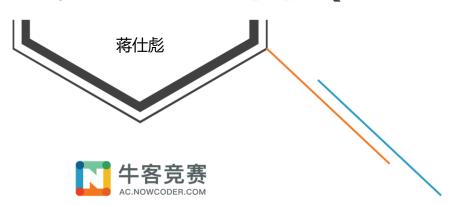
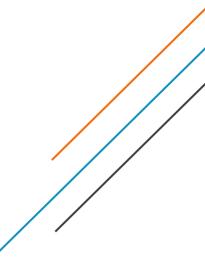


# 2020牛客暑期多校训练营 (第四场)



# Results

出题人以为的顺序	出题人以为的难度	实际难度顺序	实际过题人数
Basic GCD Problem	1	Finding the Order	1214/2786
Finding the Order	1	Basic GCD Problem	1147/5552
Harder GCD Problem	1.5	Harder GCD Problem	530/2654
Dividing Strings	2	Count New String	63/443
Investigating Legions	3	Investigating Legions	45/606
Count New String	3	Dividing Strings	20/334
Ancient Distance	4	Ancient Distance	12/182
Eliminate++	4.5	Eliminate++	9/264
Jumping on the Graph	4.5	Jumping on the Graph	1/12
Geometry Challenge	5	Geometry Challenge	0/3







# **Finding the Order**

```
while(t--){
    cin>>ac>>ad>>bc>>bd;
    if(ac>ad){
         if(bc>ac&&bc>bd)
             cout<<"AB//CD"<<endl;</pre>
         else
             cout<<"AB//DC"<<endl;
    else if(ac==ad){
         if(bc>bd)
             cout<<"AB//CD"<<endl;</pre>
         else
             cout<<"AB//DC"<<endl;</pre>
    else{
         if(bd>ad&&bd>bc)
             cout<<"AB//DC"<<endl;</pre>
         else
             cout<<"AB//CD"<<endl;</pre>
```

```
一个代码量中等的做法
```

```
scanf("%d %d %d %d", &a, &b, &c, &d);
                                                     else if (a >= b && c <= d && a <= c && b <= d)
   printf("AB//DC\n");
                                                        printf("AB//DC\n");
else if (a > b && c > d && a < c && b < d)
                                                     else if (a >= b && c <= d && a >= c && b >= d)
   printf("AB//CD\n");
                                                        printf("AB//DC\n");
else if (a < b && c > d && a > c && b > d)
                                                         printf("AB//DC\n");
    printf("AB//CD\n");
                                                     else if (a <= b && b >= c && b >= d)
else if (a > b && c < d && a > c && b > d)
                                                        printf("AB//CD\n");
   printf("AB//DC\n");
else if (a <= b && c >= d && a <= c && b <= d)
                                                        printf("AB//DC\n");
    printf("AB//CD\n");
                                                     else if (c \le d \&\& c \ge a \&\& c \ge b)
else if (a \le b \& c \ge d \& a \ge c \& b \ge d)
                                                        printf("AB//CD\n");
   printf("AB//CD\n");
                                                     else if (c <= d && d >= a && d >= b)
else if (a <= b && c >= d && a <= c && b >= d)
                                                         printf("AB//DC\n");
   printf("AB//CD\n");
```





# **Finding the Order**

#### ・大致题意

- 有两条平行的直线 AB 和 CD。
- 给出 AC, AD, BC, BD 四个距离,问是 AB//CD 还是 AB//DC。

### ・一个比较简单的做法

- 找到这四个距离的最大值。
- 如果最大值来自 AD 或 BC,则 AB//CD,否则 AB//DC。





## **Basic Gcd Problem**

### ・大致题意

$$egin{aligned} f_c(x) &= \max_{i=1\ldots x-1} c \cdot f_c(\gcd(i,x)) & x > 1 \ f_c(x) &= 1 & x = 1 \end{aligned}$$

•  $N <= 10^6$ 

### ・做法

- 观察公式, f<sub>c</sub>(x) 其实是 c 的若干次方, 且指数要尽量大。
- 最好的情况下,每次只消掉一个质因子。
- 所以  $f_c(x) = c^{x \text{的质因子个数}}$



