滚球

我们想要丢掉尽可能少的球使得剩下的球不发生碰撞，相当于保留尽可能多的球使得它们不发生碰撞。

“最早碰撞的时间最晚”启发我们二分答案 mid。

我们将球按照位置 x_i 从小到大排列，容易发现最早发生碰撞的一定是某两个相邻的球。

对于两个球 $i, j(i < j)$，如果它们相邻，那么不会碰撞的条件是 $(x_j - x_i) > mid \cdot (v_i - v_j)$，也就是 $x_j + mid \cdot v_j > x_i + mid \cdot v_i$。

令 $b_i = x_i + mid \cdot v_i$，那么条件就是对于 $i < j$，如果 $b_i < b_j$，那么 i 和 j 可以相邻。

设计 DP，$f(i)$ 表示考虑前 i 个球，i 被保留下来时，最多能保留多少个球。

DP 的转移：枚举 i 的下一个球 j，j 和 i 能相邻的条件是 $b_j > b_i$。转移方程就是 $f(j) \xleftarrow{\max} f(i) + 1$。

能发现这个 DP 其实求的就是 b 序列的最长上升子序列长度，用 二分栈 或者 离散化 + 树状数组 优化即可。

一种另外的理解方式：

如果两个球发生碰撞，那么它们的相对位置会发生改变：本来 i 在 j 前面，走着走着 i 到 j 后面了，那它们一定会发生碰撞，否则就一定没有发生碰撞。

二分答案之后，我们就知道了每个球的终止位置，为 $x_i + mid \cdot v_i$。

我们按照起始位置 x_i 排序，要选出最多的球使得这些球终止后的相对位置不变，那么就是对 $x_i - mid \cdot v_i$ 求最长上升子序列。

首都

本题难点在于建图。

首先建 n 个点，第 i 个点表示第 i 个地点，且不在任何交通上（称为第 i 个地点，记为 A_i）。

我们有两种独立的交通：单车、地铁，我们通过增加节点数量来分开这两种交通。

具体地，

- 对于单车：

  我们新增 n 个点，第 i 个点表示我们在第 i 个地点，且骑着单车（称为第 i 个单车点，记为 B_i）。

  扫描和锁车总共需要花费 x 的时间，我们可以只在上车的时候计算这个花费的时间。

  我们可以在第 i 个地点骑上单车，也可以下单车。那么我们从 A_i 向 B_i 连一条**有向边**，边权为 x，表示我们骑上了单车；从 B_i 向 A_i 连一条**有向边**，边权为 0，表示我们从单车上下来了。

- 对于地铁：

  同样的，我们新增 n 个点，第 i 个点表示我们在第 i 个点的地铁站里（记为 C_i）。

  进站和出站分别要花费 e_i 的时间，那么 A_i 向 C_i，C_i 向 A_i 都连有向边，边权均为 e_i，表示进站与出站。

  我们还有若干条地铁线路，以非环线为例：

  对于每条地铁线路，假如一条线路有 k 个点，我们会新建 $2k - 1$ 个点：

- 对于前 k 个点，第 i 个点表示我们正在这条地铁线路上，到了第 i 个站点，
- 对于后 $k-1$ 个点，我们用来回程。

这些点之间均连有向边，因为地铁只会往一个方向开。边权即为从这一站到下一站需要花费的时间。

对于地铁线路上的点，我们需要从地铁站向它连有向边，表示上地铁。

我们还需要换乘，可以通过再新建 n 个点来解决：第 i 个点表示第 i 个地铁站的换乘点。（也可以不用建换乘点，直接连向地铁站即可。但是这样建点的话直观一些）

我们发现对于上地铁的那些边，我们不能直接连一条带边权的边。因为它是按地铁的到达时间，以及你当前的时间来算的。

由于地铁发车间隔固定，我们可以在这条边上记录：8:00 出发的地铁到达这个点的时间、发车间隔。然后就能算出实际上车时间了。

对于环线，只需要多考虑反向发车的内容即可。

最后只需要在这张图上跑个最短路即可。

惩罚

考虑如何 check 一种方案是否合法。能够发现，对于两个叶子之间的路径，去掉 LCA 以外，如果路径上没有被选中的点，那么这两个叶子的权值必定相同。否则就合法。

