小土刀的面试刷题笔记 (index.html)

小土刀的面试刷题笔记

Amazon OA

OA1 - Debugging

一些可能的问题：

- while循环缺少i++造成死循环
- Print Pattern：for-loop里一共两句话但是没有用大括号，所以第二句没有被包含进去
- insert sort descending order: `<>` 反了
- selection sort: `arr[min]>arr[x]` 改成 `arr[y]`.
- reverse array: `arr[len-1]` 改成 `arr[len-i-1]`, 循环结束前去掉 `len+=1;`
- 循环里要加上 i++， 否则死循环
- 曼切斯特如果arr[i-1],arr相等为0，否则为1，要注意不但==要改成!=，ret[0]也要加一下，不然有一个case过不了。

解题技巧

因为地里面经分散,大家又说很简单,多办懒得附上题目,在这里提供几个思路给大家:

1. 排序类:这种题Compile & Run出来的结果,多半是Sort的顺序反了,稍微看一下找到关键的if statement把他反过来就成。我7题里遇到2题这种的。
2. TLE类:这种结果TLE的多半是while死循环了,有while的检查一下。
3. for loop类:有for loop的,检查一下大括号有没有加。
4. 其他类:其他特别的bug,多半地里找一下都有,记一下就成。

OA1 - Reasioning

找规律

- QPS : TSV -> IHK : (LKN) 都是+3
- 46 : 64 -> 82 : (100) （差为18）..也可能是28
- EAGLE : FZHKF -> THANKS : (UGBMLR) +1, -1 找规律的（奇数+1偶数-1）

- FASTER : HCUVGT -> SLOWER --> (UNQYGT) (+2)
- 985 : 874 -> 763 : (652) （每一位减一即可）
- 865 : 532 -> 976 : (643) （右边是左边每一位减三）
- ADBC : EHFG -> ILJK : (MPNO)（4个一组）
- JOHN : LSNV -> MARK : (OEXS)（+2 +4 +6 +8）
- COMPUTER : PMOCRETU -> TELEVISION : (VELETNOISI)（镜像倒过来）
- A17R : D12P -> G7N : ? （R=A+17）
- COMPUTER : GKQLYPIN -> SENATE : WARWXA（奇数+4偶数-4）
- KPQR : LRTV -> DGHY : (EIKC) (前后相减每一位的增加分别为1,2,3,4)
- ACFJ : CEHL -> PRUY : (RTWA) (前后相减每一位的增加分别为1,2,3,4)
- VAILANT : UBKJZOS -> TRANSCEND : SSCLRDGLC (奇数对：奇数-1，偶数+1，偶数对：奇数-2，偶数+2)
- 27 : 24 -> 64 : (60)（$24=3^{3-3}, 64=4^{3,4^{3-4=60)}}$
- MQD : KRK -> SWM : (NCF)（13, 17, 4；11, 18, 11; 交叉 -5(13-18) 6(17-11) -7(4-11)）
- AD5 : ED9 求和
- BGL : DIN : MRW : HLR（差5差6）
- PRS TVX FIK LME
- JLP LNT TVZ DFJ（感觉选1，因为不是4的倍数？）
- ABIJ DEHI MNQR STWX（ABIJ前后一对间距不同）
- ADP QTS HKR STE （选1？都是完全平方数？，或者QTS，位与的结果不是0？）
- RHCAI OEST HNDA ADEH（RHCAI？只有这个不是身体部位？）
- ADF MPR ILN EHJ（2？只有它不是以元音开头？）
- STV XYA KKT BDE（其他都是两偶一奇，只有KKT是两奇一偶）
- 956 794 884 678（678，前几组加起来和都是20）
- 1,4,16 17,20,24 8,11,18 19,20,5（感觉是最后一个，间距不是3的倍数）
- AE5 DF6 HN14 KP2（感觉选KP2，因为P！=2）
- HIK DGJ LPT SUW（1，因为不是等间距）
- LKJI XYWV WVUT KJIH（1？只有1以奇数开头？）
- 2,3,7,8,13,14,(20) （20？相差4,5,6？）
- 0 1 1 2 4 8 (16)（16前面的所有数加起来）
- 3,6,18,108,(1944) （18 * 6 = 108，所以应该是18*108）
- 1,1,4,2,13,3,40,4,(121)（1+3的1次方=4，4+3的2次方=13，13+3的3次方=40，40+3的4次方=121）
- 3 7 13 21 (31) （相差每次增长2）
- 5 11 19 29 (41)（相差每次增长2）
- 0 2 6 12 20 (30)（相差每次增长2）
- 5 9 16 29 (54) `5*2-1, 9*2-2, 16*2-3, 29*2-4`
- 4 12 6 18 12 36 30 (90)（奇数位乘以3就是偶数位）
- 1 5 (8) ($1 + 2^2 = 5$， $5 + 2^1 = 7$ ， $7 + 2^0 = 8$)
- D, H, L, (p) （P，等间距）

- 10 14 23 39 64 (100)（间距为完全平方数）
- 10 74 202 394 650（间距递增64）
- 2 8 5 6 8 (4) 11（应该是4吧，两两这么看的， 2、5、8、11， 加3得来的。 另外一组应该是减2，也就是8、6、4）
- 16 30 46 62 （13+3， 29+1， 43+3， 61+1 然后前面的都是质数，每个质数之间隔了三个质数，61之后第四个质数是79， 79+3=82，或者16加上后面的数为第二个的结果）
- 1:4:27:256:? 3125（$n^{n)}$
- 2, 5, 26，(677)（规律是当前数字是前一个数字平方加1）
- ASSERTIVENESS-> SENSSAEVISTRE : MULTINATIONAL -> ？（记录ASSERTIVENESS每个字母的位置，再记录下SENSSAEVISTRE每个字母的位置，找出mapping关系（比如A在ASSERTIVENESS中第一个位置，在SENSSAEVISTRE第六个位置，那么1->6）。最后记录MULTINATIONAL每个字母的顺序，按照之前找出的mapping对找出来（如M肯定会在所求字符串的第六个位置）。这种题字符串的长度和所包含的字母个数肯定是一样的。）重复字母的情况是有一定规律的，你看 ASSERTIVENESS-> SENSSAEVISTRE , 123456789,10,11,12,13 -> 12,11,10,3,2,1,9,8,7,13,6,5,4（用重复的字母把字符串隔开，就能看到排列顺序了）【2】ass 【5】ert 【3】ive 【1】nes 【4】s
- 原题字母，这里直接用数字表示：4，5，12，8，9 => 3，4，1，7，8问 13, 21, 13, 2, 1, 9 => ？（网上解法有：4, 5, 12, 8, 9 => (4 - 1), (5 - 1), (1 - 0)(2 - 1), (8 - 1), (9 - 1)，so13, 21, 13, 2, 1, 9 => 02, 10, 02, 1, 0,8 或者4 5 12 8 9 -> 3 4 1 7 8 （每位都-1）13 21 13 2 1 9 -> 02 10 02 1 26 8 就是2 10 2 1 26 8 （B J B A Z H）

