题解

1、交通 (traffic)

题解

将时间按模 $g+r$ 来考虑。

从起点出发，若到达一个路口时碰到红灯无法通过，则会等到绿灯后立即通过，剩余所需时间等于从该路口第 0 秒出发到终点的时间。

问题转化为求从起点出发，遇到第一个红灯时所在的路口，以及求 f_i：从第 i 个路口第 0 秒出发到达终点所需时间。

求 f_i 与求解询问的做法无异，因此考虑从 n 往前倒推 f_i。剩余问题转化为求遇到的第一个红灯路口。

套路性地，我们把每个路口的红灯时段先在模意义下平移一下，这样就不用考虑走路的耗时了。

从后往前推的时候维护每一个时间点下一次会碰到哪个红灯，需要支持区间取 $\min$ 和单点查询，线段树即可。

可以做到 $\mathcal{O}((n+q)\log n)$。

2、选拔 (selection)

题解

这题不是一个正宗字符串题。你也许会想用哈希、树上后缀排序、广义 SAM 这类东西来做，但你会发现这些非常难搞。

注意到一个询问串在树中出现的方式一定是一条从下到上的路径和一条从上到下的路径（两个都可以为空）。

DP，记 $f_{u,j,k}$ 表示从下往上走到 u，是否可能匹配 $t_j[1:k]$，$g_{u,j,k}$ 表示从 u 向下走，是否可能匹配 $t_j[k:l_j]$（l_j 为第 j 的询问串的长度）。后两维总共就 $\mathcal{O}(s)$ 的大小，因此把相邻询问串间加入分隔符，拼在一起转移就可以。

而这个 DP 显然可以 bitset 优化。

时空复杂度 $\mathcal{O}(ns/\omega)$。

比较坑的是，用 DFS 做树上 DP 似乎要消耗很多栈空间，改成先求拓扑序再 DP 才能过。

3、等待 (wait)

题解

为了区分，以下设 a_i 表示第 i 个整数的初值，b_i 表示第 i 个整数的最终值。

算法一

考虑二分答案，即将答案从高到低决策该位是否可以为 0。这里二分的具体细节会导致复杂度不同，这里给出其中一个朴素二分中接近最高效的算法。

首先对答案进行估计：位数应当不超过 $n+\max L$。

其次我们考虑如果给出一个数，如果快速判断答案能否不大于它：我们从高到低分配给出的数中的每一个 1：我们将每个数当前最高位的 1 放在一个数组里。

- 如果当前所有数的最高位已经超过未分配数的最高位，则无解。
- 否则如果相等，则我们只能将当前位分配给该数（如果有超过一个显然还是无解），可以看做该数直接将这一位去掉，如果有次高位，将次高位加入数组中。
- 否则我们只要将这位分配给一个数，则该数直接清零，（考虑在一个可行解中交换分配的位）不难证明如果有解，我们可以用当前位将剩余的最大的数清零。
- 考虑如何维护最大的数，我们预先将每个 1 和它同位置有 1 的数在一起排序，这个可以通过按顺序的基数排序来解决。然后我们每次取最大的数。
- 看起来上述方法需要排序，实则不然，我们只需考虑在这位的数中是否会有一个没有被清零，因此我们并不需要执行排序，而只需要把最小的数放在最后处理。

由于每次考虑一位最多使得数组里插入一个元素，所以数组里的总元素是 $\Theta(n+\max L)$ 而非 $\Theta(\sum L)$。

时间复杂度：$\Theta(\sum L+(n+\max L)^2)$。

算法二

考虑所有数都只有一个 1 的情况。

假设这些数从大到小排的最高位分别是 $L_1, L_2, \dots L_n$，那么答案的最高位可以发现是 $B=\max\{L_i+i-1\}$。

这显然是一个下界，因为第一个达到这个数的 i，只考虑 $1\sim i$ 的这些数就必须至少占领这位。

而这显然有解，因为考虑将最大的数调至该位置，剩下的数对应的 B 值至少 -1，归纳即得。

时间复杂度：$\Theta(n\log n+\sum L)$。

算法三

上述观察事实上对于正解有着极大的意义。我们考虑将所有数都只保留最高位，得到的数列记为 a'_i，那么原数列的最高位应当在 B 和 $B+1$ 之间。只需注意一点：将数列 a'_i 每个数都左移 1 位，那么得到的数均大于对应位置的 a_i，而这个数列的答案的最高位是 $B+1$!

我们考虑优化二分的过程。此时我们的决策要做的就更加明确了：确定 $B+1$ 这一位是否可以是 0。

考虑还有一些不用二分的位置：假设 B 取到的第一个位置是 k，也就是说对于 $i<k$，都有 $L_i+i-1<B$，那么无论我们是从 $B+1$ 开始赋值，还是从 B 开始赋值，这些数都一定不用在后面的计算中被考虑了，从整体上看，从 B 位到 $B-k+2$ 位的值都已经必然是 1 了，而二分的结果其实额外决定的是对于前 k 个数，还有一个 1 是填在 $B+1$ 位置还是 $B-k+1$ 位置。

我们考虑先假设 B 位足够，那么第 k 个数就需要减去最高位后考虑剩下位的贡献，它会被重新插入进序列进行排序，我们可以通过算法一预先进行的基数排序，算出每个 1 位置作为数的后缀，整体之间的序，然后用一个线段树来快速维护。

我们假设当前的二分如果返回可行，则已经计算出剩余部分的最优解。这样一来，二分的额外复杂度就是所有的返回“失败”的情况的复杂度之和。而在我们进行二分的时候，剩余的位数已经卡到了某个 L_k，如果二分失败，那么这一部分的复杂度就是 $\Theta(L\log(\sum L))$，注意到引发一次失败后，这个数就在接下来的过程中直接消失，因此不会被再次贡献复杂度。因此时间复杂度：$\Theta\left(\left(\sum L\right)\log(\sum L)\right)$。