【第四十六课】状态机模型与语言解释器（二）

2048. 下一个更大的数值平衡数

如果整数 `x` 满足：对于每个数位 `d`，这个数位 **恰好** 在 `x` 中出现 `d` 次。那么整数 `x` 就是一个 **数值平衡数** 。

给你一个整数 `n` ，请你返回 **严格大于** `n` 的 **最小数值平衡数** 。

示例 1：

```
输入：n = 1
输出：22
解释：
22 是一个数值平衡数，因为：
- 数字 2 出现 2 次
这也是严格大于 1 的最小数值平衡数。
```

```cpp
class Solution {
public:
    void getNumber(int d, int ind, vector<int>& buff, vector<int>& arr) {
        if (d == 0) {
            vector<int> temp;
            for (auto x : buff) for (int i = 0; i < x; i++) temp.push_back(x);
            do {
                int num = 0;
                for (auto x : temp) num = num * 10 + x;
                arr.push_back(num);
            } while (next_permutation(temp.begin(), temp.end()));
            return ;
        }
        for (int i = ind; i <= d; i++) {
            if (d - i > i || i == d) {
                buff.push_back(i);
                getNumber(d - i, i + 1, buff, arr);
                buff.pop_back();
            }
        }
        return ;
    }
    void getAllNumber(int d, vector<int>& arr) {
        vector<int> buff;
        getNumber(d, 1, buff, arr);
        return ;
    }
    int nextBeautifulNumber(int n) {
        if (n == 0) return 1;
        int d = floor(log10(n)) + 1;
        vector<int> arr;
```

```
        getAllNumber(d, arr);
        getAllNumber(d + 1, arr);
        int ans = INT_MAX;
        for (auto x : arr) {
            if (x > n) ans = min(ans, x);
        }
        return ans;
    }
};
```

352. 将数据流变为多个不相交区间

给你一个由非负整数 `a1, a2, ..., an` 组成的数据流输入，请你将到目前为止看到的数字总结为不相交的区间列表。

实现 `SummaryRanges` 类：

- `SummaryRanges()` 使用一个空数据流初始化对象。
- `void addNum(int val)` 向数据流中加入整数 `val` 。
- `int[][] getIntervals()` 以不相交区间 `[starti, endi]` 的列表形式返回对数据流中整数的总结。

