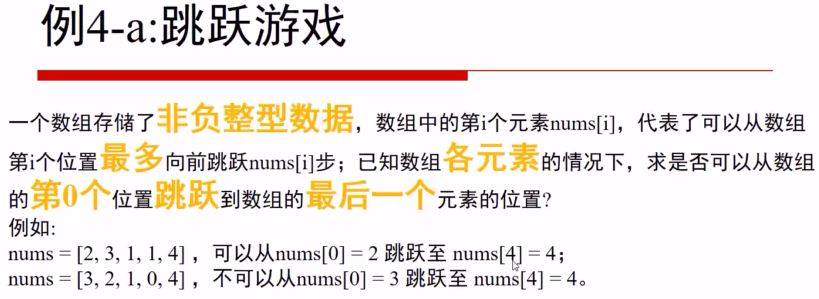
Leetcode\_55&45\_JumpGame\_I&II\_跳跃游戏\_Medium&Hard

# Leetcode\_55\_JumpGame\_跳跃游戏\_Medium

## 题目介绍



\* 难度:Medium

\* https://leetcode.com/problems/jump-game/description/

\* 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.

\*

\* Example 1:

\* Input: [2,3,1,1,4]

\* Output: true

\* Explanation: Jump 1 step from index 0 to 1, then 3 steps to the last index.

\* Example 2:

\* Input: [3,2,1,0,4]

\* Output: false

\* Explanation: You will always arrive at index 3 no matter what. Its maximum

\* jump length is 0, which makes it impossible to reach the last index.

## 思路分析

\* 思路分析：首先根据nums计算每一点可以达到的最远点，存放到新的数组index；

\* 需要定义一个int变量canJumpMaxIndex，存放可跳至最远的点；然后每次都往后跳一步，

\* 不断更新canJumpMaxIndex，直到canJumpMaxIndex大于等于maxIndex就返回true；

\* 若跳的下一步jump大于了可跳至最远点canJumpMaxIndex，则返回false，

\* while循环条件是jump<maxIndex，防止数组下标越界。

\* 若

\* 示例：

\* 下标 = 0 1 2 3 4

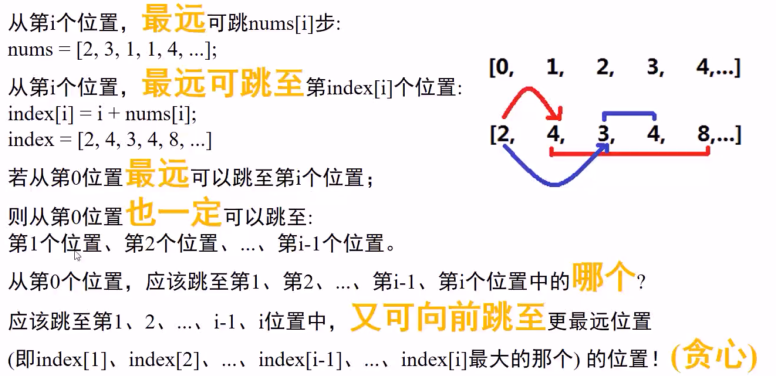
\* nums = 2 3 1 1 4

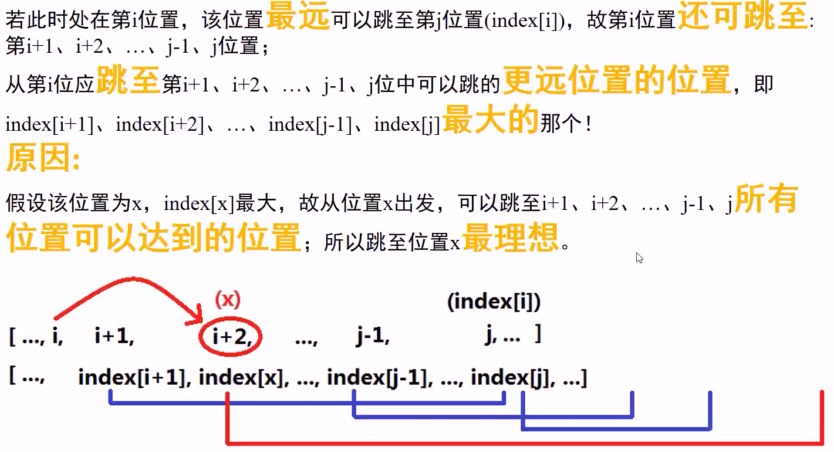
\* index= 2 4 3 4 8，分析过程：下标为0，可跳至下标2的点，canJumpMaxIndex=2；

