算法

一、排序算法

1.1、冒泡排序

- **基本思想**
    - 从左到右依次比较相邻元素，通过交换使较大数在后方，每轮可使最大数“冒泡”到最后端。
- **代码实现**

```
public void bubbleSort(int[] nums) {
    for (int i = nums.length - 1; i >= 1 ; i--) {
        for (int j = 1; j <= i; j++) {
            if (nums[j] < nums[j - 1])
                swap(nums, j, j - 1);
        }
    }
}
```

- **复杂度**
    - 时间复杂度：O(n^2)，空间复杂度：O(1)，稳定

1.2、选择排序

- **基本思想**
    - 类似冒泡，每轮都找到未排序部分最大的值通过swap放到最后
- **代码实现**

```
private void selectionSort(int[] nums) {
  for (int i = nums.length - 1; i > 0; i--) {
      int maxIndex = 0;
      for (int j = 1; j <= i; j++) {
          if (nums[maxIndex] < nums[j])
              maxIndex = j;
      }
      swap(nums, maxIndex, i);
  }
}
```

- **复杂度**
    - 时间复杂度：O(n^2)，空间复杂度：O(1)，不稳定（可通过讲swap改为插入来使算法稳定）

1.3、快速排序

- **基本思想**
    - 取第一个元素（或最后一个元素）作为分界点，把整个数组分成左右两侧，左边的元素小于等于分界点，右边的元素大于分界点，然后把分界点移到中间位置，对左右子数组分别进行递归，最后就能得到一个排序完成的数组。当子数组只有一个或者没有元素的时候就结束这个递归过程。
- **代码实现**

```
private void quickSort(int[] nums, int left, int right) {
    if (left >= right) return;
    int lo = left + 1;
    int hi = right;
    while (lo <= hi) {
        if (nums[left] < nums[lo]) {
            swap(nums, lo, hi);
            hi--;
        } else {
            lo++;
        }
    }
    lo--;
    swap(nums, left, lo);
    quickSort(nums, left, lo - 1);
    quickSort(nums, lo + 1, right);
}
```

- **复杂度**
    - 时间复杂度：平均O(nlogn) 最坏O(n^2)，空间复杂度：平均O(logn) 最坏O(n)，不稳定

1.4、堆排序

二、动态规划

- 动态规则（Dynamic Programming，简称DP），虽然抽象后进行求解的思路并不复杂，但具体的形式千差万别，找出问题的子结构以及通过子结构重新构造最优解的过程很难统一，

2.1、连续子数组的最大和

- 问题描述
    - 输入一个整型数组，数组里有正数也有负数。数组中的一个或连续多个整数组成一个子数组。求所有子数组的和的最大值。
    - 要求时间复杂度为O(n)。
- 解题思路
    - 依次求出以某个值结尾的子数组中的最大值
    - 如果第i个数大于0，则i位之前最大的子数组为max(i-1)+i，小于0则不加，或者+0。
- 代码实现

```
// 时间复杂度o(n)
class Solution {
    public int maxSubArray(int[] nums) {
        int res = nums[0];
        for(int i = 1; i < nums.length; i++) {
            nums[i] += Math.max(nums[i - 1], 0);
            res = Math.max(res, nums[i]);
        }
        return res;
    }
}
```

- 题目来源
    - https://leetcode-cn.com/problems/lian-xu-zi-shu-zu-de-zui-da-he-lcof/solution/mian-shi-ti-42-lian-xu-zi-shu-zu-de-zui-da-he-do-2/

2.2、爬楼梯问题

- 问题描述
    - 小明上楼梯，一次能上1阶或者2阶，那么爬上n阶需要多久。
- 解题思路
    - 第 i 阶可以由以下两种方法得到：
    - 在第 (i-1) 阶后向上爬一阶。
    - 在第 (i-2) 阶后向上爬 2 阶。
    - 所以到达第 ii 阶的方法总数就是到第 (i-1)阶和第 (i-2)阶的方法数之和。
- 代码实现

```java
// 时间复杂度o(n)
static class Palouti{
    public int getRes(int n){
        if (n == 1) {
            return 1;
        }
        int[] dp = new int[n + 1];
        dp[1] = 1;
        dp[2] = 2;
        for (int i = 3; i <= n; i++) {
            dp[i] = dp[i - 1] + dp[i - 2];
        }
        return dp[n];
    }
}
```

