# 题目

你将得到一个整数数组 matchsticks ，其中 matchsticks[i] 是第 i 个火柴棒的长度。你要用 所有的火柴棍 拼成一个正方形。你 不能折断 任何一根火柴棒，但你可以把它们连在一起，而且每根火柴棒必须 使用一次 。

如果你能使这个正方形，则返回 true ，否则返回 false 。

示例 1:

矩形

中度可信度描述已自动生成

输入: matchsticks = [1,1,2,2,2]

输出: true

解释: 能拼成一个边长为2的正方形，每边两根火柴。

示例 2:

输入: matchsticks = [3,3,3,3,4]

输出: false

解释: 不能用所有火柴拼成一个正方形。

提示:

1 <= matchsticks.length <= 15

1 <= matchsticks[i] <= 108

# 分析

## 方法一：回溯法

要解决“用所有火柴棒拼成正方形”的问题，核心思路是回溯法：先验证总长度是否能被4整除（否则直接返回false），再尝试将火柴棒分配到4条边中，确保每条边的长度最终等于目标边长（总长度/4）。

解题思路

1、前置验证：

- 若火柴棒总数少于4，直接返回false（无法构成正方形的4条边）。

- 计算火柴棒总长度，若总长度不能被4整除，返回false（无法均分4条边）。

- 目标边长target = 总长度 / 4，若存在火柴棒长度超过target，返回false（单根火柴无法拆分，无法放入任何一条边）。

2、回溯分配策略：

- 用数组edges存储4条边当前的长度（初始均为0）。

- 按从长到短排序火柴棒（优先分配长火柴，减少无效分支：长火柴若无法放入当前边，后续更短的边也无需尝试，提前剪枝）。

- 递归尝试将每根火柴放入4条边中的一条：

1）若当前边放入火柴后长度超过target，跳过该边。

2）若当前边与前一条边长度相同，跳过该边（避免重复尝试相同状态，例如两条边当前均为0，放入同一根火柴的效果一致，无需重复计算）。

3）放入火柴后，递归处理下一根火柴。

4）若递归成功（所有火柴分配完毕），返回true；否则回溯（移除当前边的火柴，尝试下一条边）。

3、终止条件：

- 所有火柴分配完毕（递归到最后一根火柴且成功放入），返回true。

代码：

class Solution {

public:

bool makesquare(vector<int>& matchsticks) {

// 前置验证1：火柴数量不足4，无法构成正方形

if (matchsticks.size() < 4) {

return false;

}

// 计算总长度，前置验证2：总长度不能被4整除

int total = 0;

for (int len : matchsticks) {

total += len;

}

if (total % 4 != 0) {

return false;

}

int target = total / 4; // 目标边长

// 前置验证3：存在火柴长度超过目标边长，无法放入任何边

for (int len : matchsticks) {

if (len > target) {

return false;

}

}

// 优化：按从长到短排序，优先分配长火柴，减少无效分支

sort(matchsticks.begin(), matchsticks.end(), greater<int>());

vector<int> edges(4, 0); // 存储4条边当前的长度

// 回溯分配火柴

return backtrack(matchsticks, edges, 0, target);

}

private:

// 回溯函数：尝试分配第index根火柴到4条边中

// matchsticks：火柴棒数组（已排序），edges：当前4条边的长度，index：当前处理的火柴索引，target：目标边长

bool backtrack(vector<int>& matchsticks, vector<int>& edges, int index, int target) {

// 终止条件：所有火柴分配完毕，返回true

if (index == matchsticks.size()) {

return true;

}

// 尝试将当前火柴放入4条边中的一条

for (int i = 0; i < 4; ++i) {

// 剪枝1：当前边放入火柴后超过目标边长，跳过

if (edges[i] + matchsticks[index] > target) {

continue;

}

// 剪枝2：当前边与前一条边长度相同，跳过（避免重复尝试相同状态）

if (i > 0 && edges[i] == edges[i - 1]) {

continue;

}

// 放入火柴

edges[i] += matchsticks[index];

// 递归处理下一根火柴，若成功则返回true

if (backtrack(matchsticks, edges, index + 1, target)) {

return true;

}

// 回溯：移除当前边的火柴，尝试下一条边

edges[i] -= matchsticks[index];

}

// 4条边均无法放入当前火柴，返回false

return false;

}

};

代码解释

1、前置验证：快速排除不可能的情况（数量不足、总长度不可均分、单根火柴过长），减少后续回溯的无效计算。

2、排序优化：从长到短排序火柴棒，优先处理长火柴：

- 长火柴的可选边更少（若超过当前边剩余长度，直接跳过），能更早发现无效分支，大幅减少递归次数。

3、回溯剪枝：

- 剪枝1：避免边长度超过目标，直接跳过无效边。

- 剪枝2：避免重复处理相同状态（如两条边当前长度相同，放入同一根火柴的效果一致，无需重复尝试），进一步减少分支。

4、回溯逻辑：通过递归尝试所有可能的分配方式，成功则返回`true`，失败则回溯，直到遍历所有可能性。

复杂度分析

- 时间复杂度：O(4^n)，其中n是火柴棒数量。理论上每根火柴有4条边可选，共4^n种分配方式，但通过排序和剪枝（如长火柴优先、重复状态跳过），实际运行效率会大幅提升（尤其n≤15时可高效处理）。

- 空间复杂度：O(n)。递归栈深度为n（处理每根火柴），edges数组大小固定为4（常数级），整体空间由递归栈主导。