

指导老师: 方芳 汇报人: 周宁

PREF

贪心算法:

在对问题求解时,总是做出在当前看来是最好的选择(贪心);

不从整体最优上加以考虑,所做出的仅是在某种意义上的局部最优解:

贪心算法不是对所有问题都能得到整体最优解,但对范围相当广泛的许多问题,它能产生整体最优解或者是整体最优解的近似解。

特点:既可能得到次优解,也可能得到最优解,依赖于具体问题的特点和"贪心策略"的选取

- > 多步判断+最优子结构性质+贪心选择性质
- > 通过分阶段地挑选最优解, 较快地得到整体的较优解。



Leetcode—55.跳跃游戏

给定一个非负整数数组,你最初位于数组的第一个位置。 数组中的每个元素代表你在该位置可以跳跃的最大长度。 判断你是否能够到达最后一个位置。



示例 1:

输入: [2, 3, 1, 1, 4]

输出: true

解释:从位置 0 到 1 跳 1 步,然后跳 3 步到达最后一个位置。

示例 2:

输入: [3, 2, 1, 0, 4]

输出: false

解释:无论怎样,你总会到达索引为 3 的位置。但该位置的最大跳跃长度是 0 , 所以你永远不可能

到达最后一个位置。

主要思路

思路1: 找数组中的0。观察发现只有当出现0时,才可能导致到达不了最后的位置,那么找到这个0然后再判断是否前面的数组中(i+nums[i])是否能通过这个0

思路2: 采取贪心的思想: 遍历时寻找能到达最远的点即: maxreach = max(maxreach, i + nums[i]);

思路3: 采取动态规划的思想: 每到一个点 i, 我们扫描之前所有的点, 如果之前某点j本身可达, 并且与current 点可达, 表示点i是可达的。

if (can[j] && j + A[j] >= i) can[i] = true;



Leetcode—45.跳跃游戏Ⅱ

给定一个非负整数数组,你最初位于数组的第一个位置。 数组中的每个元素代表你在该位置可以跳跃的最大长度。 判断你是否能够到达最后一个位置。



示例 1:

输入: [2, 3, 1, 1, 4]

输出: true

解释:从位置 0 到 1 跳 1 步,然后跳 3 步到达最后一个位置。

说明:

假设你总是可以到达数组的最后一个位置。

采取贪心的思想:每次在范围内寻找num[i]+i的最大值,直到能够跳到最后的位置,其中的子结构就是这个范围。

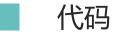
列如: [2, 3, 1, 1, 4, •••]

- ①当i=0时,发现最远到第三个位置(i=2),所以这是第一个范围,在这个范围内都只需要走一步(i=0到i=2)
- ②在i=0到i=2之间这个范围内遍历, 你发现最远可以到第五个位置 (i=4), 那么这是第二个范围 (i=3到i=4)
- ③重复上面过程直到最远可到距离大于最后的位置

主要思路

采取动态规划的思想:从第一个点到最后一个点,走的步数最短,显然是一个最短路径的问题。

设dp[i]表示到下标为i时的最少步数 ,动态转移方程为 $dp[j+i] = min(dp[i]+1, dp[j+i]) (j=1^num[i])$



class Solution: def jump(self, nums:[int]) -> int: if len(nums)<=1:</pre> return 0 res=0 next=nums[0] maxL=nums[0] cur=0 last=0 for i in range(0,len(nums)): if maxL<i+nums[i]:</pre> maxL=i+nums[i] cur=i if i==next: next=maxL last=cur res+=1 if next>=len(nums)-1: break return res+1

代码

```
class Solution {
public:
    int jump(vector<int>& nums)
        // 初始化其他的步数是很大的数
        int n = nums.size();
       vector<int>dp(n, 0x7f7f7f);
       dp[0] = 0;
       for (int i = 0; i < n; i++)
           for (int j = 1; j \le nums[i]; j++)
               if (j+i < n)
                   dp[j + i] = min(dp[i] + 1, dp[j + i]);
       return dp[n-1];
```

结束语 二

个人看法:

贪心算法和动态规划有着许多类似的地方,都拥有最优子结构,但是贪心算法和动态规划相比少一个回溯过程,所以用贪心往往能够更快。

汇报完毕 感谢聆听

Design By zn