1 讲题大会

1.1 T1 环

这个题目有两种做法，快速幂、倍增。

快速幂，二进制优化。首先由于乘法具有结合律，所以我们有：

$$a^{13} = (\underbrace{a \times a \times \cdots \times a}_{8个a})(\underbrace{a \times a \times \cdots \times a}_{4个a})(a)$$

所以我们可以对 13 这个数进行二进制分解，分解成 $(1101)_B$ 的形式。最高位的 1 看做 a^3，第二位的 1 看做 a^2，最后一位的 1 看做 a^0，然后把它们一一乘起来。

然后可以以 $O(\log r)$ 的复杂度计算 n^r.

快速幂取模，因为乘法和取模可以分配，所以放到快速幂运算过程中取模。特别要注意精度问题，如果是对 10^9+7 取模，需要先转换成`long long`的形式然后再做。

本题本身是个类似函数的复合的题目，可以使用类似快速幂的思想去计算。函数的计算必须从内向外做，所以要注意计算的顺序的问题。

对 m 进行二进制分解，假设需要跳 13 层，利用快速幂，从内层向外层跳 1 层+4 层+8 层。

注意与快速幂有一点不同，需要使用倍增算法进行计算。例如 $f^2(x) = f(f(x))$，$f^4(x) = f(f^2(x))$，$f^8(x) = f(f^4(x))$，利用这个递推性质，我们就可以利用倍增算法计算任意函数值了。

类似的有关做法：考虑有一个排列 $(1,3,2,5,4)$，有 $f(1) = 1, f(2) = 3, f(3) = 2, f(4) = 5, f(5) = 4$，把它们看做一个映射，用一个有向的箭头把 x 与函数值连接起来，可以发现，用这种方式表示一个排列，一定构成了若干个环，证明思路是双射+不重复的环，所以可以运用一个类似 DFS 的操作预处理出排列中所有的环。现在需要计算 $f^m(x)$，假设这个环的长度为 k，所以我们只需要等价的移动 $m \bmod k$ 步，再利用数组下标映射一下。例如需要计算第 i 个元素的位置，那么可以计算 $(i + m \bmod k) \bmod k$。

读入的方法：设读入的行数为 m，已知 $m = n + 2q$，若 m 为奇数，则第一行一定是排列中的一个数。剩下的就是偶数的情况，两个两个的检查第一个数是不是在之前出现过，如果出现过就说明它一定是开始询问的部分，因为数据保证一定合法。

1.2 T2 礼

思路基本上就是一个折半搜索。我们有一个集合 a，要从中选择一个子集 b，使得 $b \subseteq a$。显然我们有一个暴力的想法，直接枚举 a 的每一个子集，可以通过 50%的数据。

考虑全部的数据，能否有一个 $O(n^4)$ 的算法，不太可能。那么有没有一个 $O(2^{\sqrt{n}})$ 的算法？答案就是这样。

利用 meet-in-the-middle 的算法，我们枚举集合 s_1，使得 $s_1 \subseteq a$的左半部分，类似地枚举 s_2，最终的答案是 $sum(s_1 + s_2) \bmod m$。我们可以分两部分讨论，若 $s_1 + s_2 < m$ 则只需要找一个尽可能大的值；考虑当 $s_1 + s_2 \leq m$ 的情况，如何才能使得 $s_1 + s_2 \bmod m$ 最大？

预处理出 s_2 的所有取值，一共有 2^{17} 个数，使得 $s_2 < m - s_1$ 并且 s_2 尽可能大，所以可以对 s_2 进行排序，然后二分查找一下最大值。

时间复杂度 $O(2^{17} \log 2^{17})$.

骗分的手段：背包。由于 a[i] 不是特别的大，所以可能能骗到很多分数。假设随机生成一组 a[i]，则最大值一般就是 $M - 1$，所以在数据小的情况下，总是要取几个数。

1.3 T3 变

考虑把整个数轴以 a_x 为单位长度划分，则每次跳跃都会跳到距离 a_x 最近的点上。

有几点观察：

1. 贪心的思想，每次操作，都需要让 A 减少值尽量大。

   如果某一步有两种走法，那么令 A 减少值比较大的数的方案在下一步无论进行什么操作，可以到达的位置一定不会比另外一种方案大，即这种方法可以更快地到达 B。

2. 如果 $A - A \bmod a_x$，则说明这个 a_x 再也不会被用到了，因为对于这个 a_x，跳 a_x 步所到达的点一定小于 B，这样就永远也无法到达 B 这个点了。

根据上述两点观察我们即可得出 std 的代码了。`std::set` 支持上述两种操作。

时间复杂度的证明：

考虑在连续若干次迭代的过程中，假设集合 a 的大小不变且为 n，则在连续三次枚举后，A 的值至少会减少 n.

a 这个集合内没有重复元素，且 a 中存在一个 $a_{\max}$，则 $a_{\max} \geq n$.

如果利用这个 $a_{\max}$ 三次，则至少可以让 A 减少 n，因为根据上述的贪心，连续三次操作后，至少跳跃了一个 $a_{\max}$ 的值。

所以进行 $O(n)$ 的枚举，可以保证 A 的值一定可以减小 n。因为 A 最多可以减少 A-B 次，所以可以保证时间复杂度为 $O(A-B)$ 这个量级。

考虑 a 这个集合发生变化的情况，由于满足每个元素仅在集合中出现一次，所以每个元素最多可以被踢掉一次，每次踢元素重新枚举带来的额外的时间消耗为 $O(n)$，所以总的时间复杂度为 $O(A-B+n)$。

如果使用`std::set`，时间复杂度加一个 $n\log n$，是 $O(A-B+n\log n)$，也是可以通过所有的数据的。

注意一定需要去重，否则贪心的性质不成立，因为可能有多个 $a_{\max}$，并且不去重时间复杂度也无法保证。