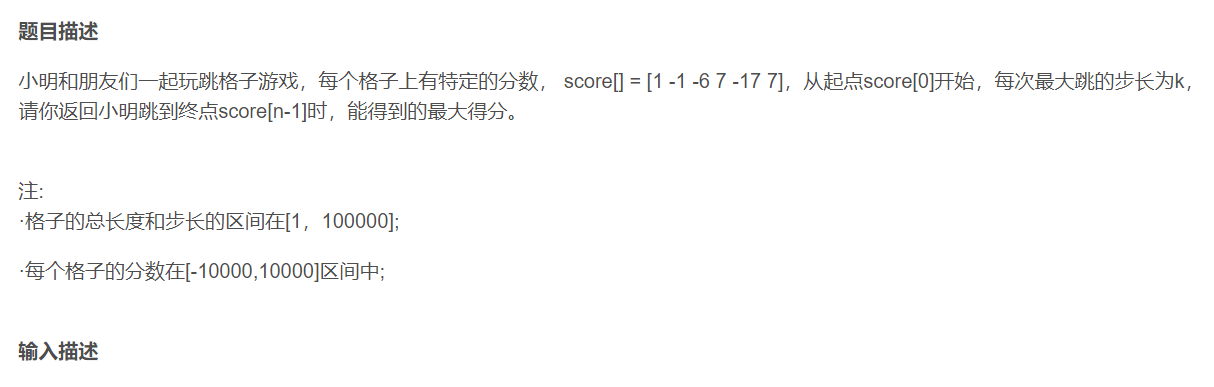
# **E卷-跳格子三[200分]（ Java | Python3 | C++ | C语言 | JsNode | Go）**



6 // 第一行输入总的格子数量

1 -1 -6 7 -17 7 1/ 第二行输入每个格子的分数score[]

2 // 第三行输入最大跳的步长k



14 // 输出最大得分数，小明从起点score[0]开始跳，第一次跳

score[1],第二次跳到score[3],第三次跳到score[5]，因此得到的最大的得分是score[0] + score[1] + score[3] + score[5] = 14



6

1 -1 -6 7 -17 7

2



14



import java.util.\*;

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// 读取输入的总格子数量

int n = sc.nextInt();

// 初始化数组来存储每个格子的分数

long [] arr = new long[n];

for (int i = 0; i < n; i++) {

arr[i] = sc.nextLong();

}

// 读取最大跳的步长

int k = sc.nextInt();

// 使用dp数组来存储到达每个格子的最大得分

long [] dp = new long[n];

// 使用一个Map来存储dp值的计数

Map<Long, Integer> mp = new HashMap<>();

// 使用一个TreeSet来维护当前最大k范围内的dp值

// 使用lambda表达式来定义比较器，按降序排序

TreeSet<Long> st = new TreeSet<>((x, y) -> y.compareTo(x));

// 初始化第一个格子的dp值等于第一个格子的分数

dp[0] = arr[0];

st.add(arr[0]);

mp.put(arr[0], 1);

// 遍历从第二个格子到最后一个格子

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

// 获取当前最大k范围内的dp值

long max = st.first();

// 当前格子的dp值等于最大k范围内的dp值加上当前格子的分数

dp[i] = max + arr[i];

// 将当前dp值添加到TreeSet和Map中

st.add(dp[i]);

mp.put(dp[i], mp.getOrDefault(dp[i], 0) + 1);

// 如果当前格子的位置大于等于k，需要维护k范围内的最大值

if (i >= k) {

long now = dp[i - k];

int ci = mp.get(now);

// 如果dp值的计数为1，移出TreeSet；否则只减少计数

if (ci == 1) {

st.remove(now);

}

mp.put(now, ci - 1);

}

}

// 输出到达最后一个格子的最大得分

System.out.println(dp[n - 1]);

}

}



import heapq

def max\_score(score, k):

n = len(score)

# 使用dp数组来存储到达每个格子的最大得分

dp = [0] \* n

dp[0] = score[0]

# 使用优先队列 (最大堆) 来维护当前最大k范围内的dp值

max\_heap = [(-dp[0], 0)]

for i in range(1, n):

# 获取当前最大k范围内的dp值

while max\_heap[0][1] < i - k:

heapq.heappop(max\_heap)

max\_val = -max\_heap[0][0]

# 当前格子的dp值等于最大k范围内的dp值加上当前格子的分数

dp[i] = max\_val + score[i]

# 将当前dp值加入最大堆

heapq.heappush(max\_heap, (-dp[i], i))

return dp[n - 1]

# 读取输入

n = int(input().strip())

score = list(map(int, input().strip().split()))

k = int(input().strip())

# 计算并输出最大得分

print(max\_score(score, k))



#include <iostream>

#include <vector>

#include <deque>

using namespace std;

int max\_score(vector<int>& score, int k) {

int n = score.size();