应用题

- If northwest becomes east, northeast becomes south, and so on, what does southeast become? (west)
- Lily can't find her home, she is 25 yards southwest of her home, then she walked 20 yards toward north, where is her home from her now? (15 yards, east)
- 一个面朝北的朋友，先左走15m，然后一个about-turn走了30，这货在哪？（about turn应该指的是向后转）
- 小明往东南走4 miles，往西走8 miles， 再往西北走4 miles。现在小明离出发点是什么方位？（正西？平行四边形？）
- 小明面朝南，往左走20miles， 再往右走 10miles， 再往左走30miles。 现在小明离出发点是什么方位？(大致东南方向？)
- 南5西4南7东4北5，问方向、离原点距离
- 一个楼有3层，每个level 坐一些人，第二层能坐最多，一共坐66个人。给了两个条件求第二层坐了多少人。
    - 1，其中有一层坐了21 人。
    - 2，第二层比其中一层多座了2人
    - 我选的可以求出第二层多少人， 21，22， 23. 第二层23人。

- 推断一个人的年龄 （1）知道所有人的平均年龄 （2）所有人年龄都一样，问（1）和（2）怎么来推断这个人的年龄

印度公司问题

有个是问印度公司在radio上做广告，记忆没错的话，选B。大意是，radio覆盖面广，公司向推广自己的，应该去上面做广告。It has been proven by research that in India, a company which purchases saturation radioadvertising will get maximum brand recognition.

1. A high degree of brand recognition will help a company win a higher share of the market.
2. Radio has wide listenership and companies intending to increase their awareness,should advertise it.
3. For maximum brand recognition, a company need not spend on media channels other than radio publicizing.
4. Brand recognition in India is more heavily dependent on where the brand advertises than the quality of its offering.

原题说的saturation advertising是指同一个广告反复宣传，就好像脑白金的洗脑歌一样。。而研究表明通过这种广播，可以收获最大的品牌认同度。

1. 高品牌认同度使公司获得更高的市场份额（研究说的是宣传与认同度的关系，没有涉及市场份额）（品牌认知度和市场份额的关系未在题目条件中提及，无关联）
2. 广播有不错的听众基础，如果公司想提高他们的知名度认同度，应该考虑通过这种方式.
3. 为了达到最大的品牌宣传效果，公司不应该考虑广播之外的宣传方式 （广播频率max导致宣传效果max也只是广播频率的影响，并不代表其他宣传渠道如何）。（并未说明其他广告渠道对最大化品牌认知度的贡献，所以不该武断排除其他一切非收音机渠道的广告策略）
4. 在印度，品牌认同更看宣传的where，而不是质量 （同样原文没提到质量和效果之间的关系）（题目条件未提及广告质量，无关联）

环保公司问题

选择是否将候选公司放到一个环保list上，条件

1. hava ECC (一种认证)
2. 生成了至少三种solar 产品
3. none of their products are from synthetic
4. headquater in Texas
5. product 都由 A -certificate
6. donot have legal dispute or pending against them

如果不满足2,但是有一种产品正在试验中：推荐给COO

如果不满足5：推荐给Director of the company

录用PM

一个公司要招PM，合理的candidate需满足以下条件：

1. 本科是学CS的
2. 有MBA学位
3. 本科GPA 3.0+
4. 如果没有MBA学位，但是工作5年以上，需上报HR
5. 本科不是学CS，但是在CS相关工作3年以上，上报HR

那么，请问：闰土本科学热水锅炉维修的，GPA 4.0，没有念过MBA，在Google修了5年的锅炉，当了3年的程序员，则应该： D

A. 录用
B. 不录用
C. 条件不充分
D. 上报HR

另一个条件：

1. 候选人必须有硕士学位，且GPA为A
2. 必须有两年以上工作经验，
3. 若1不满足报告director

小明从事某工作三年，有CS和MBA，本科GPA为A-则：报告主管。

又一个条件条件是：

1. Master in commerce and at least B / have CPA
2. 年龄大于20 ，小于25
3. 流利的英语和西班牙语.
4. 愿意付125刀押金

5. 愿意承诺为公司工作5年

- 如果1不满足-> refer to M director
- 如果4不满足 -> refer to chair man .

快递收费

快递费要不要收的问题。条件是

1. 地区code 大于10一类，小于10 另一类
2. 商品价格超过500
3. 不是deal的时候买得

4. 之前没有bulk 超过5%的折扣. 1point3acres.com/bbs

5. 客户有优良购买记录3年

- 如果不满足2，那么要是他满足地区code小于10，收10刀，大于10，收8刀。
- 如果不满足3，那么region code小于10，收5刀，大于10，收12刀

来了一个老头，买了150刀的东西，不是deal的时候买的，也没有之前折扣。问他可不可以不付运费。
若不满足两条，则必须付全款。

选：附全款30刀

四人位置

There are four coordinators named Lily, Cathy,Mary and Nina. Each coordinator is at a different corner of the rectangle meeting hall. A coffee vending machine is situated at one of the corners and a restroom at another corner of the meeting hall. Lily and Cathy are at either sides of the white board, which is situated at the center of the side which is opposite to the side at whose corners the coffee vending machine and the restroom are located. Coordinator Mary is not at the corner where the restroom is located.
Which of the following cannot be true?

1. Lily is not on the side of the hall where the white board is placed
2. Nina is adjacent to the restroom at one corner
3. Cathy is at the corner, adjacent to the coffee vending machine.
4. Mary is adjacent to the coffee vending machine, at one corner of the hall
5. Lily is at the corner, adjacent to the coffee machine

选1？Lily和cathy推出3,5Mary这个推出2,4（注意23和45的区别）
重点是这句“on either sides of the white board”，俩人分别在whiteboard的一侧，而不是要在hall的white board side,我觉得可能性有很多(V是vending machine)：

```
_____________________________
| C                     N(R)|
| ||                        |
| ||(W)                     |
| ||                        |
| L_____________________ M(V)|
_____________________________
| N                     C(R)|
| |                         |
| |(W)                      |
| |                         |
| L_____________________ M(V)|
等情况
```

总之就是lc可以互换，其他两人不可。

只要M不在Vending machine那边，L和C分别在white board两侧应该就可以…

还有问题里这个选项：Mary is adjacent to the coffee vending machine, at one corner of the hall，Mary到底是在和vending machine相邻的corner还是就在vending machine呢？ 私以为是在vending machine的相邻的两个corner，但是这样看不出哪个选项有问题…

第一个选项1. Lily is not on the side of the hall where the white board is placed 是说L不在WHITE BOARD放置的那一侧，错。

第二个选项2. Nina is adjacent to the restroom at one corner
是说N与RESTROOM相邻，在某一CORNER，对。

第三个选项3. Cathy is at the corner, adjacent to the coffee vending machine
是说C在某一个CORNER，那个CORNER与COFFEE VENDING MACHINE 相邻，所以不是COFFEE VENDING MACHINE的那个CORNER，对。

第四个选项4. Mary is adjacent to the coffee vending machine, at one corner of the hall 是说M与COFFEE MACHINE 相邻，在某一个CORNER，对。. 1point3acres.com/bbs

第五个选项5. Lily is at the corner, adjacent to the coffee machine 是说L在某一个CORNER，那个CORNER与COFFEE MACHINE相邻，对.