考虑树形 DP。设 $f(x,0/1)$ 表示 x 子树中有 $0/1$ 个叶子到 x 路径上没有选中点的最小代价。

考虑转移，

- $f(x,0)=\min\left\{\sum_{y\in son(x)} f(y,0), \min_{z\in son(x)}\left\{f(z,1)+c(x)+\sum_{y\in son(x),y\neq z} f(y,0)\right\}\right\}$
- $f(x,1)=\min_{z\in son(x)}\left\{f(z,1)+\sum_{y\in son(x),y\neq z} f(y,0)\right\}$

容易优化至线性。

方案数只需要再设一个 $g(x,0/1)$ 表示当前状态的方案数，在 f 转移的时候顺便维护一下即可。

最后还需要求哪些点可能被选中：$f(x,1)$ 代表了 x 一定没有选，若 $f(x,1)+c(x)=f(x,0)$ 并且 $(x,0)$ 可以存在于某种最优解，那么 x 就是可以选的。倒着做一遍 DP 即可得知哪些状态可以存在于最优解中。

NOIP2024 充满了希望

我们先来考虑修改操作。

一操作和二操作显然是白给的，对于三操作，我们如何维护呢？

这其实是一个非常经典的均摊：对于一个数 x，它成功被取模 $\log x$ 次就会变成 0。

这也是很好证明的，有一条性质：如果 x 成功被 p 取模了，那么 x 会变成小于 $\frac{x}{2}$ 的数，即 $(x \bmod p)<\frac{x}{2}$。

x 能够被 p 取模，那么首先有 $p\le x$。若 $p>\frac{x}{2}$，那么 $(x \bmod p)=x-p$，显然小于 $\frac{x}{2}$；若 $p\le\frac{x}{2}$，则 $(x \bmod p)<p\le\frac{x}{2}$。

所以 x 成功取模一次就会减半，$\log x$ 次之后就变成 0 了。

回到操作三。对于 n 个数，假如我们一个一个进行取模，每次都成功取模，那么只需要 $O(n\log V)$ 的时间。

也就是说，我们每次只需要找到区间中一个能取模的数即可。能取模意味着这个数 $\geq v$，若区间中没有 $\geq v$ 的数，说明这个区间没有数能够成功取模，就不必对这个区间进行操作了。

我们用线段树来维护这个序列。一个线段树节点维护区间的最大值，如果最大值 $< v$，那么就停止递归；否则往左右儿子递归，直到叶子节点，此时我们就找到了一个能取模的数。

至此我们完成了修改操作的部分。

关于部分分，其实 $r_i < l_{i+1}$ 是有一点启发的：如果 x 被影响了，那么一定是有一个操作区间覆盖到了它，然后被这个区间影响了。

考虑询问怎么回答。为了防止变量名重复（询问的 l, r 和操作区间的 l_i, r_i），我们不妨改成依次进行 L 到 R 的操作。

我们倒着考虑：R 到 L 倒着看，会有一个区间 $[l_t, r_t]$ 包含了 x，x 就被这个区间影响到了。从 t 再往前看，如果有个区间 $[l_p, r_p]$ 和区间 $[l_t, r_t]$ 有交，那么 x 会被 $[l_t, r_t]\cup[l_p, r_p]$ 里的元素影响，而显然两个区间有交，它们的并仍然是一个区间。因此 x 总是被某个区间影响到，并且 x 最终的值为这个区间中的最大值。

更具体地，我们维护一个区间表示 x 受影响的区间。一开始这个区间为 $[x, x]$，从 R 往 L 扫，如果我们的区间和当前操作区间有交，那么就和这个操作区间取并集。这样我们就能得到最终受影响的区间。

我们希望加速这个扫描的过程。我们首先找到包含 x 的第一个操作区间，然后与 x 就无关了（影响区间就是这个操作区间，如 $[l_t, r_t]$）。然后，如果它和一个区间 $[l_p, r_p]$ 取并集后，左端点变化了，那么左端点一定变成了 l_p。此时一定满足 $l_p < l_t \leq r_p$，也就是 l_t 在区间 $[l_p, r_p]$ 中。这和 r_t 是无关的，所以左右端点可以分开考虑。不妨考虑左端点。

记 lef_i 表示对于操作区间 i，往前第一个包含 l_i 的区间是什么。那么询问时就可以倍增找到最终的左端点了。右端点也是一样的。

最后一个问题是：如何找到第一个包含 x 的区间，将询问离线下来，扫描线即可。