OJ3：比武

陈彦旭 2023.11.3

问题描述

N个士兵站成一列，每个士兵都有一个武力值。对于队伍中任意两个士兵X和Y，如果他们在队伍中相邻，或者他们之间没有士兵的武力值严格大于X和Y的武力值中的较小值，那么他们需要进行一次比武。请计算总共需要进行几次比武。

输入格式

第一行：一个整数N，代表士兵的总数

第2到第N+1行：每行是一个整数，表示队伍中从头至尾每个士兵的武力值

输出格式

一个整数，表示比武的次数

输入样例

```
输入样例1：
8
2
7
1
6
5
3
4
2

输入样例2：
5
4
2
2
2
5
```

输出样例

```
输出样例1：
9

输出样例2：
10
```

提示

请使用scanf/printf实现输入/输出

比武的次数可能很大，超过int的范围

不同士兵的武力值可能相等

可能用到的结论：对于任意士兵Z，排在Z前面且武力值比Z小的士兵不会和排在Z后面的士兵比武

对于全部的测试点，保证1<=每个士兵的武力值<2^(31)

1-2测试样例：N <= 1×10^(3)

3-4测试样例：1×10^(3) < N <= 1×10^(4)

5-10测试样例：1×10^(4) < N <= 5×10^(5)

思路

根据提示："对于任意士兵Z，排在Z前面且武力值比Z小的士兵不会和排在Z后面的士兵比武"，因此只需考虑Z前面且武力值大或等于Z的。对于每一个士兵Z我们都需要知道在他前面的所有大于等于他武力值的士兵的序列，但是并非所有**在Z前面且武力值大于等于Z的**都能和Z后面的士兵比武。从左向右考虑队列，假设有顺序X<Y<Z且武力值Y>X>Z，那么X与Z之间被Y所阻隔而无法比武。因此我们所考虑的这样一个序列（排在Z前面、能和Z后面士兵进行比武的），一定满足一个**非严格的递减序列（下文简称递减序列），且序列末尾大于等于Z的武力值**，这样就保证了该序列中所有士兵武力值都大于等于Z。

在此思路的基础上，我们如果已知士兵k前面的递减序列，那么就能得到士兵k和他前面所有士兵的比武次数（需要根据k武力值和递减序列最后一个人武力值的大小关系进行分类讨论），再将当前士兵加入递减子序列中并进行调整，使之仍满足**以k为结尾且对应武力值非严格递减的序列**（因为序列中存放的是士兵的数组索引），由此遍历数组即可求出比武次数的总和。

而对于递减子序列的调整通过**单调栈**实现。对于一个士兵k和他的武力值`s[k]`，假设他前面已经维护好了一个非严格递减子序列，并使用单调栈`soldier`存放（从栈底到栈顶，对应武力值递增），那么只需考虑`s[k-1]`和`s[k]`的大小关系（因为每次对当前士兵操作结束后都会将其加入压入栈中，成为新的栈顶，因此此时的栈顶为上一次操作的士兵，一定是`k-1`）。

①`s[k-1] >= s[k]`，直接将`k`压入栈，`k`只能和`k-1`比武，比武次数+1。

②`s[k-1] < s[k]`，开始弹栈，直至栈顶对应士兵的武力值大于等于`s[k]`或栈为空，在此过程中每次弹出的士兵都会和士兵k比武。弹栈完成（退出循环）后，若此时栈空则说明`s[k]`为最大元素，无需再次比武。若栈不为空说明`k`前面还有大于等于`s[k]`的，他还需要和k比一次（因为刚才弹出的都是小于`s[k]`的）。最后`k`入栈。

然而，此时问题还没有考虑完全，若在②的弹栈过程结束后栈顶元素和s[k]相等且有重复，则会遗漏比武次数。例如：

移动到当前士兵时，单调栈为`soldier = {7,6,5,5,5,4,3,2,1}`，当前士兵武力值`s[k] = 5`，此时执行操作②，依次将`1, 2, 3, 4`弹出，直到栈顶为`5`，大于等于当前士兵武力值`s[k] = 5`，结束弹栈，然而此时栈的最上方3个5（`soldier[2], soldier[3], soldier[4]`）均能和当前士兵比武，此处仅仅计算了栈顶的一个`soldier[4]`，最终结果一定错误。

因此我们考虑使用结构体，包含武力值`force`及其出现的次数`repeat`。

```
typedef struct Soldier
{
    int force;
    int repeat;
} Soldier;
```

我们摒弃使用**单个数组存放每一个士兵的武力值**的方法，仅使用一个单调栈（数组实现）存放结构体，记录当前已经出现过的武力值及其出现的次数。这样在有重复武力值出现的时候也能够节省空间。

同时，前面的非严格递减子序列也修改为**严格递减子序列**（因为相等元素会被合并，并用重复次数记录），每次弹栈过程都直接加上栈顶元素的重复次数（此处也减少了弹栈的次数）。修改后的新思路如下：

我们可以一边读入新的武力值`Force`，一边与单调栈`soldier`进行下述操作。

①`soldier[top].force > Force`，直接将其压入栈，重复次数`repeat`显然为1，即将`{force = Force, repeat = 1}`压入栈，比武次数+1。

