# 题目

给定一个非负整数numRows，生成杨辉三角的前numRows行。

在杨辉三角中，每个数是它左上方和右上方的数的和。



<https://pic.leetcode-cn.com/1626927345-DZmfxB-PascalTriangleAnimated2.gif>

示例:

输入: 5

输出:

[

[1],

[1,1],

[1,2,1],

[1,3,3,1],

[1,4,6,4,1]

]

# 分析

## 方法一：动态规划

思路：

根据杨辉三角形的特点，符合动态规划的思想，当前的最优解基于前面的子问题的解。

两步：

1、先一个循环构造杨辉三角的边角值

2、再利用动态方程dp[i][j] = dp[i-1][j-1]+dp[i-1][j];

代码：

class Solution {

public:

vector<vector<int>> generate(int numRows) {

vector<vector<int>> ret(numRows);

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

ret[i].resize(i + 1); //预分配数组大小

ret[i][0] = ret[i][i] = 1; //初始值

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

ret[i][j] = ret[i - 1][j] + ret[i - 1][j - 1]; //状态转移方程

}

}

return ret;

}

};

**复杂度分析：**

时间复杂度：O(numRows2)。

空间复杂度：O(1)。不考虑返回值的空间占用。

或：

class Solution {

public:

vector<vector<int>> generate(int numRows) {

// 初始化数组大小

vector<vector<int>> ans(numRows);

// 边界条件判断

if (numRows < 0) return ans;

if (numRows == 1) return {{1}};

// 遍历每一行

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

ans[i].resize(i+1);

// 设置初始值（左右的值都是可以确定的初始值）

ans[i][0] = 1;

ans[i][i] = 1;

// 状态转移方程

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

ans[i][j] = ans[i-1][j-1] + ans[i-1][j];

}

}

return ans;

}

};

**另一种写法：**

class Solution {

public:

vector<vector<int>> generate(int numRows) {

vector<vector<int>> ret(numRows);

for(int i=0;i<numRows;i++)

{

ret[i] = vector<int>(i+1,0);

ret[i][0] = 1;//i行首尾设置为1

ret[i][i] = 1;

}

if(numRows<=2) return ret;

for(int i=2;i<numRows;i++)

{

for(int j=1;j<ret[i].size()-1;j++)

//前后两个位置都设置为1,不参与运算

{

ret[i][j] = ret[i-1][j-1] + ret[i-1][j];

}

}

return ret;

}

};