element in an array. and N is much larger than k.

Notice

You can swap elements in the array

Have you met this question in a real interview? Yes

Example

In array `[9,3,2,4,8]`, the `3rd` largest element is `4`.

In array `[1,2,3,4,5]`, the `1st` largest element is `5`, `2nd` largest element is `4`, `3rd` largest element is `3` and etc.

Tags

```
public class Solution {
    /*
     * @param nums: an integer unsorted array
     * @param k: an integer from 1 to n
     * @return: the kth largest element
     */
    public int kthLargestElement2(int[] nums, int k) {
        // write your code here
        PriorityQueue<Integer> q = new PriorityQueue<Integer>(k);
        for (int num : nums) {
            q.offer(num);
            if (q.size() > k) {
                q.poll();
            }
        }
        return q.peek();
    }
}
```

544. Top k Largest Numbers ☆

Description | Notes | >_ Testcase | Judge

Given an integer array, find the top *k* largest numbers in it.

Have you met this question in a real interview? Yes

Example

Given `[3,10,1000,-99,4,100]` and *k* = `3`.
Return `[1000, 100, 10]`.

Tags

```
// base on heap
class Solution {
    /*
     * @param nums an integer array
     * @param k an integer
     * @return the top k largest numbers in array
     */
     public int[] topk(int[] nums, int k) {
         PriorityQueue<Integer> minheap = new PriorityQueue<Integer>(k, new
Comparator<Integer>() {
             public int compare(Integer o1, Integer o2) {
                 return o1 - o2;
             }
         });

         for (int i : nums) {
             minheap.add(i);
             if (minheap.size() > k) {
                minheap.poll();
             }
         }

         int[] result = new int[k];
         for (int i = 0; i < result.length; i++) {
             result[k - i - 1] = minheap.poll();
         }
         return result;
    }
}
```

```
// base on quicksort
import java.util.Random;

class Solution {
    /*
     * @param nums an integer array
     * @param k an integer
     * @return the top k largest numbers in array
     */
```

```
public int[] topk(int[] nums, int k) {
    // Write your code here
    quickSort(nums, 0, nums.length - 1, k);

    int[] topk = new int[k];
    for (int i = 0; i < k; ++i)
        topk[i] = nums[i];

    return topk;
}

private void quickSort(int[] A, int start, int end, int k) {
    if (start >= k)
        return;

    if (start >= end) {
        return;
    }

    int left = start, right = end;
    // key point 1: pivot is the value, not the index
    Random rand =new Random(end - start + 1);
    int index = rand.nextInt(end - start + 1) + start;
    int pivot = A[index];

    // key point 2: every time you compare left & right, it should be
    // left <= right not left < right
    while (left <= right) {
        // key point 3: A[left] < pivot not A[left] <= pivot
        while (left <= right && A[left] > pivot) {
            left++;
        }
        // key point 3: A[right] > pivot not A[right] >= pivot
        while (left <= right && A[right] < pivot) {
            right--;
        }

        if (left <= right) {
            int temp = A[left];
            A[left] = A[right];
            A[right] = temp;

            left++;
            right--;
        }
    }

    quickSort(A, start, right, k);
    quickSort(A, left, end, k);
```

```
    }
};
```

129. Rehashing ☆

Description | Notes | >_ Testcase | Judge

The size of the hash table is not determinate at the very beginning. If the total size of keys is too large (e.g. size >= capacity / 10), we should double the size of the hash table and rehash every keys. Say you have a hash table looks like below:

`size=3`, `capacity=4`

```
[null, 21, 14, null]
       ↓    ↓
       9   null
       ↓
      null
```

The hash function is:

```
int hashcode(int key, int capacity) {
    return key % capacity;
}
```

here we have three numbers, 9, 14 and 21, where 21 and 9 share the same position as they all have the same hashcode 1 (21 % 4 = 9 % 4 = 1). We store them in the hash table by linked list.

