由于本节课内容较多，将课堂上关于 A、C 两题的内容整理成讲义，供大家参考。

1 Towers

1.1 $O(n^2)$：朴素 dp

首先记 s_i 为 h_i 的前缀和。

容易想到，令 f_i 表示合并到前 i 个位置，所需的最少操作次数。

由于合并后的高度需要满足递增性质，我们记录 g_i 为用最少操作次数合并后 i 位置塔高；如果有多种最少操作次数的方案，我们希望 g_i 尽可能小。

因此，我们可以写出如下 dp

```
for (int i = 1; i <= n; ++i) {
        for (int j = 0; j < i; ++j) {
                if (s[i] - s[j] < g[j]) continue;
                if (f[i] > f[j] + (i - j - 1)) {
                        f[i] = f[j] + (i - j - 1);
                        g[i] = s[i] - s[j];
                }
                else if (f[i] == f[j] + (i - j - 1)) {
                        g[i] = min(g[i], s[i] - s[j]);
                }
        }
}
```

稍加思考我们就能发现，每个 f_i 都是从最后一个满足 $g_j \le s_i - s_j$ 的位置 j 转移过来的，因此我们可以做一点常数优化：

```
for(int i = 1; i <= n; i++){
    for(int j = i - 1; j >= 1; j--)
        if(s[i] - s[j] >= g[j] && f[j] + (i - j - 1) < f[i])
            f[i] = f[j] + (i - j - 1), g[i] = s[i] - s[j];
}
```

1.2 $O(n \log n)$：决策单调性 + 二分优化

注意到 j 位置能够更新 i 位置答案的前提是

$$g_j \le s_i - s_j$$

稍作变形得到：

$$s_i \ge g_j + s_j$$

另外，如果 j 位置能够更新 i 位置，那它就一定也能更新 $i+1$, $i+2$, $\cdots$, n 位置 (*)。

考虑正向 dp，即，对于每个位置 j，用 f_j 来更新它能够更新的位置 i.

首先找到第一个满足 $s_i \ge g_j + s_j$ 的位置 i，这可以用 `lower_bound` 简单实现。然后令 `tag[i]=j`，其中 `tag[i]` 表示能够用来更新 i 的最后一个位置。由于 (*) 所述，我们在使用 `tag[i]` 之前，还需要与 `tag[i-1]` 取 max。

因此我们得到这样一个算法：

```
for(int j = 1; j <= n; ++j){
    // 找到 j-1 能够更新的第一个位置 i
    int i = lower_bound(s + 1, s + 1 + n, g[j-1] + s[j-1]) - s;
    // 给位置 i 打上标记
    tag[i] = j - 1;
    // 使用 tag 之前，要和前一个位置的 tag 取 max
    tag[j] = max(tag[j], tag[j-1]);
    // tag[j] 是最后一个能够更新 j 的位置，因此从 tag[j] 位置转移过来
    f[j] = f[tag[j]] + (j - tag[j] - 1);
    g[j] = s[j] - s[tag[j]];
}
```

1.3 $O(n)$：单调队列优化

下节课会说。

2 吉夫特

2.1 Lucas 定理

设 p 是质数，则组合数取模满足如下性质：

$$\binom{n}{k}\%p = \binom{n/p}{k/p} \times \binom{n\%p}{k\%p}\%p$$

预处理阶乘取模、阶乘逆元取模，我们就可以得到如下算法（伪代码）：

```
C(n, k):
    return (n < k) ? 0 : fac[n] * invfac[k] * invfac[n-k] % p;
lucas(n, k):
    return (n < p) ? C(n, k) : lucas(n/p, k/p) * C(n%p ,k%p) % p;
```

复杂度显然为 $O(\log_p n)$.

2.2 C 题题意简述

给你一个长为 n 的正整数序列 a，其中 $n < 2^{18}$，序列里的数 $< 2^{18}$ 且两两不同。问能够取出多少个长度大于 2 的**递减**子序列 h，满足：

$$\binom{h_1}{h_2} \times \binom{h_2}{h_3} \times \binom{h_3}{h_4} \times \cdots \%2 > 0$$

2.3 题意转化

设 $M = 18$.