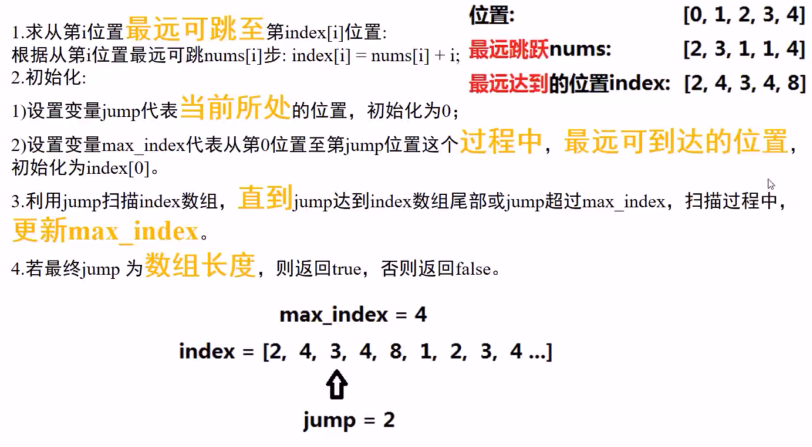
\* canJumpMaxIndex小于maxIndex，继续跳下一步；

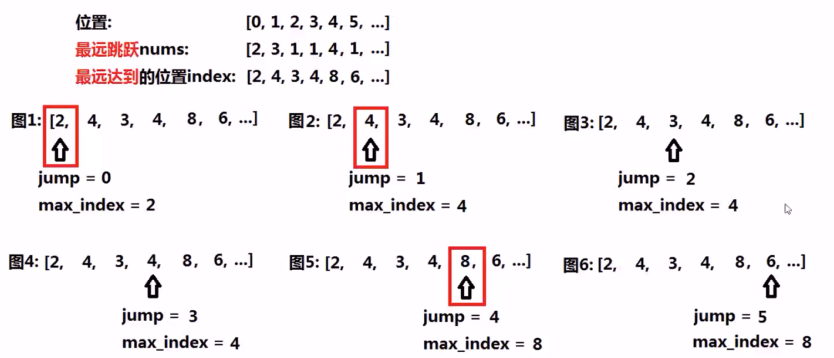
\* 下一步跳至索引为1的点，index[1]>maxJumpMaxIndex,更新maxJumpMaxIndex=4，

\* 即可跳至索引为4的点，满足canJumpMaxIndex>=maxIndex，返回true。









## Java代码

public boolean **canJump**(int[] nums) {

if(nums == null || nums.length == 0) return false;

// if(nums.length == 1&&nums[0] >= 0) return true;//后面已经涵盖

int[] index = new int[nums.length];//用于存放某点可跳的最大的点

int maxIndex = nums.length-1;//最大索引，即最后一点

**for(int i = 0;i < maxIndex;i++)**

**index[i] = nums[i] + i;//计算每一点可跳至的最远点**

int jump = 0,canJumpMaxIndex = index[0];

while**(jump <= maxIndex)**{//防止数组越界

if(jump > canJumpMaxIndex) return false;

if(index[jump] > canJumpMaxIndex) canJumpMaxIndex = index[jump];//更新可跳跃的最大位置

jump++;//继续跳下一步

**if(canJumpMaxIndex >= maxIndex) return true;**//可跳范围大于最大边界，返回true

}

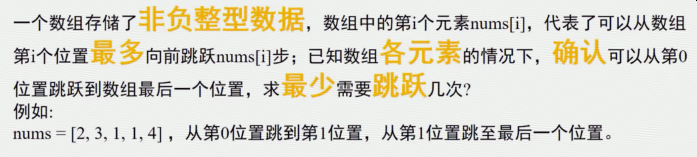
return true;//一般到不了这里，所以返回true或false都可以

}



# Leetcode\_45\_JumpGameII\_跳跃游戏2\_Hard

## 题目介绍



## 思路分析

\* 思路分析：在上一个JumpGame的基础上改进;前提已经假定每一次都可以到达终点了。

\* 目标：寻找最少跳跃次数。

\* 根据贪心规律：只需要每次跳跃最远，最终跳跃次数就是最少的。

\* 因此，只需要保证每跳一次，保证跳跃最远距离即可。

\* 具体实现：

\* 首先还是计算每一点可跳至的最远点index，遍历jump至当前可跳至的最远点，每跳一步，判断下一次跳跃最远点，

\* 从而不断更新canJumpMaxIndex；在此基础上，找出最后一个更新了canJumpMaxIndex的点，该点就是此次选择跳跃的点。

\* 依次类推，最终统计跳跃次数。

\*

\* 可以利用两次while循环实现，更容易理解；

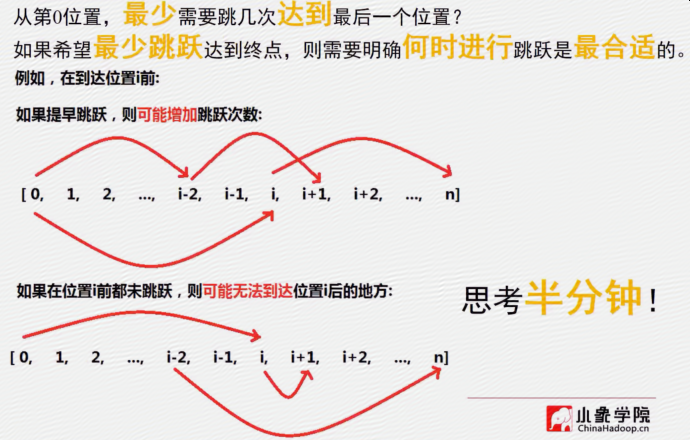
\* 也可以利用一次for循环或while循环实现.