八产品

题干是： 共有8个产品 mixer， iron, blender,water pump, geyser,juicer,heater,ginber. 4个人: Alan,Betty,Cathy,Diana 。每个人生产不同的两种产品。

Alan 和betty分别生产 mixer和iron，但是不知道具体谁生产，并且mixer不能和blender同一个人生产，cathy生产water pumper， Diana生产geyser

- 问题1： 如果mixer 和juicer同一个组生成，那么共有几种可能的排列组合
- 问题2： 如果生产alan生产mixer和heater，那么betty生产什么

A manufacture company has 8 products and 4 divisions. Four divisions are lead by Alan, Betty, Cathy, Diana. The 8 products are: mixer, iron, water pump, geyser, juicer, blender, grinder, and heater. Each division produces 2 products, no 2 divisions produces the same product. Diana’s division produced Geyser, Cathy’s division produces water pump. Mixer and iron areproduced by division lead by Alan and Betty respectively. The division that produces mixer doesn’t produce blender.

Four questions:
1. if the division that produces mixer doesn’t produce juicer, which of the following statement is true?
4. if the division that produces mixer also produces juicer,how many ways are there for product pairs? (3! = 6)

For factory problems, take care of the global assumptions andl ocal assumptions.

出差

说有M1,M2,M3,M4,M5和W1,W2,W3。出差必须派至少三男一女。M1和M3不能共存，M4和W2不能共存。

- 第一问问如果派了M2和M3和W2，还可以派谁
- 第二问问如果M1 M2去了，还可以派谁。
- 第三问如果去了四个男生，那么谁不能选。

圆桌问题

一圆桌坐八人ABCDEFGH. F在C右边两位,AE坐G两边, BH面对面:

- 问D对面是谁?G
- 以下哪两人坐对面?D&G
- 谁坐D旁边?C
- AB不坐隔壁, F对面坐A, 反时针方向可能的坐法?AHCDFBEG之类的

圆桌问题。 八角桌， B和H正对着，F在C的右边两个位置，A 和E在G的两侧，C朝北。
 6人团团坐问题，有六个人 GASMNR, 注意理解 G, A，S 两两不能对坐，所以总体来说分两种情况，GAS三人间隔而坐，或者GAS全都挨着坐。所有团团做的问题都围绕此基础展开。有一题说R 在A S 中间，问你R对面是谁。还有就是 G左右是A R,问A 对面是谁。

技巧总结

有关字母跟数字的:

1. 类比类:3组英文字母,给你推空着的那组应该是什么。 4选1,不难但是花时间,要注意时间不要用太多了。提前在纸上把26个字母跟对应的数字写上,可以节省很多时间。
2. 排异类:给你4组英文字母,叫你找出pattern不同的那组,基本就是同上。
3. 数列类:一串数列,叫你推最后一个。地里面经看过应该就知道大概要怎么去想了。
4. 转译类:xxxxx转成ooooo,问你aaaaa会转成什么这种,多半是奇偶位交错+x -x。

OA2 - Work Simulation

时间非常充足，可以慢慢做，就是问你要是你你会怎么选，你同意谁的观点和给一下几个做法打分的题

各个员工讨论case media network 服务器最近好多complaints,有德国的，有invalid recommendation的，给了个列表好多国家的服务器返回什么404/ german recommendation/ invalid recom/问是什么原因。还有俩个年轻老白讨论客人要强烈要求有硬皮书的推荐，但服务器里只有digital版本的，到底要不要加这个功能，感觉后面的视频是根据你的选择来的（有待考证）；里面有个会议室白人，亚裔，烙印在讨论服务器

最近好多complaints,然后我选则的要看Intenal test，结果后面会议结束烙印站起来义正言辞跟我说，我已经写了20年服务器了，不可能有错误的，而且我刚刚才调试过机器，绝对不可能是内部错误。呵呵，里面有个选项问，烙印 is not helpful...只能呵呵~~ 大部分跟地里说的一样，类似问卷调查，选deadline更重要和用户体验更重要。

第一个情境是给图书馆写图书推荐系统，第一问让两个人继续说，第二问选图书馆的服务器有没有开放关于实体书的api

后面有会议说系统出现bug，该做出什么反应，选看internal bug 记录。

最后是五个case看哪个可以通过，前人都提示过，注意user的构造函数没有给email赋值。

simulation就是看email，chat...大家记得每收到email就要看看，我当时碰到没有题的email直接跳过，后来做题的时候做了几道发现信息很少做不出来随便乱选了，翻了翻记录才发现有些信息都在那些没题的email里了。。看log得题就找相同错误的规律，我记得有道我选了地点都在德国，有个是因为username太长没存全， testcase就是地里说的那些email没有初始化，

找错题有5个unit test 有一个是user的payment method返回的是null，一个是user的构造函数不包含email，一个是setPrice()传进去的参数是double,但是return是int。coding： 1 reverse right half linkedlist example: 2->1->3->4->5->6->7->8 变成 2->1->3->4->8->7->6->5 ； 如果总是为奇数，中间的也要变 5->7->8->6->3->4->2 变成 5->7->8->2->4->3->6 很简单就不多说了

Work Simulation一开始两个码农撕逼，一个要用old API可以满足deadline，一个要独自开发new API可以满足requirements，这道题连续让你选三次，每次的视频都是根据你的选择不同而不同的。。楼主纠结很久后选择站在那个颜值更高的码农一边。。满足requirements。。那仨题其实是一个小测试：第一个选deadlline，因为这时没提出用户。后两个全用户优先。。。。。写这里给后人参考下。

其他不这么二选一的绝境，只要坚持deadline最好不要拖，自己辛苦一点无所谓，多咨询manager，找其他有经验的人合作啥的，随机应变吧。。

会有让你安排一个项目的计划，因为有很多不同的feature可以实现，但是要在8个月之内搞定，每个feature会有一个预计的占用时间和这个feature的重要程度。。只要坚持在占用时间一样的情况，多选牛逼的feature。。