- 题目来源
    - https://leetcode-cn.com/problems/climbing-stairs/solution/pa-lou-ti-by-leetcode/

2.3、母牛生产

- 问题描述
    - 假设农场中成熟的母牛每年都会生 1 头小母牛，并且永远不会死。第一年有 1 只小母牛，从第二年开始，母牛开始生小母牛。每只小母牛 3 年之后成熟又可以生小母牛。给定整数 N，求 N 年后牛的数量。
- 解题思路
    - 第 i 年成熟的牛的数量为
    - 第i-1年的母牛数量 + 在第i年能生小牛的母牛数量（即i-3年的母牛数量）
    - 所以第 i 年牛的数量为 (i-1)和 (i-3)的数量合。
- 代码实现

```
static class Palouti{
    public int getRes(int n){
        if (n == 1) {
            return 1;
        }
        int[] dp = new int[n + 1];
        dp[1] = 1;
        dp[2] = 2;
        dp[3] = 3;
        for (int i = 4; i <= n; i++) {
            dp[i] = dp[i - 1] + dp[i - 3];
        }
        return dp[n];
    }
}
```

- 题目来源

2.4、强盗抢劫

- 问题描述
    - 抢劫一排住户，但是不能抢邻近的住户，求最大抢劫量。
- 解题思路
    - 定义 dp 数组用来存储最大的抢劫量，其中 dp[i] 表示抢到第 i 个住户时的最大抢劫量。
    - 由于不能抢劫邻近住户，如果抢劫了第 i -1 个住户，那么就不能再抢劫第 i 个住户，所以

      dp[i] = max(dp[i-2] + nums[i], dp[i-1])
- 代码实现

```
//时间复杂度o(n)
public int rob(int[] nums) {
  int pre2 = 0, pre1 = 0;
  for (int i = 0; i < nums.length; i++) {
      int cur = Math.max(pre2 + nums[i], pre1);
      pre2 = pre1;
      pre1 = cur;
  }
  return pre1;
}
```

2.5、信件错排

- 问题描述
    - 有 N 个 信 和 信封，它们被打乱，求错误装信方式的数量。
- 解题思路
    - 定义一个数组 dp 存储错误方式数量，dp[i] 表示前 i 个信和信封的错误方式数量。假设第 i 个信装到第 j 个信封里面，而第 j 个信装到第 k 个信封里面。根据 i 和 k 是否相等，有两种情况：
    - i==k，交换 i 和 j 的信后，它们的信和信封在正确的位置，但是其余 i-2 封信有 dp[i-2] 种错误装信的方式。由于 j 有 i-1 种取值，因此共有 (i-1)*dp[i-2] 种错误装信方式。
    - i != k，交换 i 和 j 的信后，第 i 个信和信封在正确的位置，其余 i-1 封信有 dp[i-1] 种错误装信方式。由于 j 有 i-1 种取值，因此共有 (i-1)*dp[i-1] 种错误装信方式。
    - 综上所述，错误装信数量方式数量为: dp[i]=(i-1)*dp[i-2]+(i-1)*dp[i-1]
- 代码实现

```
static class Palouti{
    public int getRes(int n){
        if (n == 1) {
            return 1;
        }
        int[] dp = new int[n + 1];
        dp[2] = 1;
        dp[3] = 4;
        for (int i = 4; i <= n; i++) {
            dp[i] = (i-1) * dp[i - 2] + (i-1) * dp[i - 1];
        }
        return dp[n];
    }
}
```

- 题目来源

2.6、矩阵的总路径数

- 问题描述
    - 统计从矩阵左上角到右下角的路径总数，每次只能向右或者向下移动。
- 解题思路
    - 终点的左边那个点(i-1,j)和上面那个点(i,j-1)，都可以一步到达终点，没

有其他的选择。所以到达最后一个点(i,j)的方式，就是到达(i-1,j)的所有方式和到达(i,j-1)的所有方式之和。

- 代码实现

```
class Solution1 {
  public int uniquePaths(int m, int n) {
      int[][] dp = new int[m][n];
      for (int i = 0; i < n; i++) {
        dp[0][i] = 1;
      }
      for (int i = 0; i < m; i++) {
        dp[i][0] = 1;
      }
      for (int i = 1; i < m; i++) {
          for (int j = 1; j < n; j++) {
              dp[i][j] = dp[i - 1][j] + dp[i][j - 1];
          }
      }
      return dp[m - 1][n - 1];
  }
}

//前点为上面点加左边点，那么计算第二行时可以直接在第一行计算，及当前点加左边点就ok了，当前点原来
class Solution2 {
  public int uniquePaths(int m, int n) {
      int[] cur = new int[n];
      Arrays.fill(cur,1);
      for (int i = 1; i < m;i++){
          for (int j = 1; j < n; j++){
              cur[j] += cur[j-1] ;
          }
      }
      return cur[n-1];
  }
}
```