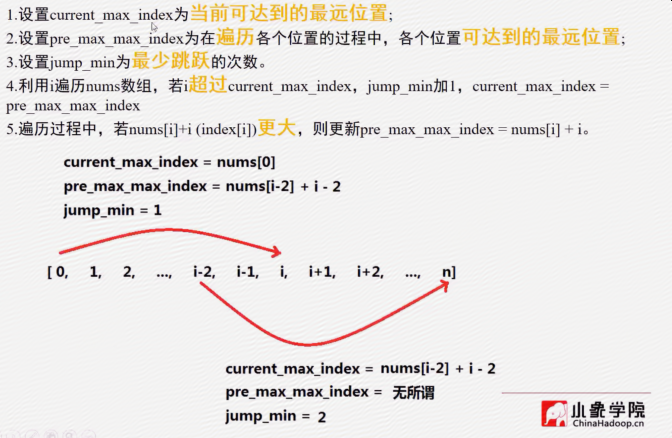
\* 如果要求不能开辟数组，可以利用nums[i]+i代替index[i].

\*

\* 改进：打印出最短跳跃次数对应的跳跃路径。

思考：





## Java代码

### 两层嵌套循环

public int **jump**(int[] nums) {

if(nums == null|| nums.length <= 1) return 0;//边界情况

int[] index = new int[nums.length];//存放每一点可跳至最远点

int maxIndex = nums.length-1;

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

index[i] = i + nums[i];

int jump = 0,canJumpCurrentMaxIndex = index[0],canJumpNextMaxIndex = index[0];

int jumpCount = 0;//跳跃次数

//利用嵌套循环实现：更容易理解。可以将内部while循环整合到大循环之中，jump2

while(jump <= maxIndex){

//当前可跳范围大于maxIndex，直接返回

if(canJumpCurrentMaxIndex >= maxIndex) return jumpCount+1;

while(jump <= canJumpCurrentMaxIndex){//从jump遍历到canJumpMaxIndex

if(index[jump] > canJumpNextMaxIndex)

canJumpNextMaxIndex = index[jump];//保存下一步可跳最远的点

jump++;

}

//while循环出来之后，jump=canJumpCurrentMaxIndex+1，即下一跳点的遍历起点

jumpCount++;//找到了当前跳的点

canJumpCurrentMaxIndex = canJumpNextMaxIndex;

}

return jumpCount+1;

}

### 单层for循环

/\*\*

\* 简化实现

\*/

public int jump2(int[] nums) {

if(nums == null|| nums.length <= 1) return 0;//边界情况

int[] index = new int[nums.length];//存放每一点可跳至最远点

int maxIndex = nums.length-1;

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

index[i] = i + nums[i];

int jump = 0,canJumpCurrentMaxIndex = nums[0],canJumpNextMaxIndex = nums[0];

int jumpCount = 0;//跳跃次数

//基于for循环实现，这种方式

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

if(canJumpCurrentMaxIndex >= maxIndex) return jumpCount+1;//增加这个判断可以简化最后的循环次数

if(i > canJumpCurrentMaxIndex){//找到了上次需要跳点

jumpCount++;

canJumpCurrentMaxIndex = canJumpNextMaxIndex;

}

if(index[i] > canJumpNextMaxIndex){//更新下次可跳最远点

canJumpNextMaxIndex = index[i];

}

}

return jumpCount+1;

}

### 打印最少跳跃点数路径

/\*\*\*

\* 添加新功能：打印最少跳跃点的路径

\*/

public int **jumpAndRoute**(int[] nums) {

if(nums == null|| nums.length <= 1) return 0;//边界情况

int[] index = new int[nums.length];//存放每一点可跳至最远点

int maxIndex = nums.length-1;

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

index[i] = i + nums[i];

int jump = 0,canJumpCurrentMaxIndex = index[0],canJumpNextMaxIndex = index[0];

int jumpCount = 0;//跳跃次数

ArrayList<Integer> route = new ArrayList<Integer>();

int jumpPoint = 0;

route.add(jumpPoint);

while(jump <= maxIndex){

if(canJumpCurrentMaxIndex >= maxIndex) {

route.add(maxIndex);//添加终点

System.out.println("跳跃路径 = " + route.toString());

return jumpCount+1;//下一次可跳范围大于maxIndex，直接返回

}

while(jump <= maxIndex && jump <= canJumpCurrentMaxIndex){//从jump遍历到canJumpMaxIndex

if(index[jump] > canJumpNextMaxIndex){

jumpPoint = jump;

canJumpNextMaxIndex = index[jump];//保存下一步可跳最远的当前点

}

jump++;

}

route.add(jumpPoint);

jumpCount++;

canJumpCurrentMaxIndex = canJumpNextMaxIndex;

}

System.out.println("跳跃路径 = " + route.toString());

return jumpCount+1;

}