Log里德语我选的proxy，invalid recommendation是因为username太长，database的那个field定义长度短了。。

ShoppingCartClass两道题三短一长选最长，之前这么选的拿到video了。。

5个Testcase选1， 3， 5过不了，2， 4能过。。

显示德语是因为proxy 推荐错误因为username 太长的被简化了。时间很充裕 完全不用着急

1. dealline与requirement。看着选吧。

2. log问题。找相同原因就行。我看的log是某个service出问题了，给了你一个report。第一问是为什么会出现德语，看report发现出现德语的共同点是locate都在德国，所以答案选的就是locate。第二问是为什么有的是invalid，看report发现共同点都是username都很长，因此选的username很长。

3. test case。关于shopping的代码。第一问是某个method为什么不行，答案选的performance issue。这个不太确定（其他几个选项更不合理）。第二问是how to improve shoppingcart class。我选的是add user.id to shoppingcart class. 第三问就是5个test case了。地里前辈说过很多了，应该是1，3，5跑不过。第一个是getdefualtpayment会返回null。第三个是user并没有初始化email，所以getemail会出错。第5个是 setprice的method 返回的是integer，而testcase set的是double 。

OA2 - Debugging

OOD的题，一共三道，第一个是问你其中有一个方法有什么问题，第二个是问有一个shopping cart class有什么问题，第三个是给了五个test case问能不能过，记得有一个是user类，构造函数有四个参数但是函数里只给其中三个赋值了email没有赋值，有一个test case大概是User user = new User(“abc”,“age”,“address”,“abc@gmail.com (mailto:abc@gmail.com)”) 问email最后等不等于abc@gmail.com (mailto:abc@gmail.com)我选的错。看log的也不难就是找相同出错原因问那几条的相同点就可以。

Coding

Right Rotation

```
public static int rightRotate(String word1, String word2) {
    if (word1 == null || word2 == null || word1.length() == 0 || word2.length() == 0 || wo
        return -1;
    }
    String str = word1 + word1;
    return str.indexOf(word2) != -1 ? 1 : -1;
}
```

Grey code

```
//term1和term2是题目给的两个BYTE
byte x = (byte)(term1 ^ term2);
int total = 0;
while(x != 0){
    x = (byte) (x & (x - 1));
    total++;
}
if(total == 1) return 1; else return 0;
```

去元音 remove vowel

```
StringBuffer sb = new StringBuffer();
String v = "aeiouAEIOU";
for(int i = 0; i < string.length(); i++){
    if(v.indexOf(string.charAt(i)) > -1) continue;
    sb.append(string.charAt(i));
}
return sb.toStirng();
```

检验括号

给你一个str,里面只有 '('和')',让你数valid pairs一共有多少,如果不是valid就返回-1. (判断是不是valid的parenthesis string，不是的话返回-1，是的话返回valid pair个数，即String.length() / 2)

```
import java.util.Stack;

public class isValid {
    public boolean isValidParentheses(String s) {
        if (s == null || s.length() == 0)   return true;
        Stack<Character> stack = new Stack<Character>();

        for (int i = 0; i < s.length(); i++) {
            if (stack.empty())  stack.push(s.charAt(i));
            else if (s.charAt(i) - stack.peek() == 1 || s.charAt(i) - stack.peek() == 2)
            else    stack.push(s.charAt(i));
        }

        return stack.empty();
    }
}
```

longest palindromic substring

```
public String longestPalindrome(String s) {
   char[] chars = s.toCharArray();
   int len = s.length();
   while (len >= 0) {
       for (int i = 0; i + len - 1 < chars.length; i++) {
           int left = i;
           int right = i + len - 1;
           boolean good = true;
           while (left < right) {
               if (chars[left] != chars[right]) {
                   good = false;
                   break;
               }
               left++;
               right--;
           }
           if (good)
               return s.substring(i, i + len);
       }
       --len;
   }
   return "";
}
```

另一个

```
public class Solution {
    public String longestPalindrome(String s) {
        String T = preProcess(s);
        int n = T.length();
        int[] p = new int[n];
        int center = 0, right = 0;
        for (int i = 1; i < n - 1; i++) {
            int j = 2 * center - i;  //j and i are symmetric around center
            p[i] = (right > i) ? Math.min(right - i, p[j]) : 0;

            // Expand palindrome centered at i
            while (T.charAt(i + 1 + p[i]) == T.charAt(i - 1 - p[i]))
                p[i]++;

            // If palindrome centered at i expand past right,
            // then adjust center based on expand palindrome
            if (i + p[i] > right) {
                center = i;
                right = i + p[i];
            }
        }

        //  Find the longest palindrome
        int maxLength = 0, centerIndex = 0;
        for (int i = 1; i < n - 1; i++) {
            if (p[i] > maxLength) {
                maxLength = p[i];
                centerIndex = i;
            }
        }

        centerIndex = (centerIndex - 1 - maxLength) / 2;
        return s.substring(centerIndex, centerIndex + maxLength);
    }

    // preProcess the original string s.
    // For example, s = "abcdefg", then the rvalue = "^#a#b#c#d#e#f#g#$"
    private String preProcess(String s) {
        if (s == null || s.length() == 0)  return "^$";
        StringBuilder rvalue = new StringBuilder("^");
        for (int i = 0; i < s.length(); i++)
            rvalue.append("#").append(s.substring(i, i+1));
        rvalue.append("#$");
        return rvalue.toString();
    }
}
```

Merge Two list

```
class ListNode {
    int val;
    ListNode next;
    ListNode(int x) { val = x; }
}

public class Solution {
    public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
        ListNode head = new ListNode(0);
        ListNode cur = head;
        while (l1 != null && l2 != null) {
            if (l1.val < l2.val) {
                cur.next = l1;
                l1 = l1.next;
            }
            else {
                cur.next = l2;
                l2 = l2.next;
            }
            cur = cur.next;
        }
        cur.next = (l1 != null) ? l1 : l2;
        return head.next;
    }
}
```

reverse second half of linked list

```
public static ListNode reverseSecondHalfList(ListNode head) {
    if (head == null || head.next == null)      return head;
    ListNode fast = head;
    ListNode slow = head;
    while (fast.next != null && fast.next.next != null) {
        fast = fast.next.next;
        slow = slow.next;
    }
    ListNode pre = slow.next;
    ListNode cur = pre.next;
    while (cur != null) {
        pre.next = cur.next;
        cur.next = slow.next;
        slow.next = cur;
        cur = pre.next;
    }
    return head;
}
```

Subtree

```
public class Subtree {
    public boolean isSubTree(TreeNode T1, TreeNode T2) {
        if (T2 == null) return true;
        if (T1 == null) return false;
        return (isSameTree(T1,T2) || isSubTree(T1.left, T2) || isSubTree(T1.right, T2));
    }
    public boolean isSameTree(TreeNode T1, TreeNode T2) {
        if (T1 == null && T2 == null)
            return true;
        if (T1 == null || T2 == null)
            return false;
        if (T1.val != T2.val)
            return false;
        return (isSameTree(T1.left, T2.left) && isSameTree(T1.right, T2.right));
    }
}
```

subtree里返回的是-1和1，而不是false和true，用迭代的同学特别注意！不能写if(isSameTree(root1, root2)||isSubTree(root1.left, roots)||isSubtree(root1.right, root2))了，因为三个function都返回int!!

