```
/*
给定一个非负索引 k，其中 k ≤ 33，返回杨辉三角的第 k 行。
在杨辉三角中，每个数是它左上方和右上方的数的和。

示例:

输入: 3
输出: [1,3,3,1]

进阶:

你可以优化你的算法到 O(k) 空间复杂度吗?

来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/pascals-triangle-ii
著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。
*/
```

分析:

- 根据二项式展开定理,第$n, n \geq 0$层杨辉三角系数有$n+1$个,且第i个系数的通用表达式为

$$C_i = C(n,i) = \frac{n!}{i!(n-i)!} \tag{1}$$

- 由于等式中的阶乘操作的复杂度为$O(n)$,且容易溢出,所以不能直接根据二项式定理直接展开做,否则复杂度为$O(n^2)$,且会有溢出错误.
- 找迭代关系

$$C_0 = \frac{n!}{0!(n-0)!} \tag{2}$$

$$C_1 = \frac{n!}{1!(n-1)!} = \frac{C_0 * n}{1} \tag{3}$$

$$C_2 = \frac{n!}{2!(n-2)!} = \frac{C_1 * (n-1)}{2} \tag{4}$$

$$\cdots \tag{5}$$

$$C_i = \frac{n!}{i!(n-i)!} = \frac{C_{i-1} * (n-i+1)}{i} \tag{6}$$

- 我们只要知道C_0的值,即可根据迭代公式进行逐一计算.

方法一:C++_迭代法

```
class Solution
{

    public:
        vector<int> getRow(int rowIndex)
        {
            int             i       = 0 ;
            vector<int>     ret_val     ;
            unsigned long   temp    = 1 ;

```

```
            ret_val.push_back(1);
            for(i = 1; i <= rowIndex ; i++)
            {
                temp = temp * (rowIndex-i+1);
                temp = temp / i ;
                ret_val.push_back(temp);
            }
            return ret_val;
        }
};

/*
执行结果:
通过
显示详情
执行用时 :0 ms, 在所有 C++ 提交中击败了100.00% 的用户
内存消耗 :8 MB, 在所有 C++ 提交中击败了99.84%的用户
*/
```

方法二:C_迭代法

```
/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
int* getRow(    int     rowIndex    ,
                int*    returnSize
           )
{
    int*            ret_val = NULL  ;
    unsigned long   temp    = 1     ;
    int             i       = 0     ;

    *returnSize = rowIndex+1;
    ret_val     = (int*)malloc(sizeof(int)*(rowIndex+1));
    ret_val[0]  = 1 ;
    for(i=1;i<=rowIndex;i++)
    {
        temp = temp *(rowIndex-i+1);
        temp = temp / i ;
        ret_val[i] = temp;
    }
    return ret_val;
}

/*
执行结果:
通过
显示详情
执行用时 :8 ms, 在所有 C 提交中击败了25.00% 的用户
内存消耗 :7.1 MB, 在所有 C 提交中击败了6.10%的用户
*/

/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
int* getRow(    int     rowIndex    ,
```

```c
                int*    returnSize
            )
{
    int*            ret_val = NULL  ;
    int             i       = 0     ;
    unsigned long   temp    = 1     ;

    *returnSize = ++rowIndex;
    ret_val     = (int*)malloc(sizeof(int)*(rowIndex));
    ret_val[0]  = temp ;
    for(i=1;i<rowIndex;i++)
    {
        temp *= (rowIndex-i);
        temp /= i;
        ret_val[i] = temp;
    }
    return ret_val;
}

/*
执行结果:
通过
显示详情
执行用时 :0 ms, 在所有 C 提交中击败了100.00% 的用户
内存消耗 :7 MB, 在所有 C 提交中击败了7.32%的用户
*/
```

AlimyBreak
2019.09.12