示例：

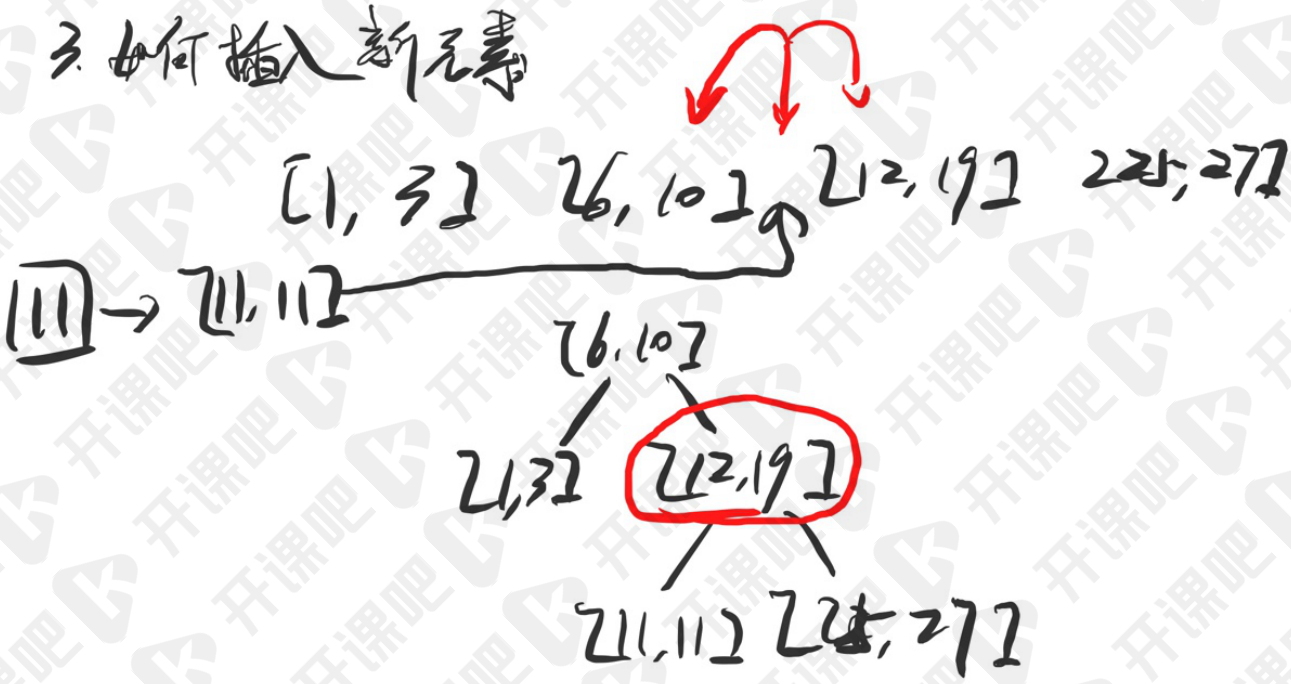
```
输入：
["SummaryRanges", "addNum", "getIntervals", "addNum", "getIntervals", "addNum", "getIntervals", "addNum", "getIntervals", "addNum", "getIntervals"]
[[], [1], [], [3], [], [7], [], [2], [], [6], []]
输出：
[null, null, [[1, 1]], null, [[1, 1], [3, 3]], null, [[1, 1], [3, 3], [7, 7]], null, [[1, 3], [7, 7]], null, [[1, 3], [6, 7]]]

解释：
SummaryRanges summaryRanges = new SummaryRanges();
summaryRanges.addNum(1);      // arr = [1]
summaryRanges.getIntervals(); // 返回 [[1, 1]]
summaryRanges.addNum(3);      // arr = [1, 3]
summaryRanges.getIntervals(); // 返回 [[1, 1], [3, 3]]
summaryRanges.addNum(7);      // arr = [1, 3, 7]
summaryRanges.getIntervals(); // 返回 [[1, 1], [3, 3], [7, 7]]
summaryRanges.addNum(2);      // arr = [1, 2, 3, 7]
summaryRanges.getIntervals(); // 返回 [[1, 3], [7, 7]]
summaryRanges.addNum(6);      // arr = [1, 2, 3, 6, 7]
summaryRanges.getIntervals(); // 返回 [[1, 3], [6, 7]]
```

1. 如何表示区间 → pair<int, int>
   - 第一个 → first
   - 第二个 → second

2. 如何维护区间之间的动态顺序
   - 红黑树 → map/set

3. 如何插入新元素




```
class SummaryRanges {
public:
    typedef pair<int, int> PII;
    set<PII> s;
    unordered_set<int> uniq;
    SummaryRanges() {}
```

```
    PII mergePre(PII d) {
        auto pre = s.find(d);
        auto now = (pre--);
        if (pre == s.end() || pre->second + 1 != now->first) return d;
        auto pre_d = *pre;
        s.erase(pre);
        s.erase(s.find(d));
        d.first = pre_d.first;
        s.insert(d);
        return d;
    }

    PII mergeNext(PII d) {
        auto next = s.find(d);
        auto now = (next++);
        if (next == s.end() || next->first - 1 != now->second) return d;
        auto next_d = *next;
        s.erase(next);
        s.erase(s.find(d));
        d.second = next_d.second;
        s.insert(d);
        return d;
    }

    void addNum(int val) {
        if (uniq.find(val) != uniq.end()) return ;
        uniq.insert(val);
        PII d(val, val);
        s.insert(d);
        d = mergePre(d);
        d = mergeNext(d);
        return ;
    }

    vector<vector<int>> getIntervals() {
        vector<vector<int>> ret;
        vector<int> range(2);
        for (auto x : s) {
            range[0] = x.first;
            range[1] = x.second;
            ret.push_back(range);
        }
        return ret;
    }
};

/**
 * Your SummaryRanges object will be instantiated and called as such:
```

```
 * SummaryRanges* obj = new SummaryRanges();
 * obj->addNum(val);
 * vector<vector<int>> param_2 = obj->getIntervals();
 */
```

