[**[LeetCode] Candy (分糖果)，时间复杂度O(n)，空间复杂度为O(1)，且只需遍历一次的实现**](https://www.cnblogs.com/felixfang/p/3620086.html)

原题：

There are *N* children standing in a line. Each child is assigned a rating value.

You are giving candies to these children subjected to the following requirements:

* Each child must have at least one candy.
* Children with a higher rating get more candies than their neighbors.

What is the minimum candies you must give?

**Solution (1)**

这题本身可以用贪心法来做，我们用candy[n]表示每个孩子的糖果数，遍历过程中，如果孩子i+1的rate大于孩子i 的rate，那么当前最好的选择自然是：给孩子i+1的糖果数=给孩子i的糖果数+1

如果孩子i+1的rate小于等于孩子i 的rate咋整？这个时候就不大好办了，因为我们不知道当前最好的选择是给孩子i+1多少糖果。

解决方法是：暂时不处理这种情况。等数组遍历完了，我们再一次从尾到头遍历数组，这回逆过来贪心，就可以处理之前略过的孩子。

最后累加candy[n]即得到最小糖果数。

这种解法是需要O(n)的辅助空间给candy[]的。

有没有更好的办法？

**Solution (2) 此方法以及代码部分参考了**[**Shangrila**](http://oj.leetcode.com/discuss/user/Shangrila)**的方法。**

请回想一下：我们为什么需要辅助空间？当孩子的rate是一个非递减曲线的时候，我们是不需要辅助空间的，比如5个孩子的rate分别是1，2，5，7，10。那么糖果数自然是1，2，3，4，5。又如5个孩子的rate分别是1，2，5，5，10，那么糖果数自然是1，2，3，1，2。

因此如果rate是非递减数列，我们可以精确计算出当前孩子应该给多少糖果，把这个糖果数加入总数即可。

当孩子的rate出现递减的情况该如何是好？不用辅助空间能处理吗？

假设5个孩子的rate是 1，5，4，3，2。我们这样计算：遍历时，第一个孩子依然糖果为1，第二个孩子糖果为2，第三个孩子糖果给几个？我们遍历到后面就会知道第二个孩子给的糖果太少了，应该给4个。有没有办法在遍历到后面时，能计算出一个修正值，使得加上这个修正值，正好依然可以使总糖果数是正确的？

其实这个修正值不难计算，**因为可以发现递减数列的长度决定了第二个孩子该给几个糖果**。仔细观察：遍历到第四个孩子时我们知道了第二个孩子不该给2，应该给3，因此Total 要 +=1；遍历到第五个孩子我们知道了第二个孩子不该给3得给4，因此Total 要 += 1。我们设一个变量beforeDenc表示进入递减序列之前的那个孩子给的糖果值，再设置length用来表达当前递减序列的长度。这两个变量就可以决定Total是不是要修正：当遍历第三个孩子的时候 beforeDenc = 2，以后每遍历一个孩子，因为length已经超过了beforeDenc，每次Total都要额外+1，来修正第二个孩子的糖果数。

对于后面三个孩子，我们可以这样计算：遍历到第三个孩子，因为这是递减数列的第二个数字，我们Total += 1；第四个孩子是递减数列的第三个数字，Total += 2；第五个孩子是递减数列的第四个数字，Total += 3。

**可以发现最后三个孩子的糖果总数依然是正确的**，虽然Total 每次增加的糖果数量正好和当前孩子得到的糖果数是反序关系。

这种边遍历边修正的方法可以保证一次遍历，不需要O(n)空间下计算出Total的正确值。

代码：

[复制代码](javascript:void(0);)

int candy(vector<int> &ratings) {

int Total = 0; /// Total candies

int length = 0; /// Continuous descending length of rate

int nPreCanCnt = 1; /// Previous child's candy count

int beforeDenc = nPreCanCnt;

if(ratings.begin() != ratings.end())

{

Total++; //Counting the first child's candy (1).

for(vector<int>::iterator i = ratings.begin()+1; i!= ratings.end(); i++)

{

if(\*i < \*(i-1))

{

length++;

if(beforeDenc <= length)

{

Total++;

}

Total += length;

nPreCanCnt = 1; //This step is important, it ensures that once we leave the decending sequence, candy number start from 1

}

else

{

int curCanCnt = 0;

if(\*i > \*(i-1))

{

curCanCnt = (nPreCanCnt + 1);

}

else

{

curCanCnt = 1;

}

Total += curCanCnt;

nPreCanCnt = curCanCnt;

length = 0; //reset length of decending sequence

beforeDenc = curCanCnt;

}

}

}

return Total;

}

[复制代码](javascript:void(0);)