②`soldier[top].force <= Force`，开始弹栈，直至栈顶对应元素的武力值大于`Force`或栈为空，在此过程中每次弹出的元素都会和当前士兵`Force`比武。弹栈过程中注意弹出的武力值在前面出现的次数（`flag`）。弹栈完成（退出循环）后，若此时栈空则说明`Force`为当前最大武力值，无需再次比武。若栈不为空说明前面还有比`Force`更大的武力值，因此栈顶还需要和k比武一次（仅1次，此时即使栈顶重复次数大于1，也只有最右边的那个能和`Force`比武）。最后是`Force`入栈并更新其重复次数（`+flag`），即`{force = Force, repeat = flag + 1}`入栈。

这种方法解决了前面所说的遗漏重复武力值的情况，如：假设当前栈为`soldier = {{8,3}, {7,1}, {6,2}, {5,3}, {4,1}, {3,2}, {2,1}, {1,1}}`，当前`Force = 5`，将`{{1,1}, {2,1}, {3,2}, {4,1}, {5,3}}`依次弹出，且最后弹出`{5,3}`时，记录下`flag = 3`，因此`Force`入栈时压入的是`{force = 5, repeat = 3 + 1}`，因此解决了重复遗漏重复的问题。

最终，时间复杂度和空间复杂度均为`O(n)`。

代码实现

C:

```
#include <stdio.h>

typedef struct
{
    int force;
    int repeat;
} Soldier;

int main()
{
    int N;
    long result = 0;
    scanf("%d", &N);
    Soldier soldier[500000];
    int Force = 0;
    int top = 0; // 记录栈顶位置
    scanf("%d", &Force);
    soldier[0].force = Force;
```

```
    soldier[0].repeat = 1;
    // 第一个士兵首先入栈

    for (int i = 1; i < N; i++)
    {
        scanf("%d", &Force);

        if (soldier[top].force > Force)
        {
            result++;
            top++;
            soldier[top].force = Force;
            soldier[top].repeat = 1;
        }

        else
        {
            int flag = 0; // 记录遇到和自己相同的次数
            while (top >= 0 && soldier[top].force <= Force)
            {
                if (soldier[top].force == Force)
                {
                    flag = soldier[top].repeat;
                    //时刻记录当前弹出士兵的武力值是否和自己相等。
                }
                result += soldier[top].repeat;
                soldier[top].force = 0;
                soldier[top].repeat = 0;
                top--;
            }
            if (top >= 0)
            {
                result++; // 此时栈不为空说明前面还有比自己更大的
            }
            top++;
            soldier[top].force = Force;
            soldier[top].repeat = flag + 1;
        }
    }

    printf("%ld", result);
    return 0;
}

```

C++:

```
#include <cstdio>
#include <stack>
struct Soldier
{
    int force;
    int repeat;
```

```cpp
};

int main()
{
    int N;
    long result = 0;
    scanf("%d", &N);
    std::stack<Soldier> soldier;
    int Force = 0;
    scanf("%d", &Force);
    Soldier s;
    s.force = Force;
    s.repeat = 1;
    soldier.push(s);
    // 第一个士兵首先入栈

    for (int i = 1; i < N; i++)
    {
        scanf("%d", &Force);

        if (soldier.top().force > Force)
        {
            result++;
            s.force = Force;
            s.repeat = 1;
            soldier.push(s);
        }

        else
        {
            int flag = 0; // 记录遇到和自己相同的次数
            while (!soldier.empty() && soldier.top().force <= Force)
            {
                if (soldier.top().force == Force)
                {
                    flag = soldier.top().repeat;
                    //时刻记录当前弹出的武力值是否和自己相等。
                }
                result += soldier.top().repeat;
                soldier.pop();
            }
            if (!soldier.empty())
            {
                result++; // 此时栈不为空说明前面还有比自己更大的
            };
            s.force = Force;
            s.repeat = flag + 1;
            soldier.push(s);
        }
    }

    printf("%ld", result);
    return 0;
}
```

最后附上作者的通过图片~

C：

提交详情（比武）

提交者: 2322022010597　　创建时间: 2023-11-03 14:10:49

运行结果　　**分数** 100.00

#	状态	时间	内存
1	Accepted	0 ms	764 KB
2	Accepted	0 ms	768 KB
3	Accepted	0 ms	776 KB
4	Accepted	0 ms	760 KB
5	Accepted	32 ms	764 KB
6	Accepted	52 ms	788 KB
7	Accepted	64 ms	788 KB
8	Accepted	64 ms	2548 KB
9	Accepted	60 ms	1876 KB
10	Accepted	48 ms	788 KB

C++：

提交详情（比武）

提交者: 2322022010597　　创建时间: 2023-11-03 16:53:51

运行结果　　**分数** 100.00

#	状态	时间	内存
1	Accepted	0 ms	916 KB
2	Accepted	0 ms	916 KB
3	Accepted	0 ms	916 KB
4	Accepted	0 ms	916 KB
5	Accepted	32 ms	912 KB
6	Accepted	52 ms	912 KB
7	Accepted	68 ms	916 KB
8	Accepted	68 ms	2792 KB
9	Accepted	64 ms	2088 KB
10	Accepted	48 ms	920 KB