本日学习内容

1. 知识分享
2. 复习简单的dp问题，完成3道状态机dp
3. 复习栈与队列，使用STL简化代码

本日分享内容

题目一：740. 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。
```

示例 2：

```
输入：nums = [2,2,3,3,3,4]
输出：9
解释：
删除 3 获得 3 个点数，接着要删除两个 2 和 4 。
之后，再次删除 3 获得 3 个点数，再次删除 3 获得 3 个点数。
总共获得 9 个点数。
```

提示：

- `1 <= nums.length <= 2 * 104`
- `1 <= nums[i] <= 104`

思路

这个问题可以转化为类似**打家劫舍**（House Robber）的问题，即不能选择相邻的数字。具体步骤如下：

1. **统计数字频率**：统计每个数字出现的次数，并计算每个数字的总点数（数字 × 出现次数）。

2. 动态规划

   使用动态规划来计算最大点数，类似于打家劫舍问题：

   - 如果选择当前数字 `i`，则不能选择 `i-1`，最大点数为 `dp[i-2] + points[i]`。
   - 如果不选择当前数字 `i`，则最大点数为 `dp[i-1]`。
   - 取两者中的较大值作为 `dp[i]`。

代码

```cpp
class Solution {
public:
    int deleteAndEarn(vector<int>& nums) {
        int n = nums.size();
        if (nums.empty()) {
            return 0;
        }
        int maxx = *max_element(nums.begin(), nums.end());
        vector<int> sum(maxx + 1, 0);
        for (int num : nums) {
            sum[num] += num;
        }
        vector<int> dp(maxx + 1, 0);
        dp[0] = sum[0];
        dp[1] = sum[1];
        for (int i = 2; i <= maxx; i++) {
            dp[i] = max(dp[i - 1], dp[i - 2] + sum[i]);
        }
        return dp[maxx];
    }
};
```

题目二：2320. 统计放置房子的方式数

一条街道上共有 `n * 2` 个 **地块** ，街道的两侧各有 `n` 个地块。每一边的地块都按从 `1` 到 `n` 编号。每个地块上都可以放置一所房子。

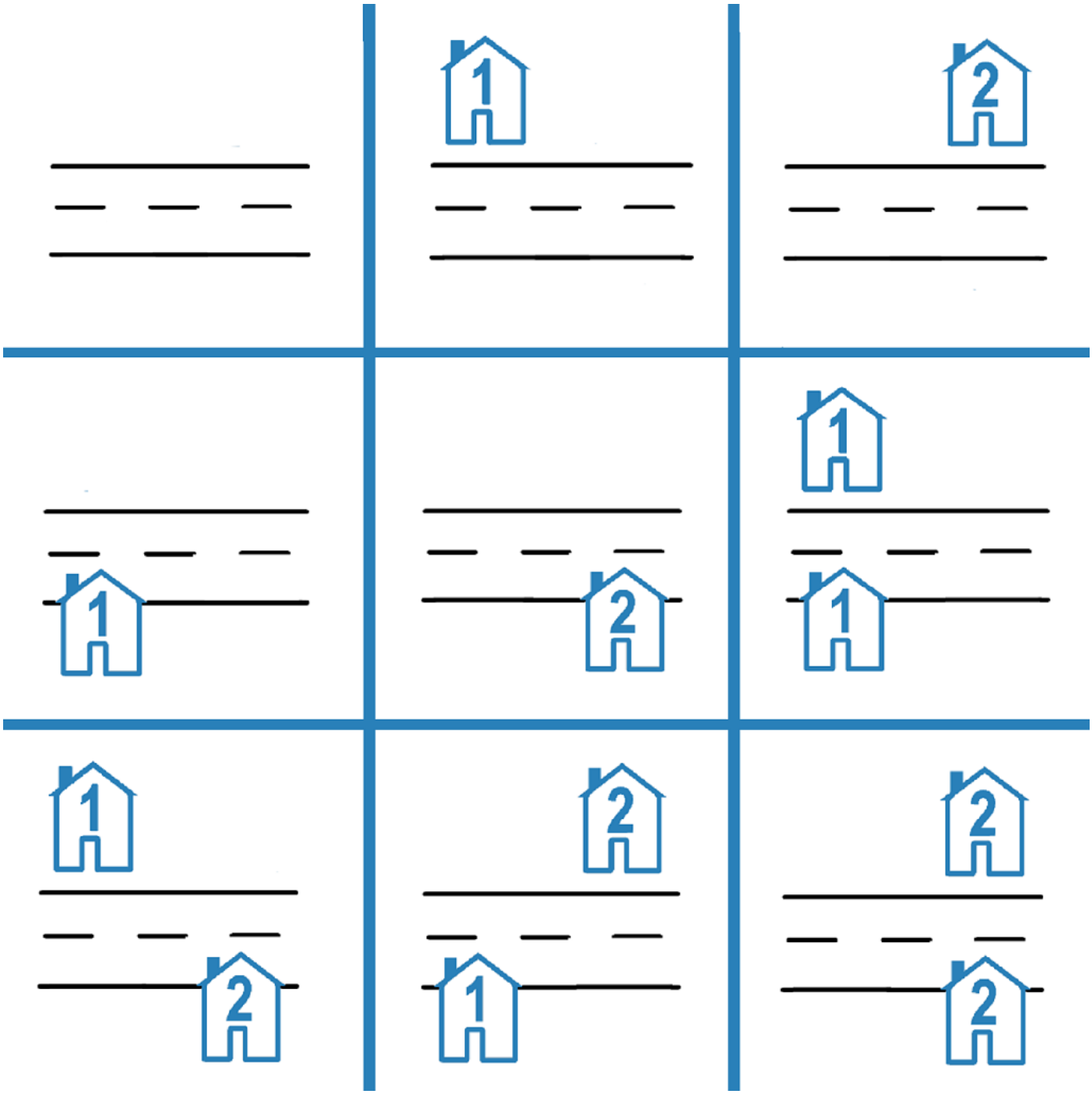
现要求街道同一侧不能存在两所房子相邻的情况，请你计算并返回放置房屋的方式数目。由于答案可能很大，需要对 `109 + 7` 取余后再返回。

注意，如果一所房子放置在这条街某一侧上的第 `i` 个地块，不影响在另一侧的第 `i` 个地块放置房子。

示例 1：