rehashing this hash table, double the capacity, you will get:

`size=3`, `capacity=8`

```
index:   0    1    2    3     4    5    6   7
hash : [null, 9, null, null, null, 21, 14, null]
```

Given the original hash table, return the new hash table after rehashing .

i Notice

For negative integer in hash table, the position can be calculated as follow:

- **C++/Java**: if you directly calculate -4 % 3 you will get -1. You can use function: a % b = (a % b + b) % b to make it is a non negative integer.
- **Python**: you can directly use -1 % 3, you will get 2 automatically.

```
/**
 * Definition for ListNode
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    /**
     * @param hashTable: A list of The first node of linked list
     * @return: A list of The first node of linked list which have twice size
     */
    public ListNode[] rehashing(ListNode[] hashTable) {
        // write your code here
            if (hashTable.length <= 0) {
            return hashTable;
        }
        int newcapacity = 2 * hashTable.length;
        ListNode[] newTable = new ListNode[newcapacity];
        for (int i = 0; i < hashTable.length; i++) {
            while (hashTable[i] != null) {
                int newindex
                 = (hashTable[i].val % newcapacity + newcapacity) % newcapacity;
                if (newTable[newindex] == null) {
                    newTable[newindex] = new ListNode(hashTable[i].val);
                   // newTable[newindex].next = null;
                } else {
                    ListNode dummy = newTable[newindex];
                    while (dummy.next != null) {
                        dummy = dummy.next;
                    }
                    dummy.next = new ListNode(hashTable[i].val);
                }
                hashTable[i] = hashTable[i].next;
            }
        }
        return newTable;
    }
};
```

104. Merge k Sorted Lists ☆

Description | Notes | >_ Testcase | Judge

Merge *k* sorted linked lists and return it as one sorted list.

Analyze and describe its complexity.

Have you met this question in a real interview? Yes

Example

Given lists:

```
[
  2->4->null,
  null,
  -1->null
],
```

return `-1->2->4->null`.

```
// version 1: Divide & Conquer
/**
 * Definition for ListNode.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int val) {
 *         this.val = val;
 *         this.next = null;
 *     }
 * }
 */
public class Solution {
    /**
     * @param lists: a list of ListNode
     * @return: The head of one sorted list.
     */
    public ListNode mergeKLists(List<ListNode> lists) {
        if (lists.size() == 0) {
            return null;
        }
        return mergeHelper(lists, 0, lists.size() - 1);
    }

    private ListNode mergeHelper(List<ListNode> lists, int start, int end) {
        if (start == end) {
            return lists.get(start);
        }

        int mid = start + (end - start) / 2;
        ListNode left = mergeHelper(lists, start, mid);
        ListNode right = mergeHelper(lists, mid + 1, end);
        return mergeTwoLists(left, right);
    }

    private ListNode mergeTwoLists(ListNode list1, ListNode list2) {
        ListNode dummy = new ListNode(0);
        ListNode tail = dummy;
        while (list1 != null && list2 != null) {
            if (list1.val < list2.val) {
                tail.next = list1;
                tail = list1;
                list1 = list1.next;
            } else {
```

```
            tail.next = list2;
            tail = list2;
            list2 = list2.next;
        }
    }
    if (list1 != null) {
        tail.next = list1;
    } else {
        tail.next = list2;
    }

    return dummy.next;
  }
}

// version 2: Heap
public class Solution {
  private Comparator<ListNode> ListNodeComparator = new Comparator<ListNode>() {
    public int compare(ListNode left, ListNode right) {
      return left.val - right.val;
    }
  };

  public ListNode mergeKLists(List<ListNode> lists) {
    if (lists == null || lists.size() == 0) {
      return null;
    }

    Queue<ListNode> heap = new PriorityQueue<ListNode>(lists.size(), ListNodeComparator);
    for (int i = 0; i < lists.size(); i++) {
      if (lists.get(i) != null) {
        heap.add(lists.get(i));
      }
    }

    ListNode dummy = new ListNode(0);
    ListNode tail = dummy;
    while (!heap.isEmpty()) {
      ListNode head = heap.poll();
      tail.next = head;
      tail = head;
      if (head.next != null) {
        heap.add(head.next);
      }
    }
    return dummy.next;
  }
}
```

```
// Version 3: merge two by two
/**
 * Definition for ListNode.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int val) {
 *         this.val = val;
 *         this.next = null;
 *     }
 * }
 */
public class Solution {
    /**
     * @param lists: a list of ListNode
     * @return: The head of one sorted list.
     */
    public ListNode mergeKLists(List<ListNode> lists) {
        if (lists == null || lists.size() == 0) {
            return null;
        }

        while (lists.size() > 1) {
            List<ListNode> new_lists = new ArrayList<ListNode>();
            for (int i = 0; i + 1 < lists.size(); i += 2) {
                ListNode merged_list = merge(lists.get(i), lists.get(i+1));
                new_lists.add(merged_list);
            }
            if (lists.size() % 2 == 1) {
                new_lists.add(lists.get(lists.size() - 1));
            }
            lists = new_lists;
        }

        return lists.get(0);
    }

    private ListNode merge(ListNode a, ListNode b) {
        ListNode dummy = new ListNode(0);
        ListNode tail = dummy;
        while (a != null && b != null) {
            if (a.val < b.val) {
                tail.next = a;
                a = a.next;
            } else {
                tail.next = b;
                b = b.next;
            }
            tail = tail.next;
```

```
        }

        if (a != null) {
            tail.next = a;
        } else {
            tail.next = b;
        }

        return dummy.next;
    }
}
```

