

期末复习计划 ☆☆☆

题目描述

考试周即将到来，你制定了一个为期 N 天的复习计划。在第 i 天，你计划背诵 a_i 个单词。

现在你有 Q 个询问，每个询问给出两个整数 L 和 R ，要求你计算从第 L 天到第 R 天（包含第 L 天和第 R 天）之间总共背诵的单词数量。

输入格式

第一行包含两个整数 N 和 Q ，表示天数和询问次数。

第二行包含 N 个整数，表示每天背诵的单词数 a_i 。

接下来 Q 行，每行包含两个整数 L 和 R 。

输出格式

输出共 Q 行，每行输出对应时间段 $[L, R]$ 的单词总数。

输入输出样例

输入	输出 #1
5 3	40
10 20 10 5 30	35
1 3	75
2 4	
1 5	

算法分析与代码实现

本题是 [前缀和 \(Prefix Sum\)](#) 的经典入门题。我们将从原理分析入手，逐步展示对应的代码实现。

1. 暴力法的局限性

如果对于每个询问 (L, R) ，我们都写一个 *for* 循环来累加求和：

- 单次查询的最坏时间复杂度为 $O(N)$ 。
- 总共有 Q 次查询，总时间复杂度为 $O(N \times Q)$ 。
- 题目中 $N, Q \leq 10^5$ ，乘积高达 10^{10} ，远超计算机 1 秒约 10^8 次运算的限制，会导致 **TLE (Time Limit Exceeded)**。

2. 数据类型定义：十年OI一场空，不开long long见祖宗

题目中 $a_i \leq 10^9$ ，且 $N \leq 10^5$ 。极端情况下，区间总和可能达到 $10^5 \times 10^9 = 10^{14}$ 。

而 *int* 类型的最大值约为 2×10^9 。

因此，累加数组 S 必须使用 **long long** 类型，否则会发生整型溢出。

3. 预处理：构造前缀和数组 ($O(N)$)

我们定义一个数组 S ，其中 $S[i]$ 表示前 i 天背诵的单词总数。

$$S[i] = a[1] + a[2] + \cdots + a[i]$$

在代码中，我们不需要每次都重新从头加，而是利用递推公式：

$$S[i] = S[i-1] + a[i] \quad (\text{其中 } S[0] = 0)$$

这意味着：当前的前缀和 = 前一天的前缀和 + 今天的单词数。

```
for (int i = 1; i <= n; i++) {  
    long long x;  
    cin >> x;  
    // 核心递推：利用 S[i-1] 快速计算 S[i]  
    S[i] = S[i-1] + x;  
}
```

4. 查询： $O(1)$ 响应

利用预处理好的 S 数组，区间 $[L, R]$ 的和可以表示为“前 R 天的总数”减去“前 $L-1$ 天的总数”：

$$\text{Sum}(L, R) = S[R] - S[L-1]$$

这样，无论区间多长，我们都只需要做一次减法即可得到答案，时间复杂度降为 $O(1)$ 。

```
while (q--) {  
    int l, r;  
    cin >> l >> r;  
    // 利用公式直接输出，无需循环  
    cout << S[r] - S[l-1] << "\n";  
}
```

拓展思考

1. 逆向思维：差分 (Difference Array)

如果题目需求变了：不是“查询”区间和，而是让你“修改”区间（例如：第 L 天到第 R 天，每天多背 V 个单词），最后只询问一次最终的数组，该怎么办？这时候就需要用到[差分数组](#)。差分是前缀和的逆运算，可以将区间修改操作从 $O(N)$ 优化到 $O(1)$ 。

2. 动态修改与查询

如果题目既有大量的“区间修改”，又有大量的“区间查询”，前缀和（修改慢）和差分（查询慢）就都不够用了。这时候我们需要更高级的数据结构，如[树状数组 \(Binary Indexed Tree\)](#) 或 [线段树 \(Segment Tree\)](#)，它们能将两种操作都维持在 $O(\log N)$ 的复杂度。