- 题目来源

三、贪心算法

3.1、田忌赛马问题

- 问题描述
    - 不多解释
- 解题思路
    - 每次拿A的“当前轮次“的最小值和B的”当前轮次“最小值比较，
    - 若大于，则OK，满足。
    - 若小于，则将A的值去和B的”最大值“配对。
- 代码实现

```
private int[] advantageCount2(int[] A, int[] B) {
      int[] res = new int[A.length];

      Arrays.sort(A);
      LinkedList<Node> listB = new LinkedList<>();
      for(int i = 0; i < B.length; i++){
          listB.add(new Node(B[i],i));
      }
      Collections.sort(listB,new Comparator<Node>(){
          public int compare(Node n1, Node n2){
              return n1.value - n2.value;
          }
      });

      // 遍历A即可，将B数组作为输出容器，因为B的信息已经都存在LinkedList里了，这里B数组已经没
      for(int i = 0; i < A.length; i++){
          if(A[i] > listB.getFirst().value){
              Node node = listB.removeFirst();
              res[node.index] = A[i];
          }else{
              Node node = listB.removeLast();
              res[node.index] = A[i];
          }
      }
      return res;
  }

  class Node{
      int value;
      int index;
      public Node(int value, int index){
          this.value = value;
          this.index = index;
      }
  }
```

- 题目来源
    - https://leetcode-cn.com/problems/advantage-shuffle/solution/java-qing-xi-ti-jie-by-jachindu2018/

3.2、买股票问题I

- 题目描述
    - 一次股票交易包含买入和卖出，只进行一次交易，求最大收益。

- 解题思路
    - 只要记录 i-1 的最小价格，将这个最小价格作为买入价格，然后将当前的价格 i 作为售出价格，查看当前收益是不是最大收益。
    - 从第 i 天（这里 i >= 1）开始，与第 i - 1 的股价进行比较，如果股价有上升（严格上升），就将升高的股价（ prices[i] - prices[i- 1] ）记入总利润，按照这种算法，得到的结果就是符合题意的最大利润。
- 代码实现

```
public int maxProfit(int[] prices) {
  int n = prices.length;
  if (n == 0) {
    return 0;
  }
  int soFarMin = prices[0];
  int max = 0;
  for (int i = 1; i < n; i++) {
      if (soFarMin > prices[i]) {
        soFarMin = prices[i];
      } else {
        max = Math.max(max, prices[i] - soFarMin);
      }
  }
  return max;
}
```

- 题目来源
    - https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock-ii/

3.3、买股票问题2

- 问题描述
    - 可以进行多次交易，多次交易之间不能交叉进行
- 解题思路
    - 对于 [a, b, c, d]，如果有 a <= b <= c <= d ，那么最大收益为 d - a。而 d - a = (d - c) + (c - b) + (b - a) ，因此当访问到一个 prices[i] 且 prices[i] - prices[i-1] > 0，那么就把 prices[i] - prices[i-1] 添加到收益中。
- 代码实现

```
public int maxProfit(int[] prices) {
  int profit = 0;
  for (int i = 1; i < prices.length; i++) {
      if (prices[i] > prices[i - 1]) {
          profit += (prices[i] - prices[i - 1]);
      }
  }
  return profit;
}
```