4. Ugly Number II ☆

Description | Notes | >_ Testcase | Judge

Ugly number is a number that only have factors `2`, `3` and `5`.

Design an algorithm to find the *n*th ugly number. The first 10 ugly numbers are `1, 2, 3, 4, 5, 6, 8, 9, 10, 12...`

Notice

Note that `1` is typically treated as an ugly number.

Have you met this question in a real interview? Yes

Example

If `n=9`, return `10`.

```
// version 1: O(n) scan
class Solution {
    /**
     * @param n an integer
     * @return the nth prime number as description.
     */
    public int nthUglyNumber(int n) {
        List<Integer> uglys = new ArrayList<Integer>();
        uglys.add(1);

        int p2 = 0, p3 = 0, p5 = 0;
        // p2, p3 & p5 share the same queue: uglys

        for (int i = 1; i < n; i++) {
            int lastNumber = uglys.get(i - 1);
            while (uglys.get(p2) * 2 <= lastNumber) p2++;
            while (uglys.get(p3) * 3 <= lastNumber) p3++;
            while (uglys.get(p5) * 5 <= lastNumber) p5++;

            uglys.add(Math.min(
                Math.min(uglys.get(p2) * 2, uglys.get(p3) * 3),
                uglys.get(p5) * 5
            ));
        }

        return uglys.get(n - 1);
    }
};

// version 2 O(nlogn) HashMap + Heap
class Solution {
    /**
     * @param n an integer
     * @return the nth prime number as description.
     */
    public int nthUglyNumber(int n) {
        // Write your code here
        Queue<Long> Q = new PriorityQueue<Long>();
        HashSet<Long> inQ = new HashSet<Long>();
        Long[] primes = new Long[3];
        primes[0] = Long.valueOf(2);
        primes[1] = Long.valueOf(3);
        primes[2] = Long.valueOf(5);
```

```
        for (int i = 0; i < 3; i++) {
            Q.add(primes[i]);
            inQ.add(primes[i]);
        }
        Long number = Long.valueOf(1);
        for (int i = 1; i < n; i++) {
            number = Q.poll();
            for (int j = 0; j < 3; j++) {
                if (!inQ.contains(primes[j] * number)) {
                    Q.add(number * primes[j]);
                    inQ.add(number * primes[j]);
                }
            }
        }
        return number.intValue();
    }
};
```

594. strStr II ☆

Description | Notes | >_ Testcase | Judge

Implement `strStr` function in O(n + m) time.