740. 删除并获得点数

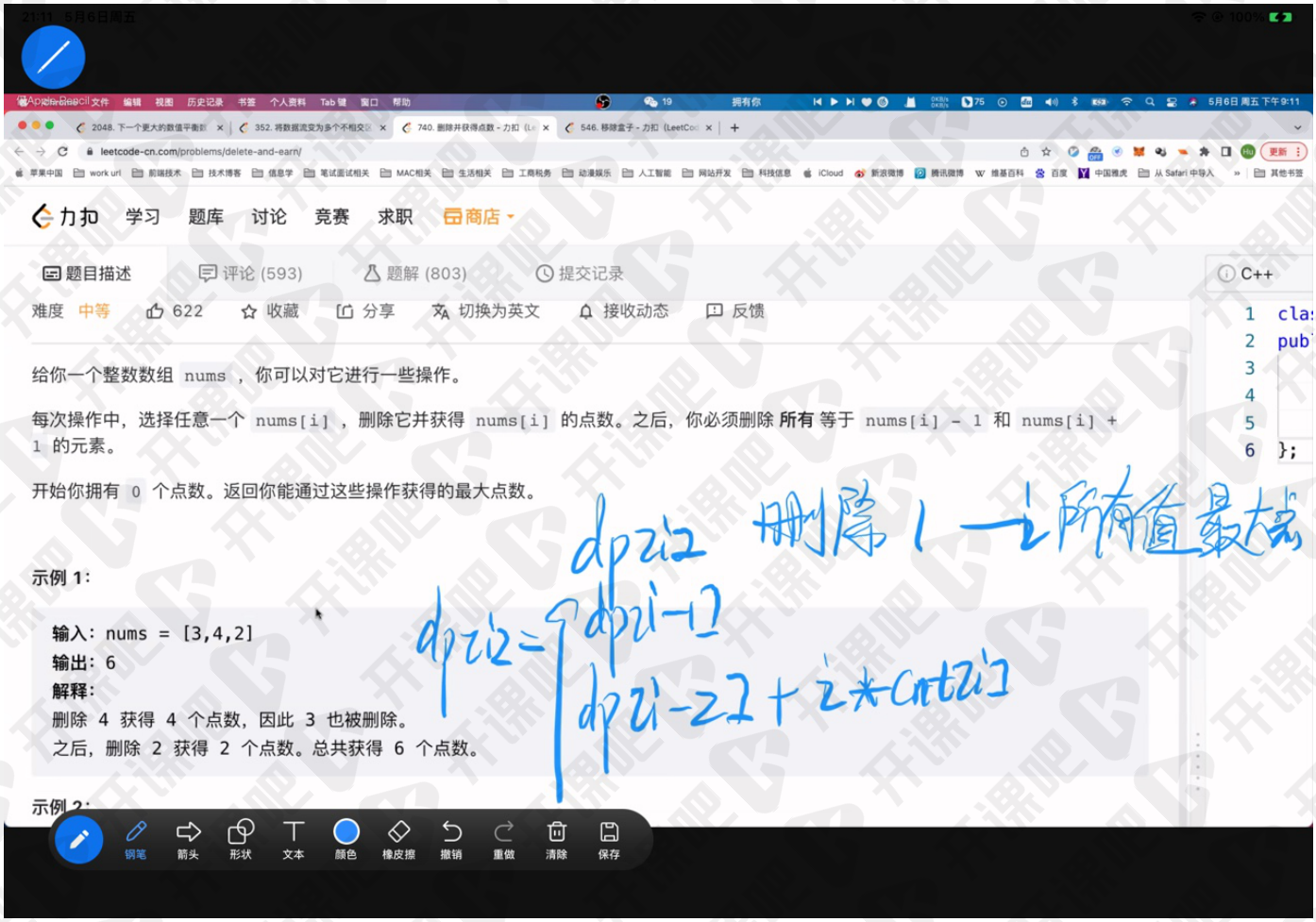
给你一个整数数组 `nums` ，你可以对它进行一些操作。

每次操作中，选择任意一个 `nums[i]` ，删除它并获得 `nums[i]` 的点数。之后，你必须删除 **所有** 等于 `nums[i] - 1` 和 `nums[i] + 1` 的元素。

开始你拥有 `0` 个点数。返回你能通过这些操作获得的最大点数。

示例 1：

```
输入：nums = [3,4,2]
输出：6
解释：
删除 4 获得 4 个点数，因此 3 也被删除。
之后，删除 2 获得 2 个点数。总共获得 6 个点数。
```



```cpp
class Solution {
public:
    int deleteAndEarn(vector<int>& nums) {
        int cnt[10005] = {0}, dp[10005] = {0};
        for (auto x : nums) cnt[x] += 1;
        dp[1] = cnt[1];
        for (int i = 2; i <= 10000; i++) {
            dp[i] = max(dp[i - 1], dp[i - 2] + i * cnt[i]);
        }
        return dp[10000];
    }
};
```