- 题目来源

四、字符串处理

4.1、最长回文子串（最长公共子串）

- 问题描述
  求一个字符串里的最长回文子串
- 解题思路
  - 根据回文串的定义，正着和反着读一样，那我们是不是把原来的字符串倒置了，然后找最长的公共子串就可以了。例如 S = "caba" ， S = "abac"，最长公共子串是 "aba"，所以原字符串的最长回文串就是 "aba"。
- 代码实现

```java
//暴力破解
public boolean isPalindromic(String s) {
          int len = s.length();
          for (int i = 0; i < len / 2; i++) {
                  if (s.charAt(i) != s.charAt(len - i - 1)) {
                          return false;
                  }
          }
          return true;
  }

public String longestPalindrome(String s) {
    String ans = "";
    int max = 0;
    int len = s.length();
    for (int i = 0; i < len; i++)
        for (int j = i + 1; j <= len; j++) {
            String test = s.substring(i, j);
            if (isPalindromic(test) && test.length() > max) {
                ans = s.substring(i, j);
                max = Math.max(max, ans.length());
            }
        }
    return ans;
}
```

```java
//扩展中心算法
//我们知道回文串一定是对称的，所以我们可以每次循环选择一个中心，进行左右扩展，判断左右字符是否相
//由于存在奇数的字符串和偶数的字符串，所以我们需要从一个字符开始扩展，
//或者从两个字符之间开始扩展，所以总共有 n+n-1 个中心。
public String longestPalindrome(String s) {
  if (s == null || s.length() < 1) return "";
  int start = 0, end = 0;
  for (int i = 0; i < s.length(); i++) {
      int len1 = expandAroundCenter(s, i, i);
      int len2 = expandAroundCenter(s, i, i + 1);
      int len = Math.max(len1, len2);
      if (len > end - start) {
          start = i - (len - 1) / 2;
          end = i + len / 2;
      }
  }
  return s.substring(start, end + 1);
}

private int expandAroundCenter(String s, int left, int right) {
    int L = left, R = right;
    while (L >= 0 && R < s.length() && s.charAt(L) == s.charAt(R)) {
        L--;
        R++;
    }
    return R - L - 1;
}
```

- 题目来源

4.2、最长公共子序列

- 问题描述
- 解题思路
- 代码实现

```java
public int longestCommonSubsequence(String text1, String text2) {
    int n1 = text1.length(), n2 = text2.length();
    int[][] dp = new int[n1 + 1][n2 + 1];
    for (int i = 1; i <= n1; i++) {
        for (int j = 1; j <= n2; j++) {
            if (text1.charAt(i - 1) == text2.charAt(j - 1)) {
                dp[i][j] = dp[i - 1][j - 1] + 1;
            } else {
                dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
            }
        }
    }
    return dp[n1][n2];
}
```

- 题目来源

4.3、0/1背包问题

- 问题描述
    - 有一个容量为 N 的背包，要用这个背包装下物品的价值最大，这些物品有两个属性：体积 w 和价值 v。
- 解题思路
    - 定义一个二维数组 dp 存储最大价值，其中 dp[i][j] 表示前 i 件物品体积不超过 j 的情况下能达到的最大价值。设第 i 件物品体积为 w，价值为 v，根据第 i 件物品是否添加到背包中，可以分两种情况讨论：
        - 第 i 件物品没添加到背包，总体积不超过 j 的前 i 件物品的最大价值就是总体积不超过 j 的前 i-1 件物品的最大价值，dp[i][j] = dp[i-1][j]。
        - 第 i 件物品添加到背包中，dp[i][j] = dp[i-1][j-w] + v。
    - 第 i 件物品可添加也可以不添加，取决于哪种情况下最大价值更大。因此，0-1 背包的状态转移方程为：dp[i][j]=max(dp[i-1][j], dp[i-1][j-w]+v)
- 代码实现

```java
// W 为背包总体积
// N 为物品数量
// weights 数组存储 N 个物品的重量
// values 数组存储 N 个物品的价值
public int knapsack(int W, int N, int[] weights, int[] values) {
    int[][] dp = new int[N + 1][W + 1];
    for (int i = 1; i <= N; i++) {
        int w = weights[i - 1], v = values[i - 1];
        for (int j = 1; j <= W; j++) {
            if (j >= w) {
                dp[i][j] = Math.max(dp[i - 1][j], dp[i - 1][j - w] + v);
            } else {
                dp[i][j] = dp[i - 1][j];
            }
        }
    }
    return dp[N][W];
}
```

- 题目来源

4.4、两数之和

- 问题描述
    - 给定一个整数数组 nums 和一个目标值 target，请你在该数组中找出和为目标值的那 两个 整数，并返回他们的数组下标
- 解题思路
    - 一遍哈希表
    - 在进行迭代并将元素插入到表中的同时，我们还会回过头来检查表中是否已经存在当前元素所对应的目标元素。如果它存在，那我们已经找到了对应解，并立即将其返回
- 代码实现

```
//O(n) O(n)
class Solution {
  public int[] twoSum(int[] nums, int target) {
      Map<Integer, Integer> map = new HashMap<>();
      for (int i = 0; i < nums.length; i++) {
          int complement = target - nums[i];
          if (map.containsKey(complement)) {
              return new int[] { map.get(complement), i };
          }
          map.put(nums[i], i);
      }
      throw new IllegalArgumentException("No two sum solution");
  }
}
```

