# 题目

如果序列 X\_1, X\_2, ..., X\_n 满足下列条件，就说它是 斐波那契式 的：

* n >= 3
* 对于所有 i + 2 <= n，都有 X\_i + X\_{i+1} = X\_{i+2}

给定一个严格递增的正整数数组形成序列arr ，找到arr中最长的斐波那契式的子序列的长度。如果一个不存在，返回0 。

（回想一下，子序列是从原序列 arr 中派生出来的，它从 arr 中删掉任意数量的元素（也可以不删），而不改变其余元素的顺序。例如， [3, 5, 8] 是 [3, 4, 5, 6, 7, 8] 的一个子序列）

示例 1：

输入: arr = [1,2,3,4,5,6,7,8]

输出: 5

解释: 最长的斐波那契式子序列为 [1,2,3,5,8] 。

示例 2：

输入: arr = [1,3,7,11,12,14,18]

输出: 3

解释: 最长的斐波那契式子序列有 [1,11,12]、[3,11,14] 以及 [7,11,18] 。

提示：

3 <= arr.length <= 1000

1 <= arr[i] < arr[i + 1] <= 10^9

# 分析

## 方法一：动态规划

思路：

这个问题可以使用动态规划来解决。我们可以定义状态dp[i][j]表示以arr[i]和arr[j]结尾的斐波那契式子序列的最大长度。状态转移方程如下：

if (arr[k] == arr[i] + arr[j]) {

dp[i][j] = dp[j][k] + 1;

}

其中k < j是满足arr[k] == arr[i] + arr[j]的最大k值。

最终的答案就是dp[i][j]中的最大值加上 2（因为每个斐波那契式子序列至少包含三个元素）。

代码：

class Solution {

public:

int lenLongestFibSubseq(vector<int>& arr) {

int n = arr.size();

unordered\_map<int, int> index; // 用于快速查找元素在数组中的位置

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

index[arr[i]] = i;

}

vector<vector<int>> dp(n, vector<int>(n, 2)); // 初始化为2，因为每个斐波那契式子序列至少包含三个元素

int longest = 0;

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

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

int target = arr[j] - arr[i];

if (target < arr[i] && index.find(target) != index.end()) {

int k = index[target];

dp[i][j] = max(dp[i][j], dp[k][i] + 1);

longest = max(longest, dp[i][j]);

}

}

}

return longest > 2 ? longest : 0;

}

};

这个实现中，我们首先用index哈希表记录每个元素在数组中的位置，然后定义一个二维动态规划数组dp，用于记录以每对元素结尾的斐波那契式子序列的最大长度。接着，我们遍历数组arr中的每一对元素 (arr[i], arr[j])，并根据状态转移方程更新dp数组。最终，我们返回dp数组中的最大值加上2，作为最长斐波那契式子序列的长度。

## 方法二：双指针

思路：

我们可以考虑使用双指针来解决这个问题。具体思路如下：

1、我们可以固定两个数arr[i]和arr[j]，然后尝试找到以arr[i]和arr[j]结尾的斐波那契式子序列。

2、我们使用两个指针left和right，分别指向arr[j]和arr[j] + arr[i]在数组中的位置，然后不断向右移动这两个指针。

3、如果arr[left] + arr[right] == arr[j] + arr[i]，说明找到了一个斐波那契式子序列，更新最大长度，并将left和right分别向右移动一位。

4、如果arr[left] + arr[right] < arr[j] + arr[i]，说明当前和太小，应该增大和，因此将right向右移动一位。

5、如果arr[left] + arr[right] > arr[j] + arr[i]，说明当前和太大，应该减小和，因此将left向右移动一位。

代码：

class Solution {

public:

int lenLongestFibSubseq(vector<int>& arr) {

int n = arr.size();

unordered\_set<int> s(arr.begin(), arr.end());

int longest = 0;

for (int i = 0; i < n - 1; i++) {

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

int a = arr[i], b = arr[j], l = 2;

while (s.find(a + b) != s.end()) {

b = a + b;

a = b - a;

l++;

}

longest = max(longest, l);

}

}

return longest > 2 ? longest : 0;

}

};

这个实现中，我们遍历数组中的每一对元素 (arr[i], arr[j])，然后尝试找到以这两个元素为起始的斐波那契式子序列。通过不断增加第三个元素的值，并检查是否在数组中，来确定是否存在这样的子序列。最终返回最长的斐波那契式子序列的长度。