546. 移除盒子

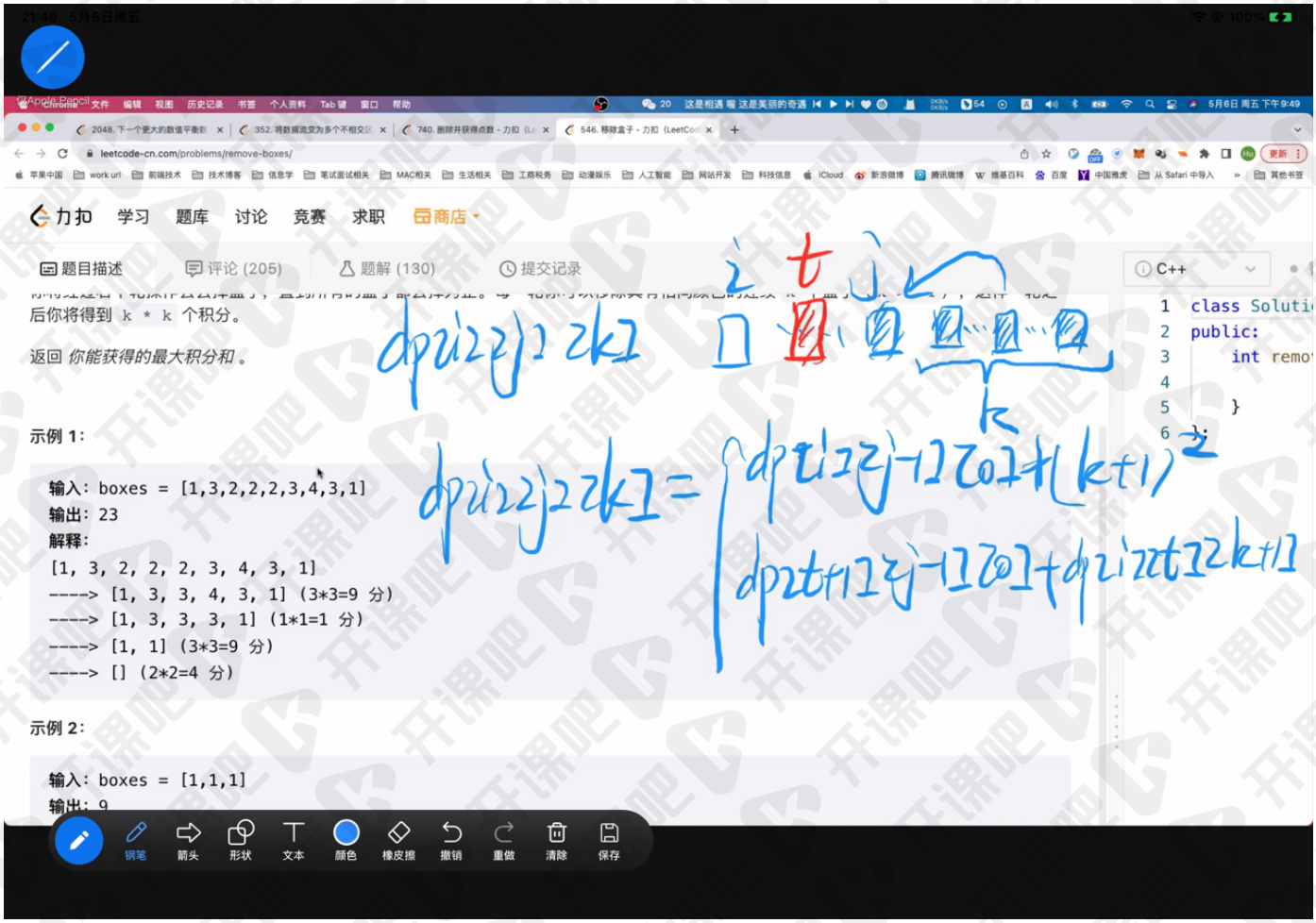
给出一些不同颜色的盒子 `boxes` ，盒子的颜色由不同的正数表示。

你将经过若干轮操作去去掉盒子，直到所有的盒子都去掉为止。每一轮你可以移除具有相同颜色的连续 `k` 个盒子（`k >= 1`），这样一轮之后你将得到 `k * k` 个积分。