- 题目来源

4.5、字符串反转

```
String str = "123";
StringBuffer sBuffer = new StringBuffer(str);
sBuffer.reverse();
```

五、other

5.1、XXXX

- 问题描述
- 解题思路
- 代码实现

```

```

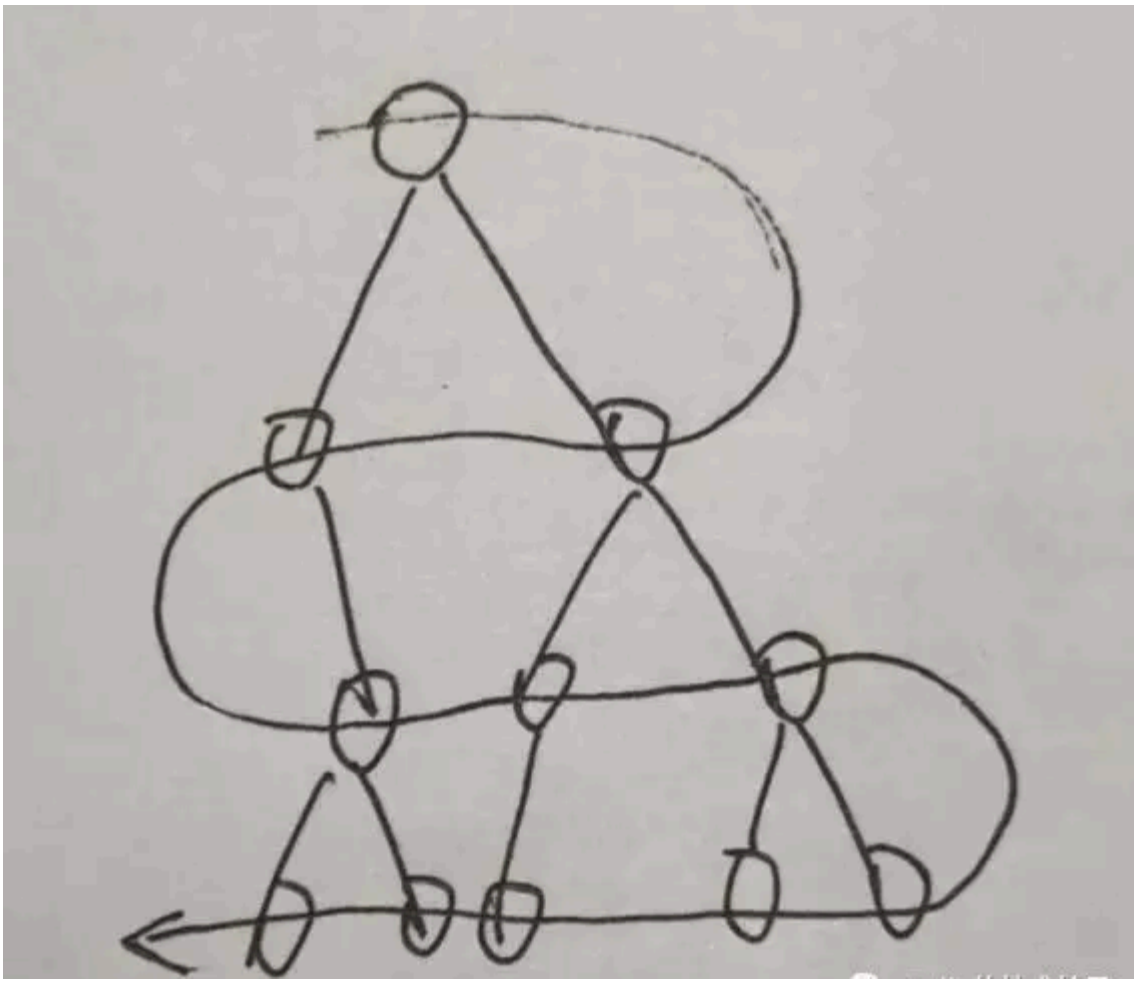
- 题目来源

5.2、LRU算法

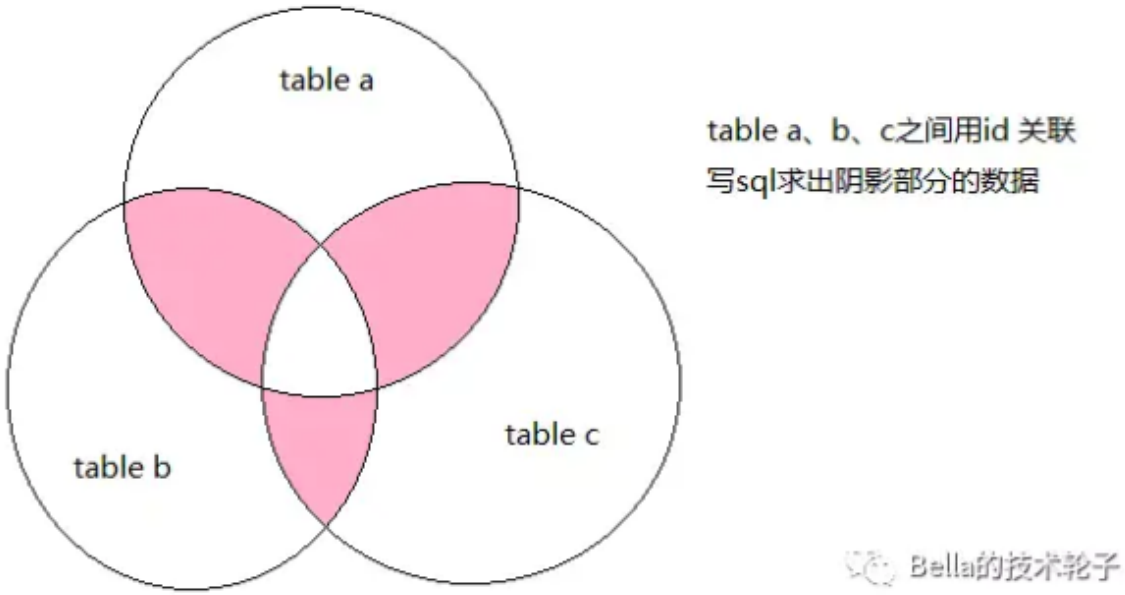
```
public static void main(String[] args) {
    LruCache lruCache = new LruCache(3);
    lruCache.put("1", "abc");
    lruCache.put("2", "def");
    lruCache.put("3", "ghi");
    lruCache.put("4", "kl;");
    System.out.println(lruCache.toString());
    lruCache.get("2");
    System.out.println(lruCache.toString());
}


static class LruCache<K, V> extends LinkedHashMap<K, V> {

    private int maxEntries = 10;

    @Override
    protected boolean removeEldestEntry(Entry<K, V> eldest) {
        return size() > this.maxEntries;
    }
    public LruCache(int maxEntries) {
