1696.跳跃游戏 VI

题目大意

一共 n 个格子，数组 $nums$ 中保存了每个格子的分数，从 0 开始，每次可以向前跳 1 到 k 步，终点是 $n-1$，最大化你的得分。

题目分析

一眼 dp，很快能够盯出状态转移方程：

$$dp[i] = nums[i] + \max_{max\{0,i-k\} \le j < i} dp[j]$$

然而数据范围中提到 $n, k \le 1e5$，如果暴力地双重循环，那么 $O(nk)$ 肯定会超时，需要寻找一种方法来快速地维护 $dp[i-k]$ 到 $$ dp[i - 1]的最大值。

可以使用线段树来实现 $O(\log n)$ 维护，~~但是众所周知leetcode的“中等”难度是普及减~~，搓一个线段树还是太冗余了，有一种聪明的方法可以做到平均 $O(1)$ 维护。

单调队列

本来我一开始就直接写线段树了，但是想想leetcode的中等大概不会涉及到这个内容，上网搜索之后发现了聪明的单调队列算法：

“如果一个人比你年轻，还比你强，那你就可以退役了”

考虑一个队列，当我们把一个新数推进队列时，它会让队列中所有比它小的数字出队（显然队列中的数字都比它年长，因为更早入队），因此考虑让队列中数字从队头到队尾按照从大到小排列，新入队的元素从队尾一路向前 crush ，直到遇到比它大的节点，这样的维护方式同时也保持了我们队列如上的性质，队头元素永远是滑动窗口中的最大值。如果队头元素已经比当前位置落后了 k，那么就让他出队。

观察到，每一个元素只会入队一次，并出队一次，时间复杂度为 $O(n)$。完整代码如下：

```
vector<int> best;
deque<int> choice;
int update(int pos, int k){
    if(!choice.empty() and pos - k >= choice.front()) choice.pop_front();
    while(!choice.empty() and best[choice.back()] < best[pos]) choice.pop_back();
    choice.push_back(pos);
    return best[choice.front()];
```

```
}
    int maxResult(vector<int>& nums, int k) {
    best.push_back(nums[0]);
    for(int i=1;i<nums.size();i++) best.push_back(nums[i] + update(i - 1, k));
    return best[best.size() - 1];
}
```