### **Harder Gcd Problem**

#### ・大致题意

- 把 1~N 的数选尽量多的组,使得每组 gcd 大于 1。
- 输出任意一种方案。

#### ・做法

- p\*2>n的 p必然不能匹配,将它们除去。
- 倒序枚举所有质因子 p, 考虑所有是 p 的倍数、且未被匹配的数,任意将它们进行匹配。
- 如果个数是奇数就留下 p\*2。
- 最后把剩下的偶数都随意匹配一下。



# **Count New String**

### ・大致题意

- 定义字符串函数  $f(S,x,y)(1 \le x \le y \le n)$  , 返回一个长度为y-x+1的字符串,第i位是  $\max_{i=x...x+k-1} S_i$
- 求集合 A 的大小
- N <=100000
- · 字符集大小<=10



# **Count New String**

#### · 核心点 1

• 这题等价于 f(S,i,n) 这 n 个串的不同子串的个数

#### ·核心点 2

- 假设当前字符的位置是 i, 最近的大于等于它的字符的位置是 j, 那么新增的代价是 j-i。
- f(S,i,n) 翻转后构成的字典树的大小不超过 10N

### ・所以我们要考虑这个字典树本质不同的子串

- 最暴力的方法:在构成的字典树上建广义后缀自动机即可,
   设 k 为字符集大小,复杂度O(Nk²)
- 也可以用一些哈希或序列自动机的方法求一求。





#### ・大致题意

- 有N个点(30-300)和M(1~N/30)个团,每个点仅属于 一个团(等概率在0~M-1选择一个整数作为它的团)。
- 有一个常数 S (20~100) 。按以下的方式生成 01 矩阵 a: 如果 i 和 j 属于同一个团, a[i][j]=1, 否则a[i][j]=0。同时该值有 1/S 的概率被翻转。
- 给出 N 和 S 和 a (M 不给出),要还原每个点属于的团。
- 多种可能的情况?
- 后验概率最大!



- ・该问题仅供娱乐>\_<
- ・乱搞的大致想法
  - 从一个点出发先把所有 1 的的连通块构出来。如果其中混入了错误点,那么他和其他点之间的 1 肯定偏少。
  - 难点在于 m 没有给出,所以偏少的这个阈值需要进行构造 一下。这个可以在本地自行生成数据测试。
  - 还可以用团与团之间 0 比较多的特性去纠正一些例子。





### ・标算的做法

- 标算采用了一个迭代的形式化解法。
- 我们的期望:对于图中的一个三元环 (i, j, k),我们总是希望他们之间的边要不全相等要不是 (0, 0, 1),却不能是 (1, 1, 0)。因为后者代表了不合法情况。
- 我们试图用迭代的方法去纠正少量的错误,使得图中不存在 (1, 1, 0) 的 Case。
- 具体迭代方法详见 https://arxiv.org/pdf/1706.05067.pdf



### ・另一个做法

- 对于两个点集A,B来说,它们之间有AB条边,如果有 K 条 边是相等的,则它们分开的概率为  $(\frac{1}{c})^K (1 \frac{1}{c})^{AB-K}$  。
- 我要在图上找一个割,使得这个概率最大。
- 为了可加,可以给它取  $\log$ ,即 1 边权值为 $-\log\frac{1}{s}$ ,0 的边权值为 $-\log(1-\frac{1}{s})$ ,求全局最小割。
- 全局最小割后,只会分出两个点集,之后像 K 短路那样, 从堆里取出概率最大的,分成两个塞回去即可。



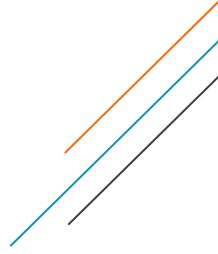
# **Dividing Strings**

### ・大致题意

- 把一个数字串划成若干段(至少两段),使得最大值和最小值的差尽量的小。
- 注意不能有前导 0。
- $N \le 10^5$

#### ・一个显然而重要的性质

- 答案必然 <= 9。
- 因为我们总能把每一段都划成一位数。







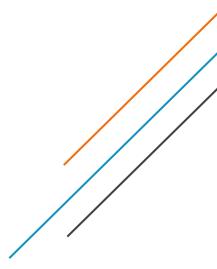
# **Dividing Strings**

#### ・分类讨论

- 我们已经有一个全划成个位数的解了,该如何优化?
- 有两种可能的情况:
  - 1. 划成的每一段长度相等
  - 2. 划成的段里最大段和最小段的长度差为1。

