# 题目

给定一个三角形 triangle ，找出自顶向下的最小路径和。

每一步只能移动到下一行中相邻的结点上。相邻的结点 在这里指的是 下标 与 上一层结点下标 相同或者等于 上一层结点下标 + 1 的两个结点。也就是说，如果正位于当前行的下标 i ，那么下一步可以移动到下一行的下标 i 或 i + 1 。

示例 1：

输入：triangle = [[2],[3,4],[6,5,7],[4,1,8,3]]

输出：11

解释：如下面简图所示：

2

3 4

6 5 7

4 1 8 3

自顶向下的最小路径和为 11（即，2 + 3 + 5 + 1 = 11）。

示例 2：

输入：triangle = [[-10]]

输出：-10

提示：

1 <= triangle.length <= 200

triangle[0].length == 1

triangle[i].length == triangle[i - 1].length + 1

-10^4 <= triangle[i][j] <= 10^4

进阶：

你可以只使用 O(n) 的额外空间（n 为三角形的总行数）来解决这个问题吗？

注意：本题与主站 120 题相同：

<https://leetcode-cn.com/problems/triangle/>

# 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。我们可以从三角形的底部开始向上计算最小路径和。对于三角形中的每个位置 (i, j)，最小路径和可以选择从下一行的相邻两个位置 (i+1, j) 和 (i+1, j+1) 中选择较小的值，并将当前位置的值加上这个较小值。最终，顶部位置 (0, 0) 的值即为整个三角形的最小路径和。

在实现时，我们可以直接修改原始三角形数组，从倒数第二行开始向上更新每个位置的值，最终得到顶部位置的最小路径和。这样可以节省额外的空间。

动态规划的初始状态可以设定为最后一行的值，因为最后一行的最小路径和就是自身的值。然后，从倒数第二行开始，根据状态转移函数dp[i][j] = triangle[i][j] + min(dp[i+1][j], dp[i+1][j+1])来计算出每个位置的最小路径和。

在这里，我们可以列举一下dp[0]和dp[1]的值：

假设输入的三角形为triangle = [[2],[3,4],[6,5,7],[4,1,8,3]]。

1、初始化状态dp = [[0],[0,0],[0,0,0],[0,0,0,0]]。

2、计算最后一行的值，即dp[3][0] = 4，dp[3][1] = 1，dp[3][2] = 8，dp[3][3] = 3。

3、从倒数第二行开始向上更新，计算dp[2][0]、dp[2][1]、dp[2][2]：

- dp[2][0] = triangle[2][0] + min(dp[3][0], dp[3][1]) = 6 + min(4, 1) = 7

- dp[2][1] = triangle[2][1] + min(dp[3][1], dp[3][2]) = 5 + min(1, 8) = 6

- dp[2][2] = triangle[2][2] + min(dp[3][2], dp[3][3]) = 7 + min(8, 3) = 10

4、最终，dp[0][0]的值即为所求的最小路径和，为 11。

因此，dp[0][0] = 11。

使用这种方法，可以避免额外的空间开销，直接在原始数组上进行计算。

代码：

class Solution {

public:

int minimumTotal(vector<vector<int>>& triangle) {

int n = triangle.size();

vector<vector<int>> dp(n, vector<int>(n, 0));

// 初始化最后一行的最小路径和

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

dp[n - 1][i] = triangle[n - 1][i];

}

// 从倒数第二行开始向上计算最小路径和

for (int i = n - 2; i >= 0; --i) {

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

dp[i][j] = triangle[i][j] + min(dp[i + 1][j], dp[i + 1][j + 1]);

}

}

return dp[0][0];

}

};