算法讨论班第十六期 – 王海璐

2015年12月11日星期五

55 Jump Game

Given an array of non-negative integers, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Determine if you are able to reach the last index.

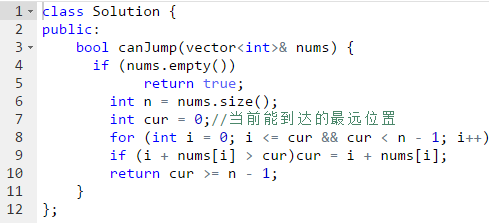
For example:  
A = [2,3,1,1,4], return true.

A = [3,2,1,0,4], return false.

思路1：（从前向后）

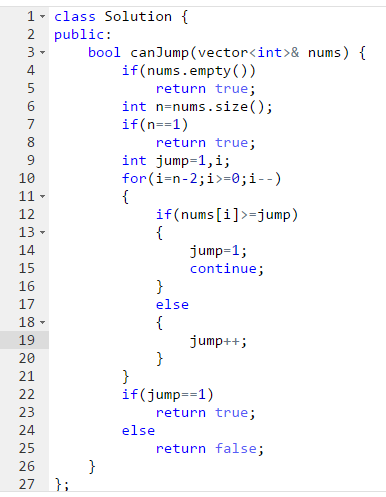
i+A[i] 为在第i个位置所能够跳转到的**最远位置**，进行一次从前向后遍历检查**当前能够达到的位置**所能够到达的最远位置，如果最远位置点能够到达到n-1位置，就返回true.

1. 从前往后，每次记录当前可以到达的最远点。
2. 用cur记录当前可以达到的最远位置,初始化为0，然后依次检查当前可以到达的点i**（i<=cur）**，如果可以达到更远的点**（i+nums[i]>cur）**，则更新最远点cur。
3. 当最远点cur大于等于n-1就返回true。



思路2：（从后往前）

1. 从后往前进行遍历一遍数组，设变量jump为当前位置i可以到达（直接或者间接跳步）last需要的**最小跳步数**，初始化为1，一直到位置0，A[0]能够大于等于最小跳步数返回true.
2. 循环过程中，当前A[i]大于等于jump 时表明从当前位置可以达到last；反之不能到达。
3. 循环过程中当位置i不能到达（直接或者间接跳步）last时候，那么对于i-1位置的跳步数提高了要求，jump++;当i位置可以（直接或者间接跳步）时，前面的i-1位置将只要求可以跳一步即可，即jump=1。



45 Jump Game II

Given an array of non-negative integers, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Your goal is to reach the last index in the minimum number of jumps.

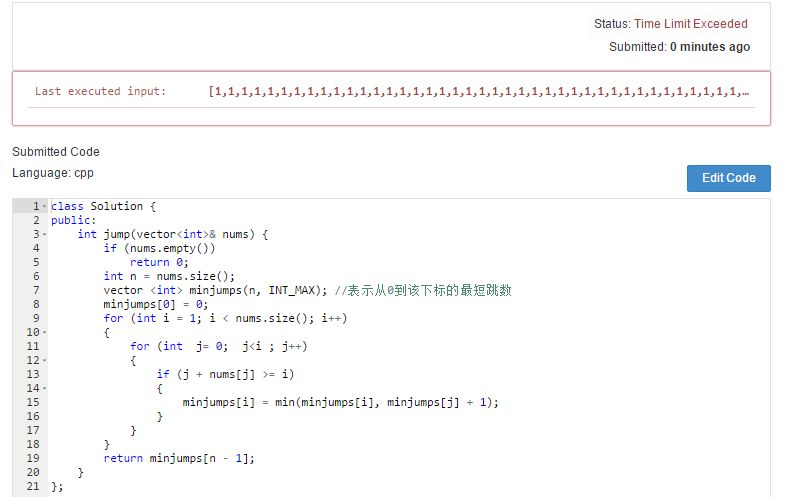
For example:  
Given array A = [2,3,1,1,4]

The minimum number of jumps to reach the last index is 2. (Jump 1 step from index 0 to 1, then 3 steps to the last index.)