Two Sum

```
import java.util.HashMap;
import java.util.Map;

public class Solution {
    public static int TwoSumCount(int[] nums, int target) {
        if (nums == null || nums.length < 2)
            return 0;
        Map<Integer, Integer> map = new HashMap<Integer, Integer>();
        int count = 0;
        for (int i = 0; i < nums.length; i++) {
            if (map.containsKey(target - nums[i]))
                count += map.get(target - nums[i]);
            if (!map.containsKey(nums[i]))
                map.put(nums[i], 1);
            else map.put(nums[i], map.get(nums[i]) + 1);
        }
        return count;
    }

    public static void main(String[] args) {
        int rvalue = TwoSumCount(new int[] {1, 1, 2, 3, 4}, 5);
        System.out.println(rvalue);
        return;
    }
}

// 另一个

public int[] twoSum(int[] nums, int target) {
    HashMap<Integer, Integer> hm = new HashMap<Integer, Integer>();
    for(int i = 0; i < nums.length; i++){
        int key = target - nums[i];
        if(hm.containsKey(key)){
            return new int[]{hm.get(key) + 1, i + 1};
        }
        hm.put(nums[i], i);
    }
    return new int[]{0 , 0};
}
```

Find K Nearest Point

```
import java.util.PriorityQueue;
import java.util.Comparator;

public class kNearestPoint {
    public Point[] Solution(Point[] array, Point origin, int k) {
        Point[] rvalue = new Point[k];
        int index = 0;
        PriorityQueue<Point> pq = new PriorityQueue<Point> (k, new Comparator<Point> () {
            @Override
            public int compare(Point a, Point b) {
                return (int) (getDistance(a, origin) - getDistance(b, origin));
            }
        });

        for (int i = 0; i < array.length; i++) {
            pq.offer(array[i]);
            if (pq.size() > k)
                pq.poll();
        }
        while (!pq.isEmpty())
            rvalue[index++] = pq.poll();
        return rvalue;
    }
    private double getDistance(Point a, Point b) {
        return Math.sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
    }
}
```

Overlap Rectangle

```
// Overlap Rectangle
// Rect 1: top-left(A, B), bottom-right(C, D)
// Rect 2: top-left(E, F), bottom-right(G, H)
public int computeArea(int A, int B, int C, int D, int E, int F, int G, int H) {
   int innerL = Math.max(A, E);
   int innerR = Math.max(innerL, Math.min(C, G));
   int innerT = Math.max(B, F);
   int innerB = Math.max(innerT, Math.min(D, H));
   return (C - A) * (B - D) - (innerR - innerL) * (innerT - innerB) + (G -E) * (F - H);
}
```

给两个长方形的topLeft和bottomRight坐标, 判断这两个长方形是否重叠

Rectangle Overlap。这题和leetcode 算相交面积的区别：它帮你定义好两个类，一个叫Point，一个叫Rectangle，Rectangle包括左上右下两个Point, Point包括x, y的值， 这种细节并不影响程序，总之一句判断直接通过了全部20多个case.

```
// Returns true if two rectangles (l1, r1) and (l2, r2) overlap
bool doOverlap(Point l1, Point r1, Point l2, Point r2)
{
    // If one rectangle is on left side of other
    if (l1.x > r2.x || l2.x > r1.x)
        return false;

    // If one rectangle is above other
    if (l1.y < r2.y || l2.y < r1.y)
        return false;

    return true;
}
```

window sum

```
public List<Integer> GetSum(List<Integer> A, int k) {
   ArrayList<Integer> result  = new ArrayList<>();
   if (A == null || A.size() == 0 || k <= 0) return result;
   int count = 0;
   for (int i = 0; i < A.size(); i++) {
       count++;
       if (count >= k) {
           int sum = 0;
           for (int j = i; j >= i - k + 1; j--) {
               sum += A.get(j);
           }
           result.add(sum);
       }
   }
   return result;
}
```

注意(arraylist == null || arraylist.size() == 0)要return一个已经初始化的arrayList而不是null，否则会有一个test case过不去

另一个

```
public class SumOfWindow {
    public int[] Solution(int[] array, int k) {
        if (array == null || array.length < k || k <= 0)    return null;
        int[] rvalue = new int[array.length - k + 1];
        for (int i = 0; i < k; i++)
            rvalue[0] += array[i];
        for (int i = 1; i < rvalue.length; i++) {
            rvalue[i] = rvalue[i-1] - array[i-1] + array[i+k-1];
        }
        return rvalue;
    }
}
```

GCD Greatest Common Divisor

就是给一个数组找这些数的最大公约数

```
public class GCD {
    public int Solution(int[] array) {
        if (array == null || array.length == 1) return 0;
        int gcd = array[0];
        for (int i = 1; i < array.length; i++) {
            gcd = gcd(gcd, array[i]);
        }
        return gcd;
    }
    private int gcd(int num1, int num2) {
        if (num1 == 0 || num2 == 0) return 0;
        while (num1 != 0 && num2 != 0) {
            if (num2 > num1) {
                num1 ^= num2;
                num2 ^= num1;
                num1 ^= num2;
            }
            int temp = num1 % num2;
            num1 = num2;
            num2 = temp;
        }
        return num1 + num2;
    }
}
```

LRU Cache count miss

实现 LRU Cache再判断啥时候miss就好了，返回miss数。建议看看用LinkedHashMap实现lru cache, 代码很简洁。

```
import java.util.LinkedList;
import java.util.List;

public class CacheMiss {
    public int Solution(int[] array, int size) {
        if (array == null)  return 0;
        List<Integer> cache = new LinkedList<Integer>();
        int count = 0;
        for (int i = 0; i < array.length; i++) {
            if (cache.contains(array[i])) {
                cache.remove(new Integer(array[i]));
            }
            else {
                count++;
                if (size == cache.size())
                    cache.remove(0);
            }
            cache.add(array[i]);
        }
        return count;
    }
}
```