#### ・对于第一种情况

- 因为 N 只有 105, 我们可以枚举 N 的所有约数。
- 对于给定约数,线性扫一遍判断答案、
- 复杂度  $O(N\sqrt{N})$  (而且远远不到)





# **Dividing Strings**

#### ・对于第二种情况

- 因为我们要验证答案是否能小于9,形式只可能是 99···9x 和 100···0y,其中 x=2..9, y=0..7
- 因为 x 不能是 1, y 不能是 9, 容易发现,满足这种构型的方案只有 O(1) 种,稍微特判一下即可。复杂度 O(N)

### • 一些细节的说明

- 第二种情况存在很多需要处理的细节。
- 如果位数是1和2,那么99<sup>···</sup>9x这个构型起始并不是9开始的,而是一个个位数,枚举时需要特判。
- 注意连续的一串 9 可能需要拆成若干个。所以建议去找极 长的 100···0y 的构型去判断。



### **Ancient Distance**

#### ・大致题意

- 给一个有根树, 在树上选择 k 个关键点(根必须选)
- 最小化点到最近关键祖先距离的最大值
- 求出 k 分别为 1,2,...,n 时答案的和

### ・基础暴力

- 当 k 固定时,可以二分答案 x
- 每次选择当前深度最深的点,将它的第x个祖先设为关键点, 并且删除这个关键点的子树
- 直到整棵树被删完,然后比较关键点数量和 k 的大小关系, 来改变二分的范围



### **Ancient Distance**

### ・结论

- 有一个显然的结论,当答案为 x 时,关键点数量  $\leq \frac{n}{(x+1)} + 1$
- 因为按照上面的贪心,每次挑选一个关键点时,至少能删除 x+1 个节点。所以对于一个 k,二分上界是  $O\left(\frac{n}{k}\right)$ 。

### ・做法

- 我们按顺序枚举 1~k,争取每次验证二分时,把复杂度和 N 剥离开来,搞成和关键点数量有关。
- 如果每次验证的复杂度是 关键点数量\*log, 那么容易证明总 复杂度是  $N(log N)^2$  的(还有一个 log 是调和级数)。

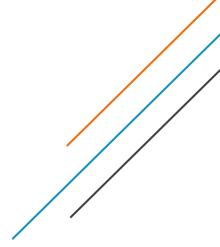




### **Ancient Distance**

### ・如何验证二分的答案 x

- 我们要使复杂度和最终放置的关键点个数有关。
- 容易想到,我们用一颗线段树去维护整棵树的 DFS 序。
- 我们每次的操作是:
  - 1. 找到最深的还未被染色的节点 p。
  - 2. 将 p 向上 x 步的节点 q 置为关键点, 把 q 的子树区间染色。
- 这样每次验证的复杂度是 关键点 \* log N 。





#### ・大致题意

- 黑板上有 N 个互不相同的数, 且 N 是奇数。
- 每次选取连续的三个数,只将它们的中位数保留。
- 重复上述操作直到剩下一个数字。
- 对于原来的每一个数字 x 都询问:是否存在一种擦除方法, 使得最终剩下的数是 x。
- $N < 10^6$



### ・先考虑暴力

• 我们依次枚举一个数 x, 判断是否能将它保留。

### ・一个有用的转化

- 对于枚举的 x, 把大于它的数看成 1, 小于它的数看成 0。 我们的目的是把所有 01 都消掉最终剩下x。
- 000 $\rightarrow$ 0, 010 $\rightarrow$ 0, 011 $\rightarrow$ 1, 111 $\rightarrow$ 1, 01x $\rightarrow$ x, 00x $\rightarrow$ 0, 11x $\rightarrow$ 1

#### ・一个重要的引理:

• x 左右的 01 无关。即如果原问题有解,一定存在一个方案是: 把左右单独做一遍消除(只剩下一个或者两个数字),最后左右和中间的 x 再合并消除剩下 x。



#### ・引理证明

- 我们可以认为,在左右各自消了一会的某个时间点后**都是 跨 x 的消除** (我们总能把之后的左右单独消这一步提前做)
- 注意: 左边提供 2 个数结合 x 的消法也可以看做是左右单独的消除 (而不是跨 x 的消除) , 因为这样只能是 01x 或者 10x, 可以在一开始就结合那一侧的数消掉这对 01。
- 由于 0x0 和 1x1 会使 x 消失,跨区间的只能形如 0x1 或 1x0 (然后01同时消失)。所以在最后跨区间消的时候, 左右01的个数一样且互补。如果跨区间消出现了至少三次, 我们可以单独把左边的三个和右边的三个在之前单独消
- 所以在最后的互补状态里,左右最多两个数字。