思路一：(从前向后)

想到了动态规划的思想：

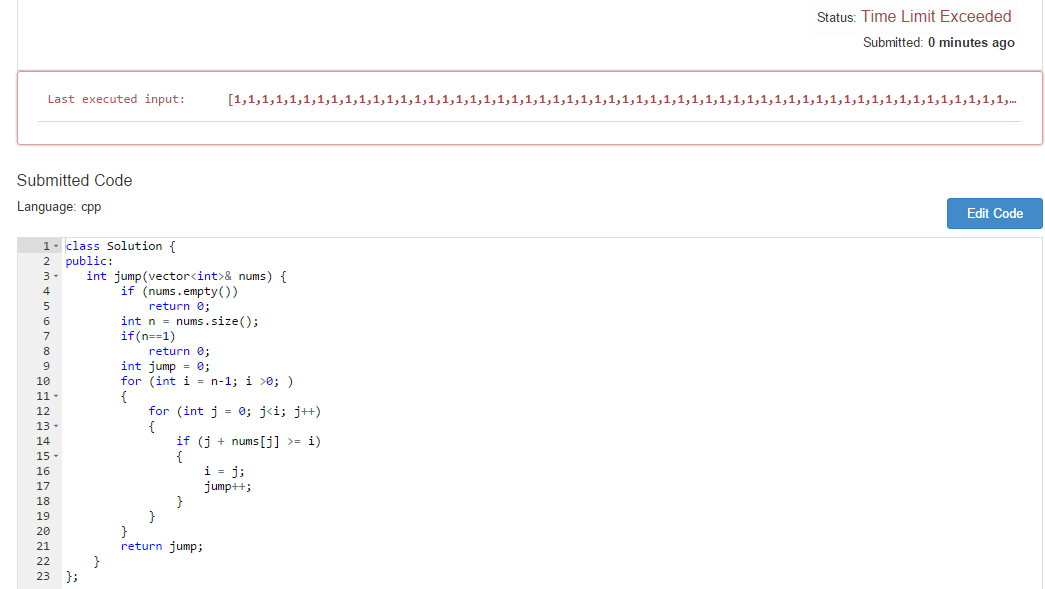
设置一个长度为n的最最小跳转数数组，表示从开始位置0达到给定位置i所需要的最小跳步数。递推关系为：**minjumps[i] = min{minjumps[k] + 1},k<i 且 k+A[k]>=i，**时间复杂度为O(n2),代码调试正确，但是提交显示超时了。



思路二：（从后向前）

从后向前检查能够**通过一跳**到达最后位置点的**最小位置**，然后再检查**通过一跳**到这个位置的最小位置，依次达到位置0。

还没有提交就想到了应该和上面一样会超时，但是这也是可以解决的思路……



思路三：（优化一遍扫描过程）

思考扫描一遍（时间复杂度为O(n)）的思路。

主要思想是：争取每跳都跳到最远。

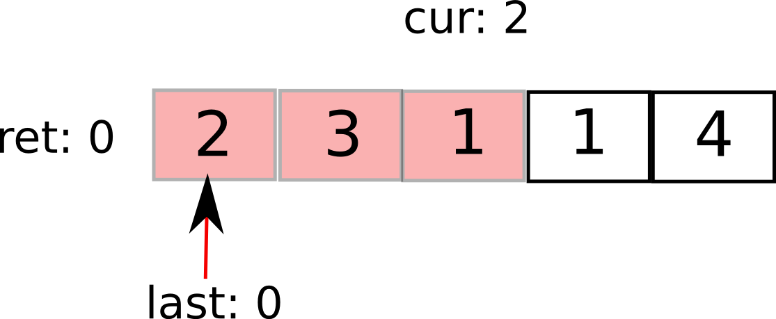
1. 用last表示已跳最小步ret可以达到的最远点，0 <= i <= last作为下一跳最远点探索的范围。
2. 用cur表示最小步（ret+1）可以到达的最远点
3. 更新cur = max(i+A[i]) i满足 0 <= i <= last，即i在上一跳的范围内探索出来的cur。
4. 当 i<last 的时候更新last为cur,跳数ret++。

最后返回ret.