```
输入：n = 1
输出：4
解释：
可能的放置方式：
1．所有地块都不放置房子。
2．一所房子放在街道的某一侧。
3．一所房子放在街道的另一侧。
4．放置两所房子，街道两侧各放置一所。
```

示例 2：



```
输入：n = 2
输出：9
解释：如上图所示，共有 9 种可能的放置方式。
```

提示：

- `1 <= n <= 104`

思路

这个问题类似于**斐波那契数列**问题，因为放置房子的方式数满足递推关系：

- 对于第i块地，有两种选择：
  - 不放置房子：则前 `i-1` 块地的放置方式数为 `dp[i-1]`。
  - 放置房子：则第 `i-1` 块地不能放置房子，前 `i-2` 块地的放置方式数为 `dp[i-2]`。
- 因此，递推公式为 `dp[i] = dp[i-1] + dp[i-2]`。

代码

```cpp
class Solution {
public:
    int countHousePlacements(int n) {
        const int MOD = 1e9 + 7;
        vector<long long> dp(n + 1, 0);
        dp[0] = 1;
        dp[1] = 2;
        for (int i = 2; i <= n; i++) {
            dp[i] = (dp[i - 1] + dp[i - 2]) % MOD;
        }
        long long res = (dp[n] * dp[n]) % MOD;
        return res;
    }
};
```

题目三：20. 有效的括号

给定一个只包括 `'('`，`')'`，`'{'`，`'}'`，`'['`，`']'` 的字符串 `s`，判断字符串是否有效。

有效字符串需满足：

1. 左括号必须用相同类型的右括号闭合。

2. 左括号必须以正确的顺序闭合。
3. 每个右括号都有一个对应的相同类型的左括号。

示例 1：

输入：s = "()"

输出：true

示例 2：

输入：s = "()[]{}"

输出：true

示例 3：

输入：s = "(]"

输出：false

示例 4：

输入：s = "([])"

输出：true

提示：

- `1 <= s.length <= 104`
- `s` 仅由括号 `'()[]{}'` 组成

思路：

1. 栈处理：

   遇到左括号时，压入对应的右括号。

   遇到右括号时，检查栈顶是否匹配：

   栈为空或不匹配，返回 `false`。

   匹配则弹出栈顶元素。
2. **最终检查**：遍历结束后，栈为空则返回 `true`，否则返回 `false`。

代码：

```cpp
class Solution {
public:
    bool isValid(string s) {
        int n = s.size();
        if (s.size() % 2 != 0) {
            return false;
        }
        stack<char> stk;
        for (char c : s) {
            if (c == '(') {
                stk.push(')');
            } else if (c == '[') {
                stk.push(']');
            } else if (c == '{') {
                stk.push('}');
            } else if (stk.empty() || stk.top() != c) {
                return false;
            } else {
                stk.pop();
            }
        }
        return stk.empty();
    }
};
```

本日遇到的问题

1. 之前写过的题不能很快反应出来

明日学习内容

1. 开始复习算法：哈希，双指针，kmp字符串