# 题目

给你一个整数数组 nums，请你找出并返回能被三整除的元素最大和。

示例 1：

输入：nums = [3,6,5,1,8]

输出：18

解释：选出数字 3, 6, 1 和 8，它们的和是 18（可被 3 整除的最大和）。

示例 2：

输入：nums = [4]

输出：0

解释：4 不能被 3 整除，所以无法选出数字，返回 0。

示例 3：

输入：nums = [1,2,3,4,4]

输出：12

解释：选出数字 1, 3, 4 以及 4，它们的和是 12（可被 3 整除的最大和）。

提示：

1 <= nums.length <= 4 \* 10^4

1 <= nums[i] <= 10^4

# 分析

## 方法一：动态规划

思路：

使用动态规划来解决问题。具体思路如下：

1、创建一个二维数组dp，其中dp[i][j]表示考虑前i个元素时，余数为j的最大和。

2、初始化dp[0][0] = 0，dp[0][1] = INT\_MIN，dp[0][2] = INT\_MIN，表示没有任何元素时的情况。

3、遍历数组nums，对于每个元素num：

- 如果num除以3的余数为0，则更新dp[i][0]、dp[i][1]和 dp[i][2]，分别取当前值和加上num的较大值。

- 如果num除以3的余数为1，则更新dp[i][0]、dp[i][1]和dp[i][2]，分别取当前值和加上num的较大值。

- 如果num除以3的余数为2，则更新dp[i][0]、dp[i][1]和dp[i][2]，分别取当前值和加上num的较大值。

4、最后返回dp[n][0]，其中n是nums的长度，表示考虑所有元素时余数为0的最大和，即为结果。

代码：

class Solution {

public:

int maxSumDivThree(vector<int>& nums) {

int n = nums.size();

vector<vector<int>> dp(n + 1, vector<int>(3, 0));

dp[0][0] = 0;

dp[0][1] = INT\_MIN;

dp[0][2] = INT\_MIN;

for (int i = 1; i <= n; ++i) {

if (nums[i - 1] % 3 == 0) {

dp[i][0] = max(dp[i - 1][0], dp[i - 1][0] + nums[i - 1]);

dp[i][1] = max(dp[i - 1][1], dp[i - 1][1] + nums[i - 1]);

dp[i][2] = max(dp[i - 1][2], dp[i - 1][2] + nums[i - 1]);

} else if (nums[i - 1] % 3 == 1) {

dp[i][0] = max(dp[i - 1][0], dp[i - 1][2] + nums[i - 1]);

dp[i][1] = max(dp[i - 1][1], dp[i - 1][0] + nums[i - 1]);

dp[i][2] = max(dp[i - 1][2], dp[i - 1][1] + nums[i - 1]);

} else if (nums[i - 1] % 3 == 2) {

dp[i][0] = max(dp[i - 1][0], dp[i - 1][1] + nums[i - 1]);

dp[i][1] = max(dp[i - 1][1], dp[i - 1][2] + nums[i - 1]);

dp[i][2] = max(dp[i - 1][2], dp[i - 1][0] + nums[i - 1]);

}

}

return dp[n][0];

}

};