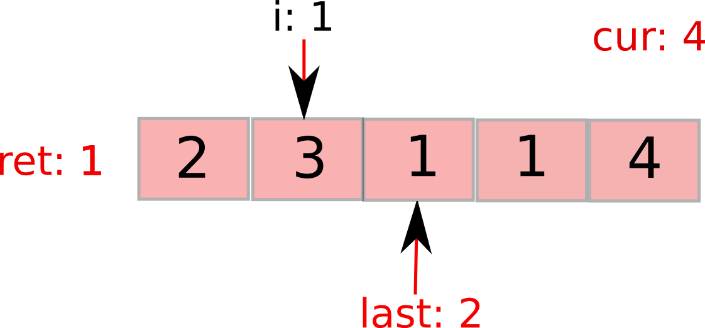
举例：[2,3,1,1,4]。初始状态：cur表示最远能覆盖到的地方，用红色表示。last表示已经覆盖的地方，用箭头表示。它们都指向0。



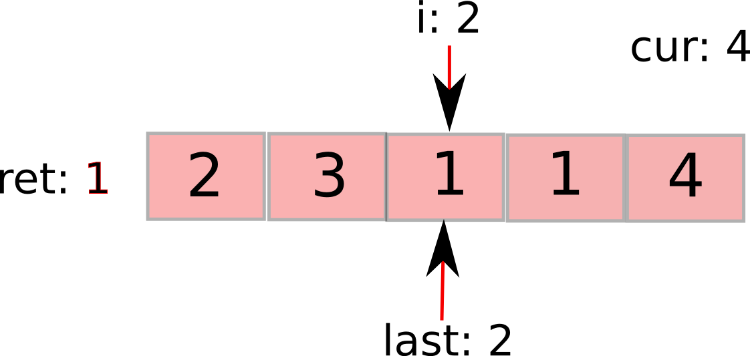
接下来，第一元素告诉cur，最远可以走2步。于是：



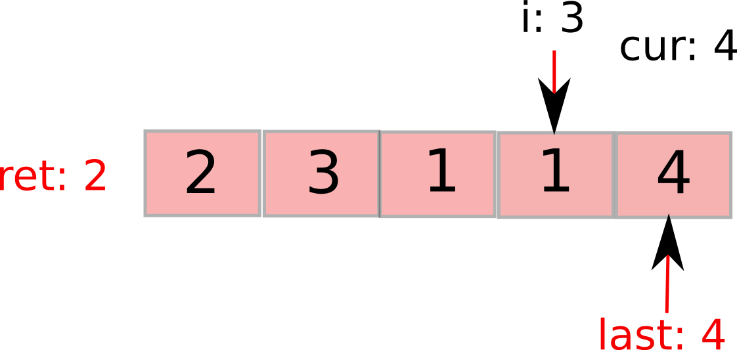
下一循环中，i指向1（图中的元素3），发现， i小于last能到的范围，于是更新last（相当于说，进入了新的势力范围），步数ret加1，同时要更新cur为4。



