# 题目

数组的每个索引作为一个阶梯，第 i个阶梯对应着一个非负数的体力花费值 cost[i](索引从0开始)。

每当你爬上一个阶梯你都要花费对应的体力花费值，然后你可以选择继续爬一个阶梯或者爬两个阶梯。

您需要找到达到楼层顶部的最低花费。在开始时，你可以选择从索引为 0 或 1 的元素作为初始阶梯。

**示例 1:**

输入: cost = [10, 15, 20]

输出: 15

解释: 最低花费是从cost[1]开始，然后走两步即可到阶梯顶，一共花费15。

**示例 2:**

输入: cost = [1, 100, 1, 1, 1, 100, 1, 1, 100, 1]

输出: 6

解释: 最低花费方式是从cost[0]开始，逐个经过那些1，跳过cost[3]，一共花费6。

**注意：**

cost 的长度将会在 [2, 1000]。

每一个 cost[i] 将会是一个Integer类型，范围为 [0, 999]。

# 分析

## 方法一：动态规划

**思路：**

计算花费f[i]有一个清楚的递归关系：f[i] = cost[i] + min(f[i+1], f[i+2])。我们可以使用动态规划来实现。

**算法：**

当我们要计算f[i]时，要先计算出f[i+1]和f[i+2]。所以我们应该从后往前计算f。

在第i步，让f1，f2为f[i+1]，f[i+2]的旧值，并将其更新为f[i]，f[i+1]的新值。当我们从后遍历i时，我们会保持这些更新。在最后答案是min(f1, f2)。

**代码：**

class Solution {

public int minCostClimbingStairs(vector<int>& cost) {

int f1 = 0, f2 = 0;

for (int i = cost.size() - 1; i >= 0; --i) {

int f0 = cost[i] + min(f1, f2);

f2 = f1;

f1 = f0;

}

return min(f1, f2);

}

}

复杂度分析：

时间复杂度：O(N)。N指的是cost的长度

空间复杂度：O(1)，只使用了f1, f2。

另一种写法（推荐）：

class Solution {

public:

int minCostClimbingStairs(vector<int>& cost) {

int n = cost.size();

vector<int> dp(n + 1);

dp[0] = dp[1] = 0;

for (int i = 2; i <= n; i++) {

dp[i] = min(dp[i - 1] + cost[i - 1], dp[i - 2] + cost[i - 2]);

}

return dp[n];

}

};

**复杂度分析：**

时间复杂度和空间复杂度都是O(n)