Round Robin

一个处理器要处理一堆request，一次只能处理一条，每次执行一个任务最多执行时间q，接着执行等待着的下一个任务。若前一个任务没执行完则放到队尾，等待下一次执行

假设只要有任务开始以后cpu是不会空闲的，也就是说cpu开始后如果空闲了就说明没有任务了，另外Robin Round最后返回值是float

```
import java.util.LinkedList;
import java.util.Queue;

class process {
    int arrTime;
    int exeTime;
    process(int arr, int exe) {
        arrTime = arr;
        exeTime = exe;
    }
}

public class RoundRobinScheduling {
    public float Solution(int[] Atime, int[] Etime, int q) {
        if (Atime == null || Etime == null || Atime.length != Etime.length)
            return 0;
        int length = Atime.length;
        Queue<process> queue = new LinkedList<process>();
        int curTime = 0, waitTime = 0;
        int index = 0;
        while (!queue.isEmpty() || index < length) {
            if (!queue.isEmpty()) {
                process cur = queue.poll();
                waitTime += curTime - cur.arrTime;
                curTime += Math.min(cur.exeTime, q);
                for (; index < length && Atime[index] <= curTime; index++)
                    queue.offer(new process(Atime[index], Etime[index]));
                if (cur.exeTime > q)
                    queue.offer(new process(curTime, cur.exeTime - q));
            }
            else {
                queue.offer(new process(Atime[index], Etime[index]));
                curTime = Atime[index++];
            }
        }
        return (float) waitTime / length;
    }
}
```

Rotate Matrix

把一个m*n的矩阵旋转90度，给一个flag规定是向左转还是向右转

```
public class Rotate {
    public int[][] Solution(int[][] matrix, int flag) {
        if (matrix == null || matrix.length == 0 || matrix[0].length == 0)  return matrix;
        //int m = matrix.length, n = matrix[0].length;
        int[][] rvalue;
        rvalue = transpose(matrix);
        flip(rvalue, flag);
        return rvalue;
    }

    private int[][] transpose(int[][] matrix) {
        int m = matrix.length, n = matrix[0].length;
        int[][] rvalue = new int[n][m];
        for (int i = 0; i < n; i++)
            for (int j = 0; j < m; j++)
                rvalue[i][j] = matrix[j][i];
        return rvalue;
    }

    private void flip(int[][] matrix, int flag) {
        int m = matrix.length, n = matrix[0].length;
        if (flag == 1) {    //clock wise
            for (int i = 0; i < m; i++)
                for (int j = 0; j < n / 2; j++) {
                    matrix[i][j] ^= matrix[i][n-j-1];
                    matrix[i][n-j-1] ^= matrix[i][j];
                    matrix[i][j] ^= matrix[i][n-j-1];
                }
        }
        else {
            for (int i = 0; i < m / 2; i++)
                for (int j = 0; j < n; j++) {
                    matrix[i][j] ^= matrix[m-i-1][j];
                    matrix[m-i-1][j] ^= matrix[i][j];
                    matrix[i][j] ^= matrix[m-i-1][j];
                }
        }
    }
}
```

rotate matrix好像有变化，flag的值和左旋右旋的对应倒过来了，反正我是从面经里看到的代码，写完后发现要把左旋右选的选择换一下就pass了

insert into cycle linked list

插入一个新的节点到一个sorted cycle linkedlist，返回新的节点。给的list节点不一定是最小节点

所以先要找到最小的点

```
public class LinkedListInsert {
    public ListNode Solution(ListNode head, int val) {
        if (head == null) {
            ListNode rvalue = new ListNode(val);
            rvalue.next = rvalue;
            return rvalue;
        }

        ListNode cur = head;

        do {
            if (val <= cur.next.val && val >= cur.val)  break;
            if (cur.val > cur.next.val && (val < cur.next.val || val > cur.val))    break;
            cur = cur.next;
        } while (cur != head);

        ListNode newNode = new ListNode(val);
        newNode.next = cur.next;
        cur.next = newNode;
        return newNode;
    }
}
```

Loop in linked list

检查是否有环，以及环的起点在哪里

```
class ListNode {
    int val;
    ListNode next;
    ListNode(int x) {
        val = x;
        next = null;
    }
}

public class LinkedListLoop {
    public boolean hasCycle(ListNode head) {
        if (head == null)   return false;
        ListNode slow = head;
        ListNode fast = head;
        while (fast.next != null && fast.next.next != null) {
            slow = slow.next;
            fast = fast.next.next;
            if (slow == fast)   return true;
        }
        return false;
    }

    public ListNode detectCycle(ListNode head) {
        if (head == null || head.next == null)
            return null;
        ListNode slow = head;
        ListNode fast = head;
        ListNode entry = head;

        while (fast.next != null && fast.next.next != null) {
            slow = slow.next;
            fast = fast.next.next;
            if (slow == fast) {
                while (entry != slow) {
                    entry = entry.next;
                    slow = slow.next;
                }
                return entry;
            }
        }

        return null;
    }
}
```

Shorted job first

一个处理器要处理一堆request，一次只能处理一条，如果它有几个积压着的requests，它会先执行持续时间短的那个；对于持续时间相等的requests，先执行最早到达处理器的request。问平均每个request要等多久才能被处理。input：requestTimes[]，每个request到达处理器的时间; durations[] 每个request要处理的

持续时间。 两个数组是一一对应的，并已按requestTimes[] 从小到大排序过

public double CalWaitingTime(int requestTimes[], int[] durations[]){}

用priorityqueue做，地里有一个两层循环的答案，没仔细看，做完round robin以后发现思路很相似。注意用priorityqueue写comparator的时候，要先判断两者的execute time,如果execute time相同，则返回arrival time之差，即先按执行时间排序，若执行时间相同则按到达的时间排。

```
import java.util.Comparator;
import java.util.PriorityQueue;

public class ShortestJobFirst {
    public float Solution(int[] req, int[] dur) {
        if (req == null || dur == null || req.length != dur.length)
            return 0;
        int index = 0, length = req.length;
        int waitTime = 0, curTime = 0;
        PriorityQueue<process> pq = new PriorityQueue<process>(new Comparator<process>() {
            @Override
            public int compare(process p1, process p2) {
                if (p1.exeTime == p2.exeTime)
                    return p1.arrTime - p2.arrTime;
                return p1.exeTime - p2.exeTime;
            }
        });
        while (!pq.isEmpty() || index < length) {
            if (!pq.isEmpty()) {
                process cur = pq.poll();
                waitTime += curTime - cur.arrTime;
                curTime += cur.exeTime;
                while (index < length && curTime >= req[index])
                    pq.offer(new process(req[index], dur[index++]));
            }
            else {
                pq.offer(new process(req[index], dur[index]));
                curTime = req[index++];
            }
        }
        return (float) waitTime / length;
    }
}
```

Binary search tree minimum sum from root to leaf

跟BST没啥关系，不要看到BST就以为是最左边的路径之和（左边路径可以很长，右边路径可以很短），用递归做很简单

```
public class PathSum {
    public int Solution(TreeNode root) {
        if (root == null)   return 0;
        if (root.left != null && root.right == null)
            return Solution(root.left) + root.val;
        if (root.left == null && root.right != null)
            return Solution(root.right) + root.val;
        return Math.min(Solution(root.left), Solution(root.right)) + root.val;
    }
}
```