我们考虑 $\binom{a}{b}\%2 > 0$ 意味着什么。

我们记 $a\langle k\rangle$ 表示 a 的二进制 **从右往左**第 k 个数（末位是第 0 个），根据 Lucas 定理，有：

$$\begin{aligned}\binom{a}{b}\%2 &= \binom{a/2}{b/2} \times \binom{a\langle 0\rangle}{b\langle 0\rangle}\%2 \\ &= \binom{a/4}{b/4} \times \binom{a\langle 1\rangle}{b\langle 1\rangle} \times \binom{a\langle 0\rangle}{b\langle 0\rangle}\%2 \\ &\vdots \\ &= \binom{a\langle M-1\rangle}{b\langle M-1\rangle} \times \cdots \times \binom{a\langle 1\rangle}{b\langle 1\rangle} \times \binom{a\langle 0\rangle}{b\langle 0\rangle}\%2\end{aligned}$$

因此，$\binom{a}{b}\%2 > 0$，当且仅当 $a\langle i\rangle \geq b\langle i\rangle$ 对每个 i 都成立。也就是说，b 是 a 的二进制子集。

因此，题意被我们转化为：能够取出多少个长度大于 2 的子序列 h，满足 h_{i+1} 是 h_i 的二进制子集。

2.4 dp 求解

我们设 $f_{i,S}$ 表示在 $a_i, a_{i+1}, ..., a_n$ 中，令第一个数为 S，能够取出多少个子序列。

考虑 $f_{i,S}$ 所表示的序列。如果 $S = a_i$，那么它就是以 a_i 开始的，下一个数 x 必须是 a_i 的二进制子集；否则，它就是以后面的某个数开始的，因此当 $S \neq a_i$ 时，$f_{i,S} = f_{i+1,S}$.

这样一来，我们就可以写出转移（伪代码）：

```
for i (倒着枚举)
    f[i][...] = f[i+1][...];
    for (x 是 a[i] 的二进制子集)
        f[i][a[i]] += f[i+1][x]
    f[i][a[i]]++
```

我们发现第一维可以优化掉，直接写成（伪代码）：

```
for i (倒着枚举)
    for (x 是 a[i] 的二进制子集)
        f[a[i]] += f[x]
    f[a[i]]++
```

上述代码中的 `f[a[i]]++` 表示一个以 a_i 组成的长度为 1 的子序列。因此，如果我们要把统计答案写进上述过程中，为了保证统计到的答案是长度至少为 2 的子序列，`ans += f[a[i]]` 必须发生在 `f[a[i]]++` 之前。

2.5 二进制子集枚举

枚举 a 的二进制（真）子集有一个常用的套路：

```
for(int x = a & (a-1); x; x = a & (x-1))
```

这个方法可以感性理解一下。举例而言：

```
a      = 11110001110000
a-1    = 11110001101111
x      = 11110001100000
x - 1  = 11110001011111
下一个x = 11110001010000
```

2.6 一种不好的复杂度分析

想象最坏情况：$a_i = 2^M - 1$，那么枚举 a_i 子集的复杂度是 $O(2^M)$。

一共有 n 个 a_i，因此复杂度是 $O(n \cdot 2^M)$，炸了。

这种分析并不算错误，只能说“不好”。因为他把“最坏情况”估计得太坏了。

2.7 合理的复杂度分析

我们应该这样估计最坏情况：$n = 2^M$，$a_1, ..., a_n$ 取遍 $0, 1, ..., 2^M - 1$ 中的所有数。

如果一个 a_i 的二进制具有 p 个 1，那么枚举它的子集的复杂度是 $O(2^p)$。而具有 p 个 1 的 a_i 一共有 $\binom{M}{p}$ 个。因此，枚举子集的总复杂度为：

$$\sum_{p=0}^{M} \binom{M}{p} 2^p = \sum_{p=0}^{M} \binom{M}{p} 2^p 1^{M-p} = (2+1)^M = 3^M \tag{1}$$

这里用到了二项式定理。

根据这个复杂度分析，我们发现 $O(3^M)$ 的复杂度是能够承受的。