返回 *你能获得的最大积分和* 。

示例 1：

```
输入：boxes = [1,3,2,2,2,3,4,3,1]
输出：23
解释：
[1, 3, 2, 2, 2, 3, 4, 3, 1]
----> [1, 3, 3, 4, 3, 1] (3*3=9 分)
----> [1, 3, 3, 3, 1] (1*1=1 分)
----> [1, 1] (3*3=9 分)
----> [] (2*2=4 分)
```




```cpp
class Solution {
public:
    int removeBoxes(vector<int>& boxes) {
        int dp[105][105][105] = {0}, n = boxes.size();
        for (int l = 1; l <= n; l++) {
            for (int i = 1; i + l - 1 <= n; i++) {
                int j = i + l - 1;
                for (int k = 0; k <= n; k++) {
                    dp[i][j][k] = dp[i][j - 1][0] + (k + 1) * (k + 1);
                    for (int t = i; t < j; t++) {
                        if (boxes[t - 1] != boxes[j - 1]) continue;
                        dp[i][j][k] = max(dp[i][j][k], dp[t + 1][j - 1][0] + dp[i][t][k + 1]);
                    }
                }
            }
        }
        return dp[1][n][0];
    }
};
```

10. 正则表达式匹配

给你一个字符串 `s` 和一个字符规律 `p`，请你来实现一个支持 `'.'` 和 `'*'` 的正则表达式匹配。

- `'.'` 匹配任意单个字符
- `'*'` 匹配零个或多个前面的那一个元素

所谓匹配，是要涵盖 **整个** 字符串 `s` 的，而不是部分字符串。

示例 1：

```
输入：s = "aa", p = "a"
输出：false
解释："a" 无法匹配 "aa" 整个字符串。
```

```cpp
class Solution {
public:
    bool __isMatch(const char *s, const char *p) {
        if (p[0] == 0) return s[0] == 0;
        if (s[0] == 0) {
            if (p[1] == '*') return __isMatch(s, p + 2);
            else return false;
        }
        bool first_match = s[0] && (s[0] == p[0] || p[0] == '.');
        if (p[1] == '*') {
            return __isMatch(s, p + 2) || (first_match && __isMatch(s + 1, p));
        }
        return first_match && __isMatch(s + 1, p + 1);
    }
    bool isMatch(string s, string p) {
        return __isMatch(s.c_str(), p.c_str());
    }
};
```

1316. 不同的循环子字符串

给你一个字符串 `text` ，请你返回满足下述条件的 **不同** 非空子字符串的数目：

- 可以写成某个字符串与其自身相连接的形式（即，可以写为 `a + a`，其中 `a` 是某个字符串）。

例如，`abcabc` 就是 `abc` 和它自身连接形成的。

示例 1：

```
输入：text = "abcabcabc"
输出：3
解释：3 个子字符串分别为 "abcabc"，"bcabca" 和 "cabcab" 。
```

S:

