1 LaTang's Number

参考原题: https://leetcode-cn.com/problems/check-if-it-is-a-good-array
题目类型: 数论
考察知识: 最小公因数, 扩展欧几里得

由扩展欧几里得, 裴蜀定理 (贝祖定理) 可得
$ax + by = c$
其中 c 是 $\gcd(a, b)$ 的倍数
要想得到 1 , 则要 a, b 互质
a, b 可以是数组中任意两个, 也可以是任意个通过变换得到
或者说我们找到 $k_{p_1} \times a_{p_1} + k_{p_2} \times a_{p_2} + .. = z_1$
$k_{q_1} \times a_{q_1} + k_{q_2} \times a_{q_2} + .. = z_2$
使得 $\gcd(z_1, z_2) == 1$ 即可
负数并不影响, 提前乘上 -1 当作正数
其中 a_i, a_j 能组成 $\gcd(a_i, a_j)$ 的倍数的数
再有 a_k, a_l 能组成 $\gcd(a_k, a_l)$ 的倍数的数
(其中 i 可能等于 j, k, l 等等)
那么这四个数又可以组成 $\gcd(\gcd(a_i, a_j), \gcd(a_k, a_l))$ 倍数的数
所以所有的数能组成 $\gcd\limits_{1 \le i \le n} a_i$ 的倍数的数
因为不断做 gcd 只会使结果越来越小, 也就是能组成的数的范围越来越大
最后判断所有数的最大公约数是否为 1 即可
时间复杂度 $O(n \log n)$

2 LaTang's Game

参考原题: https://atcoder.jp/contests/abc170/tasks/abc170_f
题目类型: 图论
考察知识: 最短路

对于 $k = 1$ 的情况, 就是裸的最短路
对于 $k \geq n$ 的情况, 就是走相同颜色不消耗体力
对于 $m = n$ 的情况, 其实起点到终点就是一条链, 可以乱搞
对于 k 比较小的情况
可以在做最短路时额外记录一个状态, 表示当前经过了几个连续的相同颜色
例如记 $dis[i][j]$ 表示到第 i 个点, 连续经过了 j 个颜色 $c[i]$ 的点的最少体力
然后把每对 (i, j) 当做一个点跑最短路, 这种题型应该不少见
复杂度 $O(nk \log(nk))$
对于 k 比较大的情况, 这种题有一个小技巧
最多可以连过 k 关颜色相同的转化为过一关需要 $\frac{1}{k}$ 体力
如果当前场地和下一个场地颜色不同, 就将当前体力向上取整即可
为了避免精度问题 (虽然不会有) 可以先将体力乘上 k 到最后除掉即可
复杂度 $O(n \log n)$

3 LaTang's Fight

参考原题: http://acm.hdu.edu.cn/showproblem.php?pid=6856
题目类型: 数据结构 + 动态规划
考察知识: 区间最值, 线段树, 动态规划, 离散化, 单调队列

首先对于 $L=1, R=n$ 的情况可以考虑贪心做法
以小辣为例, 小汤就稍微反一下嘛
为了使正数贡献最大, 正数段尽量多, 所以最好一个一段
负数段尽量少, 所以尽可能将连续的负数合并
0 不影响, 再考虑让正数吞掉负数, 也就是正数和负数合并还是正数
我们可以用栈来实现这一过程
遇到负数, 如果前面是正数可行的话被吞掉, 不可行就入栈, 如果前面是负数, 就合并
遇到正数, 如果前面是负数, 可行的话就尽量吞掉前面的负数, 如果是正数就直接入栈
可以发现一个正数可以吞前面和后面的若干负数, 以上做法好像是优先吞了前面的
是否有前后都可以吞但只能吞一个, 选择吞后面的更优的情况呢
这就是个贪心的过程, 吞前后贡献都为 1
而后面的负数还可能被下一个正数吞, 所以贪心吞前面的
按上述方法即可 $O(n)$ 解决该问题
其他的情况容易想到一个 $O(n \times (R-L))$ 的 DP 做法
记 $dp[i]$ 为数组前 i 位的最大高兴值
小辣 $dp[i] = \max\limits_{i-R \le j \le i-L} dp[j] + \frac{sum[j+1,i]}{abs(sum[j+1,i])}$
小汤 $dp[i] = \max\limits_{i-R \le j \le i-L} dp[j] - \frac{sum[j+1,i]}{abs(sum[j+1,i])}$
现在考虑用数据结构优化, 还是以小辣为例
发现状态转移无非就是三种情况 +1, 0, -1 那么不妨分类讨论
求区间和的操作一般采用前缀和 $sum[j+1,i] = presum[i] - presum[j]$ 发现
如果 $presum[i] > presum[j]$ 则 $dp[i] = \max(dp[i], dp[j]+1)$
如果 $presum[i] = presum[j]$ 则 $dp[i] = \max(dp[i], dp[j])$
如果 $presum[i] < presum[j]$ 则 $dp[i] = \max(dp[i], dp[j]-1)$
也就是说有如下转移
$dp[i] = \max(dp[i], \max\limits_{presum[i]>presum[j]} dp[j]+1)$
$dp[i] = \max(dp[i], \max\limits_{presum[i]=presum[j]} dp[j])$
$dp[i] = \max(dp[i], \max\limits_{presum[i]<presum[j]} dp[j]-1)$
那么我们就想要快速求出
$\max\limits_{presum[i]>presum[j]} dp[j]$
$\max\limits_{presum[i]=presum[j]} dp[j]$
$\max\limits_{presum[i]<presum[j]} dp[j]$
想到区间最值, 因为需要修改, 不妨使用单点修改区间查询的区间最值线段树

以前缀和 $presum$ 的值 (离散一下) 作为下标建树即可
该做法复杂度 $O(n \log n)$
还要考虑区间长短的要求, 这个应该不难维护
$\forall i, j \in [i - R, i - L] \&\& j > 0$
可以用 $multiset$ 维护不过要慢一点 (不过实际上不会慢太多吧
一个优秀的做法应该是用单调队列维护, 复杂度是线性的
还有一种方法是在离散化的时候按照前缀和大小离散但仍给每个点一个不同的值
记录一个前缀和值对应的离散后的区间, 就能处理出分类转移的区间
修改删除直接在那个点操作即可
给出的标程中 fight1.cpp 是单调队列做法, fight2.cpp 是神奇离散操作