- ·基于引理和跨区间消除的性质,我们可以把原问题转化成一些 左右独立的询问(每次问消到这种状态是否可行):
  - 1. 左 0/00 右 1/11
  - 2. 左 1/11 右 0/00
  - 3. 左 01/10 右 01/10
- ・如果能满足以上的某一条 (左右都能消成对应状态), 就有解。





### · 先考虑如何判断一侧是否能留下全 0 (全 1 也一样)

- 核心想法:尽量凑三个1进行消除,使得0剩下的更多。
- 如果原来就有连续三个1相邻,直接消除肯定没问题。
- 存在一种情况类似于 11011, 这时候是两段 1 要合并再一 起消除。这样损耗了 3 个 1 和 1 个 0.
- 从左到右贪心做,维护前缀 0 的个数 zero (0我们都留着不消)以及紧接着一段连续 1 的个数 one (one 的值最多只会是 2,一旦大于等于 3 就立刻消除)。如果新来的数是 0 且 one 有值,带来的效果就是 one-- (这组01被消掉,以利于one这段1和后面的1合并)。
- · 做到最后如果 one > zero, 那么能留下全 0。





### ・如何判断能否留下 01 (或 10) 呢?

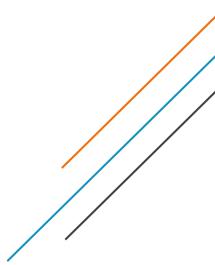
- 重要性质: 如果 0 和 1 的个数一样, 一定能消除成 01/10。
- 证明:因为我们每次可以选择相邻的01,再随便加一个数就可以把他们成对地消除。
- 考虑之前的的贪心做法,它极大地留下了 0 的个数。所以对于一个 01 序列 (假设 0 出现的多),我们可以先套用上述算法,直到做到某个时刻 01 个数一样时就合法了。
- 这些做法都是线性的,总复杂度 O(N2),需要优化。





### ・用一定顺序去枚举 x

- 假设我们从小到大去枚举 x, 那么相当于是 01 序列里不断 有 1 变成 0, 每次要快速做之前的两种判断。
- 我们用 (zero<sub>i</sub>, one<sub>i</sub>) 代表之前的贪心算法做完第 i 个数后的状态。 随着 x 的枚举,这些 pair 的前者变大后者变小。
- 当第 i 个数从 1 变成 0 后, 考虑 (zero<sub>i\_1</sub>, one<sub>i\_1</sub>):
- 1. 如果 one;-1=2,则 one;~one,不变,zero;~zero,不变。
- 2. 如果 one;-1=0,则 one;~one,不变, zero;~zero,加一。
- 3. 如果 one<sub>i-1</sub>=1,则要不 zero<sub>i+1</sub>~zero<sub>n</sub>加一 (如果 a<sub>i+1</sub>=0) 要不 one<sub>i+1</sub>~one<sub>n</sub>不变,zero<sub>i+1</sub>~zero<sub>n</sub>不变 (如果 a<sub>i+1</sub>=1)





### ・复杂度分析

- 当第 i 位从 1 变成 0 后, 我们从第 i 位开始重新修改。
- 由以上分析,要不 (zero, one) 的值迅速收敛 (和上一次保持一致) ,那我们就直接 break; 要不 zero 的值全都集体 +1, 这样这次更新的确可能影响 O(N) 个位置。
- 第一种贪心算法要求判断 zero > one 是否可行。所以当 zero; >=3 后, i 以及 i 之后的点必然合法。每个点的 zero 最多增加三次,由均摊分析复杂度是线性的。
- 对于第二种贪心算法,我们只需在 1 比 0 多的时候调用 zero >= one, 0 比 1 多时反一下。出题人在这里偷懒用了一个树状数组,总复杂度是 **0**(N log N)