Day change(cell growth)

int[] dayChange(int[] cells, int days), cells 数组，有8个元素，每天的变化情况是 当前位置 cell = (cell[i - 1] == cell[i + 1]) ? 0 : 1, 左右相等，当前位置为0， 不等为1， 默认 cells[0]左边 和 cells[cells.length - 1]右边的位置元素都是0， 求days天后的变化.

```
import java.util.Arrays;

public class DaysChange {
    public int[] Solution(int[] days, int n) {
        if (days == null || n <= 0) return days;
        int length = days.length;
        int[] rvalue = new int[length + 2];
        rvalue[0] = rvalue[length+1] = 0;
        int pre = rvalue[0];
        for (int i = 1; i <= length; i++)
            rvalue[i] = days[i-1];
        for (int i = 0; i < n; i++) {
            for (int j = 1; j <= length; j++) {
                int temp = rvalue[j];
                rvalue[j] = pre ^ rvalue[j+1];
                pre = temp;
            }
        }
        return Arrays.copyOfRange(rvalue, 1, length+1);
    }
}
```

Maze

给个array,其中只有一格是9，其他格子是0或1，0表示此路不通，1表示可以走，判断从（0,0) 点开始上下左右移动能否找到这个是9的格子

```
import java.util.LinkedList;
import java.util.Queue;

public class Maze {
    private final static int[] dx = {-1, 0, 0, 1};
    private final static int[] dy = {0, 1, -1, 0};
    public int Solution(int[][] matrix) {
        if (matrix == null || matrix.length == 0 || matrix[0].length == 0)
            return 0;
        if (matrix[0][0] == 9)
            return 1;
        int m = matrix.length, n = matrix[0].length;
        Queue<int[]> queue = new LinkedList<int[]>();
        queue.offer(new int[]{0, 0});
        matrix[0][0] = 1;
        while (!queue.isEmpty()) {
            int[] cur = queue.poll();
            for (int i = 0; i < 4; i++) {
                int[] next = {cur[0] + dx[i], cur[1] + dy[i]};
                if (next[0] >= 0 && next[0] < m && next[1] >= 0 && next[1] < n) {
                    if (matrix[next[0]][next[1]] == 9)
                            return 1;
                    else if (matrix[next[0]][next[1]] == 0) {
                        queue.offer(next);
                        matrix[next[0]][next[1]] = 1;
                    }
                }
            }
        }
        return 0;
    }
}
```

Maze : 这题折腾了我好久啊！之前没时间把所有题都做一遍（地里的面经已经几乎涵盖了题库了），然后就没做这题，觉得不会遇到……

这题呢，用故事性强一点的说法就是，一只可怜的饥饿的小老鼠在一个N乘以N的迷宫里面（其实就一个二维数组……），它从（0，0）点开始出发，要寻找奶酪来吃（似乎是它闻到有奶酪的味了？）。

然后呢，这个二维数组表示的迷宫里面，1是路，0是墙（如果没记错的话，大家还是认真看看这个），值为9的地方是终点！

题目给出的就是这个二维数组，问你小老鼠最后能不能吃到奶酪（到达9）呢？能的话就返回true，不(e)能(si)的话就返回false。

我留意到了之前做过这个题的楼主提醒，老鼠在（0，0）开始，然后要测（0，0）就是终点（==9）的情况， 多么坑的一个corner case。

然后我就很欢(ku)乐(bi)地开始写了。但是！楼主这题花了30多分钟才写完，原因如下：

倒霉1： 楼主之前各种训练shortest time job first, round robin， 然后各种需要递归迭代的题通通很弱……于是这题就花了很长的时间。

倒霉2： 楼主的网络连接不给力，10分钟之内断了3次，重复登录3次，幸运的是都登录回去了。每次登出都得等2分钟来登回去。（现在想想我应该利用这个登出的时间把题妥妥的做完在登入啊！！）

倒霉3： 楼主没想到起点为0（墙）的情况是直接返回不能(false)的！那时候我已经被网络不佳给搞得特别慌了，就剩12分钟了，心里想着这回惨了，简单题都过不了。然后楼主一个不小心点到“test case”那个tab。（注意不是你的test case运行的情况，而是OA给出来的两个示例test case） 然后那里就有一个起点为墙返回为false…… 然后楼主立即马上迅速地加上了这条才过了的。

Minimum Path Sum

```
public class PathSum {
    public int Solution(TreeNode root) {
        if (root == null)   return 0;
        if (root.left != null && root.right == null)
            return Solution(root.left) + root.val;
        if (root.left == null && root.right != null)
            return Solution(root.right) + root.val;
        return Math.min(Solution(root.left), Solution(root.right)) + root.val;
    }
}
```

Window Minimum。具体题目如下：

给了一个ArrayList：4, 2, 12, 11, -5，窗口size为2，返回的ArrayList为：2, 2, 11, -5。这里窗口size是一个参数。

```
public class Solution {
    public int[] maxSlidingWindow(int[] nums, int k) {

        if (nums == null || k <= 0) {
            return new int[0];
        }
        int n = nums.length;
        int[] r = new int[n-k+1];
        int ri = 0;
        // store index
        Deque<Integer> q = new ArrayDeque<>();
        for (int i = 0; i < nums.length; i++) {
            // remove numbers out of range k
            while (!q.isEmpty() && q.peek() < i - k + 1) {
                q.poll();
            }
            // remove larger numbers in k range as they are useless
            while (!q.isEmpty() && nums[q.peekLast()] > nums[i]) {
                q.pollLast();
            }
            // q contains index... r contains content
            q.offer(i);
            if (i >= k - 1) {
                r[ri++] = nums[q.peek()];
            }
        }
        return r;

    }
}
```

Four Integer

Given four integers, make F(S) = abs(S[0]-S[1])+abs(S[1]-S[2])+abs(S[2]-S[3]) to be largest

```
import java.util.Arrays;

public class FourInteger {
    public int[] Solution(int A, int B, int C, int D) {
        int[] rvalue = new int[4];
        rvalue[0] = A;
        rvalue[1] = B;
        rvalue[2] = C;
        rvalue[3] = D;
        Arrays.sort(rvalue);
        swap(rvalue, 0, 1);
        swap(rvalue, 2, 3);
        swap(rvalue, 0, 3);
        return rvalue;
    }
    private void swap(int[] array, int i, int j) {
        array[i] ^= array[j];
        array[j] ^= array[i];
        array[i] ^= array[j];
    }
}
```

