AcWing-4738：快乐子数组

我们将
$F(B, L, R)$ 定义为整数数组
B 的索引从
L 到
R（包括两者）的子数组的各个元素之和。

更具体的说，$F(B, L, R) = B_L + B_{L+1} + \ldots + B_R$。

如果一个长度为
K 的整数数组
C 满足其所有前缀和均为非负整数，则称数组
C 为快乐数组。

更具体的说，如果
$F(C, 1, 1), F(C, 1, 2), \ldots, F(C, 1, K)$ 均为非负整数，则数组
C 为快乐数组。

给定一个包含
N 个整数的数组
A，请你计算数组
A 中的所有快乐连续子数组的元素和相加的结果。

输入格式

第一行包含整数
T，表示共有
T 组测试数据。

每组数据第一行包含整数 N。

第二行包含
N 个整数
$A_1, A_2, \ldots, A_N$。

输出格式

每组数据输出一个结果，每个结果占一行。

结果表示为 `Case #x: y`，其中

时/空限制：	3s / 64MB
总通过数：	709
总尝试数：	2110
来源： Google Kickstart2022 Round G Problem C	
算法标签	

x 为组别编号（从
1 开始），
y 为所有快乐连续子数组的元素和相加的结果。

数据范围

$1 \le T \le 100$,
$-800 \le A_i \le 800$,
每个测试点最多 30 组数据满足 $1 \le N \le 4 \times 10^5$，其余数据满足 $1 \le N \le 200$。

输入样例：

```
2
5
1 -2 3 -2 4
3
1 0 3
```

输出样例：

```
Case #1: 14
Case #2: 12
```

样例解释

在 Case 1 中，满足条件的快乐连续子数组有
$[1], [3], [3, -2], [3, -2, 4], [4]$，它们的元素和分别为
$1, 3, 1, 5, 4$，相加得到结果
14。

在 Case 2 中，满足条件的快乐连续子数组有
$[1], [1, 0], [1, 0, 3], [0], [0, 3], [3]$，它们的元素和分别为
$1, 1, 4, 0, 3, 3$，相加得到结果
12。

该题由于数据量的关系，只能支持`O(n)`的时间复杂度，故前缀和+双指针的方法依然是复杂度过高的。故这里需要进行一些数学的推导：

假设前缀和数组是`S`，即原数组元素`I[x]=S[x]-S[x-1]`。那么需要符合要求的“快乐连续子数组”只需要确定起始位置`i`，然后找第一个`S[x]-S[i-1]<0`，即找到第一个`S[x]<S[i-1]`，问题就转换为，找到前缀和数组每个元素“右边”的第一个比他小的元素。

这种问题可以通过**单调栈**解决：

单调栈是一种和单调队列类似的数据结构。单调队列主要用于 $O(n)$ 解决滑动窗口问题，单调栈则主要用于 $O(n)$ 解决**NGE问题**（Next Greater Element），也就是，对序列中每个元素，找到下一个比它大的元素。（当然，“下一个”可以换成“上一个”，“比它大”也可以换成“比他小”，原理不变。）

这比单调队列还简单一点。我们维护一个栈，表示“**待确定NGE的元素**”，然后遍历序列。当我们碰上一个新元素，我们知道，越靠近栈顶的元素离新元素位置越近。所以不断比较新元素与栈顶，如果新元素比栈顶大，则可断定新元素就是栈顶的NGE，于是弹出栈顶并继续比较。直到新元素不比栈顶大，再将新元素压入栈。显然，这样形成的栈是单调递减的。

代码如下（输入输出格式见洛谷模板题）：

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    ios::sync_with_stdio(false);
    int n;
    cin >> n;
    vector<int> V(n + 1), ans(n + 1);
    for (int i = 1; i <= n; ++i)
        cin >> V[i];
    stack<int> S;
    for (int i = 1; i <= n; ++i)
    {
        while (!S.empty() && V[S.top()] < V[i])
        {
            ans[S.top()] = i;
            S.pop();
        }
        S.push(i);
    }
    for (int i = 1; i <= n; ++i)
        cout << ans[i] << " ";
    return 0;
}
```

第二个技巧是二重前缀和，即为前缀和数组的二次前缀和，有效避免循环中累加。

Code：

```
#include <iostream>
#include <cstring>
#include <stack>
```

```cpp
#include <algorithm>
using namespace std;

const int N = 4e5 + 10;
long long p[N], pp[N], r[N];

int main () {
    int T; cin >> T;
    for (int t = 1; t <= T; t ++) {
        int n; cin >> n;
        for (int i = 1; i <= n; i ++) {
            int x; scanf("%d", &x);
            p[i] = p[i - 1] + x;
            pp[i] = pp[i - 1] + p[i];
        }

        stack<int> stk;
        stk.push(n + 1), p[n + 1] = -1e9;
        for (int i = n; i >= 0; i --) {
            while (p[i] <= p[stk.top()]) {
                stk.pop();
            }
            r[i + 1] = stk.top() - 1;
            stk.push(i);
        }

        long long res = 0;
        for (int i = 1; i <= n; i ++) {
            int j = r[i];
            res += pp[j] - pp[i - 1] - (j - i + 1) * p[i - 1];
        }
        cout << "Case #" << t << ": " << res << endl;
    }
    return 0;
}
```

代码提交状态：Accepted