### ・大致题意

- 给出一张无向带权图,定义一条路径的权值为所有经过边的权值中的次大值。
- 求两两点对之间次大值之和。

### ・基础暴力

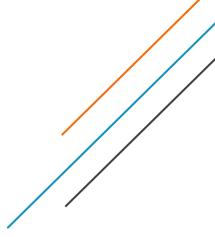
- 不妨枚举次大边的权重,此时我们考虑将大于次大边的边权 重赋值为1,其他边赋值为0。
- 此时若 (i,j) 之间的最短路径 <=1,则说明 i,j 的次大值<=当前 枚举权值, 否则必然大于。
- 记 Ans[k] 表示最短路<=k 的点对数,每次通过 Floyd  $O(N^3)$  直接跑,作差之后统计,总复杂度  $O(MN^3)$





### ・暴力优化

- 上述问题可以这样转化: 当我加入一些边之后, 所有的点会由权值为 0 的边联通成一些连通块。
- 此时,对于所有权值为1的边,他所能带来的点对数量是其两边连通块的 size 和。那我们不妨枚举每条当前权值为1的边,然后将每条边的答案合并起来。当然要注意去重,也就是如果两条边连接两个相同的连通块,只能被计算一次。这样就可以做到  $O(NM \log N)$ 。
- 更进一步,我们只需要计算每次新增的点对数,假设这一步新增的边将接通连通块 x 和 y,新增的点对数量应该是 size[x] \* (y的出边 size 和) + size[y] \* (x的出边 size 和) (size[x] + size[y]) \* (x 与 y 的出边交集)。





### ・出边和统计

- 为了统计每个连通块的出边和,对于点x,我们定义其权值为出边的数量,一个连通块的权值为所有点的权值之和。
- 对于每个连通块,我们维护其所有出边,这个出边直接表明了另一个连通块的代表标号。当我们将 y 合并到 x 上时,需要将 y 中所有边对应修改成以 x 为顶点的边。方便点,使用启发式合并和 set 来维护,复杂度为  $O(M(\log M)^2)$
- 对于每个连通块,若其权值第一次大于√M,则扫描所有
   边,并且累加对应的 size 之和,注意到上述情况一共只会
   发生√M次。



### ・出边和统计

- 对于权值不超过 √M 的点,我们扫描所有边直接询问。否则,我们直接调用已经得到的值.
- 当然我们需要维护所有"大"点的值。为此,每个点需要维护 一个其所有连出的"大"点列表。每次发生修改时,需要同步 修改所有会发生改变的权值。维护该列表的方法与维护出 边表相同,复杂度也不会劣于之前的复杂度。
- 据此,复杂度为  $O(M\sqrt{M} + M(\log M)^2)$ 。算法中某些部分需要一些简单的分类讨论,留给选手自己思考。



# **Geometry Challenge**

#### 题意

- 给出若干个符合一定规则的几何题条件。
- 要求出某个角或者某条边或未知数 x 的值。

#### ・核心点

- 注意题目保证了每一步的结果依然是 expressions。
- 核心思路:我们不断用已知的条件和定理扩展我们知道的量,直到找到答案。中途可能需要构方程和解方程。
- 整个过程有点像迭代加深搜索。



# **Geometry Challenge**

#### ・一些注意点

- 同一个角的表示有很多种,注意要对它们建立 Equal 关系。
- 对于所有 PointLiesOnLine, 要构建边长相加的 Equal 关系和角度相加的 Equal 关系。
- 对于平行线,同位角可能不存在,所以可以只用内错角和 同旁内角建立关系。
- 可以先不断地用"基本定理"传播 expression, 到最后再用勾股定理、相似定理等复杂的定理解方程。
- 要能检测出图中的三角形(只要三点不共线就算),并积极寻找全等和相似的三角形对。



# **Geometry Challenge**

#### ・一些注意点

- 在运用一些定理时,所用的条件可能不是"完全"的。比如我们知道两条边的长度都是 2x+3,我们依然可以利用它们相等来构建 对角相等 或者 三角形全等 的关系。
- 可以用 Find 导向去优化搜索方向。不过这道题是不必要的,你把所有可以求的量求出来也是 OK 的。
- 在解出 x 后,要把之前所有用 x 表示的表达式都带入一遍。
- 复杂度为 所有状态量 \* 定理数量 \* 步数上限 (4)。





# Thanks