`strStr` return the first index of the target string in a source string. The length of the target string is *m* and the length of the source string is *n*.
If target does not exist in source, just return -1.

Have you met this question in a real interview? Yes

Example

Given source = `abcdef`, target = `bcd`, return `1`.

Tags

```
public class Solution {
    /**
     * @param source a source string
     * @param target a target string
     * @return an integer as index
     */
    public int strStr2(String source, String target) {
        if(target == null) {
            return -1;
        }
        int m = target.length();
        if(m == 0 && source != null) {
            return 0;
        }

        if(source == null) {
            return -1;
        }
        int n = source.length();
        if(n == 0) {
            return -1;
        }

        // mod could be any big integer
        // just make sure mod * 33 wont exceed max value of int.
        int mod = Integer.MAX_VALUE / 33;
        int hash_target = 0;

        // 33 could be something else like 26 or whatever you want
        for (int i = 0; i < m; ++i) {
            hash_target = (hash_target * 33 + target.charAt(i) - 'a') % mod;
            if (hash_target < 0) {
                hash_target += mod;
            }
        }

        int m33 = 1;
        for (int i = 0; i < m - 1; ++i) {
            m33 = m33 * 33 % mod;
        }

        int value = 0;
        for (int i = 0; i < n; ++i) {
            if (i >= m) {
                value = (value - m33 * (source.charAt(i - m) - 'a')) % mod;
```

```
            }

            value = (value * 33 + source.charAt(i) - 'a') % mod;
            if (value < 0) {
                value += mod;
            }

            if (i >= m - 1 && value == hash_target) {
                // you have to double check by directly compare the string
                if (target.equals(source.substring(i - m + 1, i + 1))) {
                    return i - m + 1;
                }
            }
        }
        return -1;
    }
}
```

134. LRU Cache ☆

Description | Notes | Testcase | Judge

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

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

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

Have you met this question in a real interview? Yes

```
public class LRUCache {
    private class Node{
        Node prev;
        Node next;
        int key;
        int value;

        public Node(int key, int value) {
            this.key = key;
            this.value = value;
            this.prev = null;
            this.next = null;
        }
    }

    private int capacity;
    private HashMap<Integer, Node> hs = new HashMap<Integer, Node>();
    private Node head = new Node(-1, -1);
    private Node tail = new Node(-1, -1);
    /*
    * @param capacity: An integer
    */
    public LRUCache(int capacity) {
            // do intialization if necessary

        this.capacity = capacity;
        tail.prev = head;
        head.next = tail;
    }
    /*
     * @param key: An integer
     * @return: An integer
     */
    public int get(int key) {
        if( !hs.containsKey(key)) {
            return -1;
        }

        // remove current
        Node current = hs.get(key);
        current.prev.next = current.next;
        current.next.prev = current.prev;

        // move current to tail
        move_to_tail(current);

        return hs.get(key).value;
```

```
    }

    /*
     * @param key: An integer
     * @param value: An integer
     * @return: nothing
     */

    public void set(int key, int value) {
        // get 这个方法会把key挪到最末端，因此，不需要再调用 move_to_tail
        if (get(key) != -1) {
            hs.get(key).value = value;
            return;
        }

        if (hs.size() == capacity) {
            hs.remove(head.next.key);
            head.next = head.next.next;
            head.next.prev = head;
        }

        Node insert = new Node(key, value);
        hs.put(key, insert);
        move_to_tail(insert);
    }

    private void move_to_tail(Node current) {
        current.prev = tail.prev;
        tail.prev = current;
        current.prev.next = current;
        current.next = tail;
    }
}
```