        super(maxEntries, 0.75f, true);
        this.maxEntries = maxEntries;
    }
}
```

3.3、找出数组中不重复的元素

- [1,1,2,2,3,4,4,5,5,5] 找出不重复的元素（黄包车）
- 反转链表，要求时间复杂度O(N)，空间复杂度O(1) （火币）
- 非递归实现斐波那契数列 （爱奇艺）
- 这一周股市价格为[2,6,1,4,8]，求哪一天买入哪一天卖出，可获得最大收益，最大收益为多少 （爱奇艺）
- 按照箭头方向查找二叉树 （金山云）



- 表a b c之间用id关联，求阴影部分的数据 （金山云）【原创公众号：Bella的技术轮子】



- 一个整形无序数组，里面三个数只和等于一个目标值，求这三个数 （小米）
- 链表问题 （小米）

```
Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)
Output: 7 -> 0 -> 8
Explanation: 342 + 465 = 807.
```


- 扑克牌问题 （小米）
- 手写大顶堆 （linkedMe）
- 手写LRU 算法 （火币）
- 字符串相加 （滴滴）
  两个数字类型的字符串，直接转int或者double肯定都放不下，然后求这两个数的和，返回值还是字符串，15分钟时间，要求无bug
- 寻找目标值位置 （滴滴）
  有一个二维数组，数组横向有序，纵向有序，求目标值的位置，10分钟时间
- 求字符串“efabcbaefehiabcba”中最长的回文数，不去重（美团）
- 反转int类型的值x，不要借用String，只用int 即可。&& 针对该程序，写出其应有的测试用例 （美团）
- top K 问题（每日一淘）

参考文章

- 排序算法
- 还不会七大排序，是准备家里蹲吗！？