// 使用dp数组来存储到达每个格子的最大得分

vector<long long> dp(n, 0);

dp[0] = score[0];

// 使用双端队列来维护当前最大k范围内的dp值

deque<int> deq;

deq.push\_back(0);

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

// 如果队列的头部索引不在当前范围内，移除

if (deq.front() < i - k) {

deq.pop\_front();

}

// 获取当前最大k范围内的dp值

dp[i] = dp[deq.front()] + score[i];

// 维护双端队列，保持队列单调性（从大到小）

while (!deq.empty() && dp[deq.back()] <= dp[i]) {

deq.pop\_back();

}

deq.push\_back(i);

}

return dp[n - 1];

}

int main() {

int n;

cin >> n;

vector<int> score(n);

for (int i = 0; i < n; i++) {

cin >> score[i];

}

int k;

cin >> k;

cout << max\_score(score, k) << endl;

return 0;

}



#include <stdio.h>

#include <stdlib.h>

#define MAXN 100000

// 定义一个结构体用于双端队列的元素

typedef struct {

int index; // 元素在数组中的索引

long long value; // 元素值

} Node;

// 计算最大得分的函数

long long max\_score(int score[], int n, int k) {

long long dp[MAXN]; // 定义dp数组保存到达每个格子的最大得分

Node deque[MAXN]; // 用于维护一个双端队列

int front = 0, back = 0; // 定义双端队列的前后指针

// 初始化第一个格子的dp值

dp[0] = score[0];

deque[back++] = (Node) {

0, dp[0]

}; // 将第一个格子的dp值入队

// 遍历从第二个格子到最后一个格子

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

// 如果队列的头部索引不在当前范围内，则移除

while (front < back && deque[front].index < i - k) {

front++;

}

// 获取当前范围内的最大dp值

dp[i] = deque[front].value + score[i];

// 维护队列，保持队列单调性（从大到小）

while (front < back && deque[back - 1].value <= dp[i]) {

back--;

}

deque[back++] = (Node) {

i, dp[i]

}; // 新的dp值入队

}

// 返回最后一个格子的最大得分

return dp[n - 1];

}

int main() {

int n;

// 读取总格子数量

scanf("%d", &n);

int score[MAXN];

// 读取每个格子的分数

for (int i = 0; i < n; i++) {

scanf("%d", &score[i]);

}

int k;

// 读取最大跳的步长

scanf("%d", &k);

// 计算并输出最大得分

printf("%lld\n", max\_score(score, n, k));

return 0;

}



const readline = require('readline');

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout

});

let input = [];

rl.on('line', (line) => {

input.push(line);

}).on('close', () => {

let n = parseInt(input[0].trim());

let score = input[1].trim().split(' ').map(Number);

let k = parseInt(input[2].trim());

function maxScore(score, k) {

let n = score.length;

// 使用dp数组来存储到达每个格子的最大得分

let dp = new Array(n).fill(0);

dp[0] = score[0];

// 使用双端队列来维护当前最大k范围内的dp值

let deque = [];

deque.push(0);

for (let i = 1; i < n; i++) {

// 如果队列的头部索引不在当前范围内，移除

if (deque[0] < i - k) deque.shift();

// 获取当前最大k范围内的dp值

dp[i] = dp[deque[0]] + score[i];

// 维护双端队列，保持队列单调性（从大到小）

while (deque.length > 0 && dp[deque[deque.length - 1]] <= dp[i]) deque.pop();

deque.push(i);

}

return dp[n - 1];

}

console.log(maxScore(score, k));

});



package main

import (

"bufio"

"fmt"

"os"

"strconv"

"strings"

)

func maxScore(score []int, k int) int {

n := len(score)

// 使用dp数组来存储到达每个格子的最大得分

dp := make([]int, n)

dp[0] = score[0]

// 使用一个双端队列来维护当前最大k范围内的dp值

deque := []int{0}

for i := 1; i < n; i++ {

// 如果队列的头部索引不在当前范围内，移除

if deque[0] < i-k {

deque = deque[1:]

}

// 获取当前最大k范围内的dp值

dp[i] = dp[deque[0]] + score[i]

// 维护双端队列，保持队列单调性（从大到小）

for len(deque) > 0 && dp[deque[len(deque)-1]] <= dp[i] {

deque = deque[:len(deque)-1]

}

deque = append(deque, i)

}

return dp[n-1]

}

func main() {

scanner := bufio.NewScanner(os.Stdin)

// 读取输入

scanner.Scan()

n, \_ := strconv.Atoi(scanner.Text())

scanner.Scan()

scoreStrings := strings.Split(scanner.Text(), " ")

score := make([]int, n)

for i := 0; i < n; i++ {

score[i], \_ = strconv.Atoi(scoreStrings[i])

}

scanner.Scan()

k, \_ := strconv.Atoi(scanner.Text())

// 计算并输出最大得分

fmt.Println(maxScore(score, k))

}