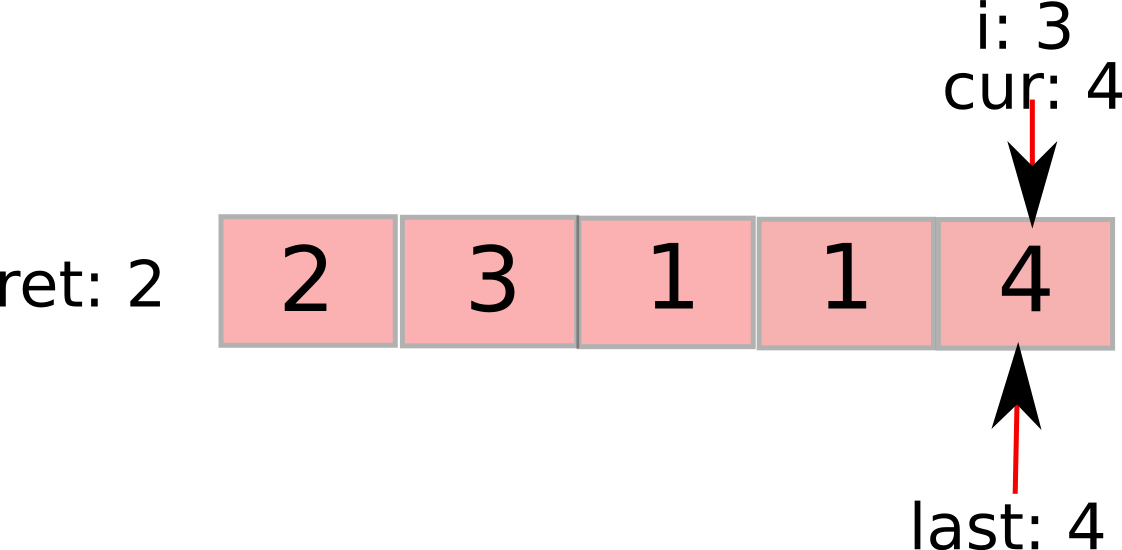
接下来，i继续前进，发现i在当前的势力范围内，无需更新last和步数ret。更新cur。



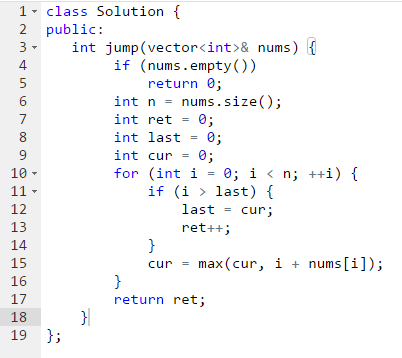
i继续前进，接下来发现**超过**当前势力范围，更新last和步数。cur已然最大了。



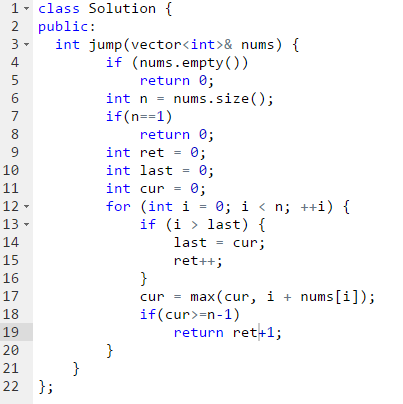
最后，i到最后一个元素。依然在势力范围内，遍历完成，返回ret。



那为啥要用last，因为**直接用cur跳那每次都是跳最远的，但是最优路径不一定是这样。**



可以在上述基础上修改代码，i不用走到最后，只要当cur>=n-1的时候就返回ret+1。



120 Triangle

Given a triangle, find the minimum path sum from top to bottom. Each step you may move to adjacent numbers on the row below.

For example, given the following triangle

[

[2],

[3,4],

[6,5,7],

[4,1,8,3]

]

[

[-1],

[2 3],

[1 -1 -3]

]

The minimum path sum from top to bottom is 11 (i.e., 2 + 3 + 5 + 1 = 11).

**Note:**  
Bonus point if you are able to do this using **only *O*(*n*) extra space**, where *n* is the total number of rows in the triangle.

[Subscribe](https://leetcode.com/subscribe/) to see which companies asked this question

一开始以为很简单的，从上层到下层，每次转到下一行中相邻两个元素中较小的相加就可以了，然后提交就错啦。因为局部和最小未必最终和最小。

动态规划思想：（从底层向顶层计算）

解题思路：

1. 当我们计算第i层的数到底层的最小和时，如果我们知道第i+1层的数到底层最小的和就可以推算出来了。
2. 递推关系：minsum[i][j]=triangle[i][j]+min( minsum[i+1][j] , minsum[i+1][j+1] )；从底层向顶层逐层计算，就能得到最终结果。

**由于only *O*(*n*) extra space，**使用大小为n的数组minsum记录当前层的结果，初始化为最下面一行元素值，直到第0层，minsum[0]就是所求的最小和。空间复杂度为 O（n）。

代码如下：



下期题目：144 145 199