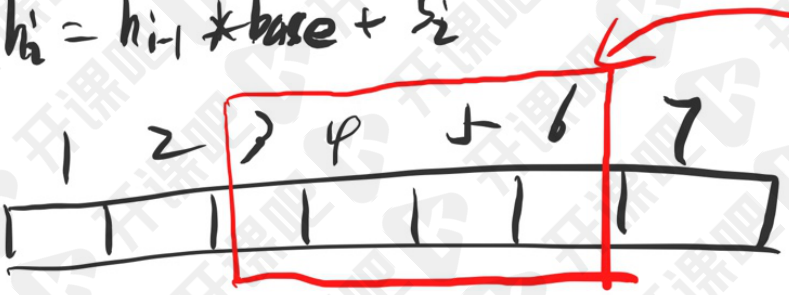
h_i：$1 \longleftrightarrow i$ 的字符串哈希值

$h_i = h_{i-1} * base + s_i$

↓

s_1

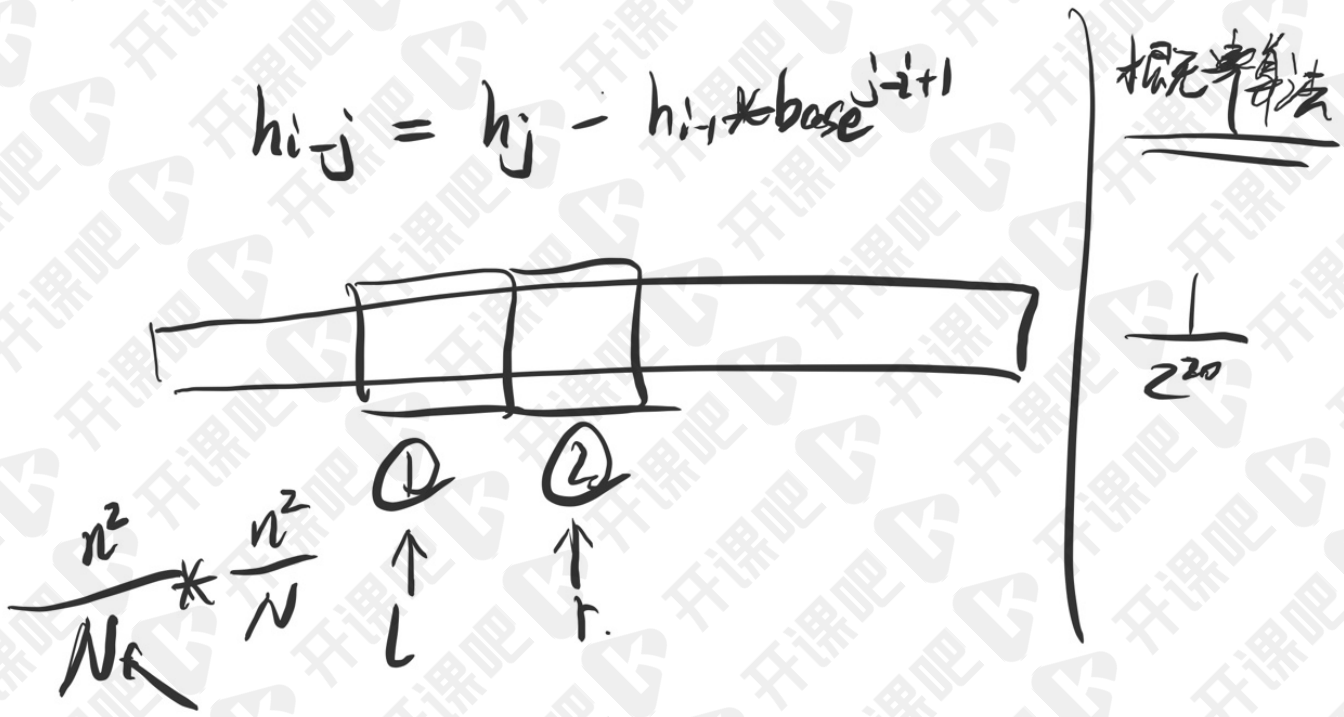
$h_i = h_{i-1} * base + s_i$



$h_6 = s_6 + s_5 * base + s_4 * base^2 + s_3 * base^3$

$+ \boxed{s_2 * base^4 + s_1 * base^5} = h_2 * base^4$

$h_2 = s_2 + s_1 * base$




```cpp
class Solution {
public:
    int distinctEchoSubstrings(string s) {
        long long base = 31, MOD = (1e9 + 7), h[2005] = {0}, mul[2005] = {0};
        mul[0] = 1;
        for (int i = 1, n = s.size(); i <= n; i++) {
            h[i] = (h[i - 1] * base + s[i - 1]) % MOD;
            mul[i] = mul[i - 1] * base % MOD;
        }
        int ans = 0;
        unordered_set<long long> uniq;
        for (int l = 1, n = s.size(); l * 2 <= n; l++) {
            for (int i = 1; i + 2 * l - 1 <= n; i++) {
                int h1 = (h[i + l - 1] - h[i - 1] * mul[l] % MOD + MOD) % MOD;
                int h2 = (h[i + 2 * l - 1] - h[i + l - 1] * mul[l] % MOD + MOD) % MOD;
                if (uniq.find(h1) != uniq.end() || h1 != h2) continue;
                ans += 1;
                uniq.insert(h1);
            }
        }
        return ans;
    }
};
```

1980. 找出不同的二进制字符串

给你一个字符串数组 `nums` ，该数组由 `n` 个 **互不相同** 的二进制字符串组成，且每个字符串长度都是 `n` 。请你找出并返回一个长度为 `n` 且 **没有出现** 在 `nums` 中的二进制字符串。如果存在多种答案，只需返回 **任意一个** 即可。

示例 1：

```
输入：nums = ["01","10"]
输出："11"
解释："11" 没有出现在 nums 中。"00" 也是正确答案。
```

```cpp
class Solution {
public:
    string findDifferentBinaryString(vector<string>& nums) {
        string t = "";
        for (int i = 0, n = nums.size(); i < n; i++) {
            t += nums[i][i] == '0' ? '1' : '0';
        }
        return t;
    }
};
```