Arithmetic sequence

Given an array, return the number of possible arithmetic sequence

```
public static int getLAS(int[] A){
    // Time: O(n)
    // Space: O(1)
    if (A.length < 3) return 0;

    int res = 0;
    int diff = Integer.MIN_VALUE;
    int count = 0;
    int start = 0;
    for (int i = 1; i < A.length; i++){
        int currDiff = A[i] - A[i - 1];
        if (diff == curDiff){
            count += i - start - 1 > 0 ? i - start - 1 : 0;
        } else {
            start = i - 1;
            diff = currDiff;
            res += count;
            count = 0;
        }
    }
    res += count;
    return res;
}
```

另一个做法

```
public class ArithmeticSlice {
    public int Solution(int[] array) {
        if (array == null || array.length < 3)  return 0;
        int rvalue = 0, gap = array[1] - array[0], length = 2;
        for (int i = 1; i < array.length - 1; i++) {
            if (array[i+1] - array[i] == gap)   length++;
            else {
                gap = array[i+1] - array[i];
                if (length >= 3)
                    rvalue += (length - 1) * (length - 2) / 2;
                if (rvalue > 1000000000)    return -1;
                length = 2;
            }
        }
        if (length >= 3)
            rvalue += (length - 1) * (length - 2) / 2;
        return rvalue > 1000000000 ? -1 : rvalue;
    }
}
```

Tree Amplitude

Given a tree of N nodes, return the amplitude of the tree

就是从 root 到 leaf max - min 的差

```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode (int x) {
        val = x;
        left = null;
        right = null;
    }
}

public class TreeAmplitude {
    public int Solution(TreeNode root) {
        if (root == null)   return 0;
        return helper(root, root.val, root.val);
    }

    private int helper(TreeNode root, int min, int max) {
        if (root == null)   return max - min;

        if (root.val < min) min = root.val;
        if (root.val > max) max = root.val;

        return Math.max(helper(root.left, min, max), helper(root.right, min, max));
    }
}
```

Maximum Minimum Path

给一个int[][]的matirx，对于一条从左上到右下的path pi，pi中的最小值是mi，求所有mi中的最大值。只能往下或右.
比如：

```
[8, 4, 7]
[6, 5, 9]
有3条path:
8-4-7-9 min: 4
8-4-5-9 min: 4
8-6-5-9 min: 5
return: 5
```

解答

dp 方法

```
int helper(int[][] matrix){
    int[] result = new int[matrix[0].length];
    result[0] = matrix[0][0];
    for(int i=1; i<matrix[0].length; i++){
        result[i] = Math.min(result[i-1], matrix[0][i]);
    }
    for(int i=1; i<matrix.length; i++){
        result[0] = Math.min(matrix[i][0], result[0]);
        for(int j=1; j<matrix[0].length; j++){
            result[j] = (result[j]<matrix[i][j] && result[j-1]<matrix[i][j])?Math.max(resu
            }
        }
        return result[result.length-1];
}
```

dfs

```
 public class MaximumMinimumPath {
    private int min, max, row, col;
    public int maxMinPath(int[][] matrix) {
        row = matrix.length;
        col = matrix[0].length;
        min = Integer.MAX_VALUE;
        max = Integer.MIN_VALUE;
        dfsHelper(matrix, min, 0, 0);
     return max;
    }

    public void dfsHelper(int[][] matrix, int min, int i, int j ){
        if (i >= row || j >= col) return;
        if (i == row - 1 && j == col - 1) {
            min = Math.min(min, matrix[i][j]);
            max = Math.max(max, min);
            return;
        }
        min = Math.min(min, matrix[i][j]);
        dfsHelper(matrix, min, i, j + 1);
        dfsHelper(matrix, min, i + 1, j);
    }
}
```

Window Minimum

给了一个ArrayList：4, 2, 12, 11, -5，窗口size为2，返回的ArrayList为：2, 2, 11, -5。这里窗口size是一个参数。

```
import java.util.ArrayDeque;
import java.util.Deque;

public class Main {

    public static void main(String[] args) {
        int[] arr = new int[]{4, 2, 12, 11, -5};
        int[] result = minWindow(arr, 2);
        for (int i = 0; i < result.length; i++){
            System.out.println(result[i]);
        }
    }

    public static int[] minWindow(int[] nums, int k){
        if (nums == null || k <= 0)
            return new int[0];

        int n = nums.length;
        int[] r = new int[n-k+1];
        int ri = 0;

        // store index
        Deque<Integer> q = new ArrayDeque<>();
        for (int i = 0; i < n; i++){
            // remove number out of range k
            while (!q.isEmpty() && q.peek() < i - k + 1){
                q.poll();
            }

            // remove larger numbers in k range as they are useless
            while(!q.isEmpty() && nums[q.peekLast()] > nums[i]){
                q.pollLast();
            }

            // q contains index... r contains content
            q.offer(i);
            if (i >= k - 1){
                r[ri++] = nums[q.peek()];
            }
        }
        return r;
    }
}
```

实战唯一不同在于给的是arraylist，没错，你需要可耻的这么声明：ArrayList result=new ArrayList();

另外，1. 若是用Java，用到queue, list啥的记得前面手动import java.util.* 2.所有函数都是static的，所以自己写其他helper函数的时候记得加上static

Search in 2D matrix

```
class Point {
    int x;
    int y;
}

Point isInMatrix(int[][] matrix, int target){
    int row = matrix.length;
    int column = matrix[0].length;
    int r = 0;
    int c = column - 1;
    while (r < row && c >= 0){
        if (matrix[r][c] == target){
            return new Point(r,c);
        }
        if (matrix[r][c] > target){
            c--;
        } else {
            r++;
        }
    }
    return new Point(-1,-1);
}
```

Close Two Sum

findOptimalWeights,但大致是這樣:

/* 一個已經預設好的class */

```
class Container {
    public double first;
    public double second;
}
```

現在給某個容量(double capacity), 還有一個array(double[] weights), 和int numOfContainers

主要是要在array中選出兩個weights總和小於等於capacity但最接近capacity 然後指定到一個Container object並且return

first和second的順序並不影響，numOfContainer在java裡好像也是沒有用的,因為double[]本身就自帶length資訊

public Container findOptimalWeights(double capacity, double[] weights, int numOfContainers)

最後用了最簡單的方法兩個 for loop找總和最接近capacity的pair
總共8個test cases直接就過了

```
public static void findOptimalWeights(double capacity, double[] weights, int numOfContair
       double min = 0.0;
       int minPos = 0;
       int maxPos = weights.length - 1;
       int i =0 , j =weights.length-1;

       Arrays.sort(weights);

       while(i < j){
           double sum = weights[i] + weights[j];

           if(sum > min && sum <= capacity){
               min = sum;
               minPos = i;
               maxPos = j;
           }

           if(sum > capacity){
               j--;
           }else {
               i++;
           }
       }

       System.out.println("The two numbers for which sum is closest to target are "
               + weights[minPos] + " and " + weights[maxPos]);
   }
```

« Amazon (14520850399925.html)

Python » (14520848328638.html)

TOP