A-B 数对 题解

> 2024寒假十大必做——编程打卡（1月22日）
>
> 题目为洛谷P1102 A-B 数对

题目

题目背景

出题是一件痛苦的事情!

相同的题目看多了也会有审美疲劳，于是我舍弃了大家所熟悉的 A+B Problem，改用 A-B 了哈哈!

题目描述

给出一串正整数数列以及一个正整数 C，要求计算出所有满足 $A-B=C$ 的数对的个数（不同位置的数字一样的数对算不同的数对）。

输入格式

输入共两行。

第一行，两个正整数 N,C。

第二行，N 个正整数，作为要求处理的那串数。

输出格式

一行，表示该串正整数中包含的满足 $A-B=C$ 的数对的个数。

样例 #1

样例输入 #1

```
4 1
1 1 2 3
```

样例输出 #1

```
3
```

时空限制

时间限制 $1.00s$

空间限制 $125.00MB$

提示

对于 75% 的数据，$1\leq N\leq 2000$。

对于 100% 的数据，$1\leq N\leq 2\times 10^5$，$0\leq a_i<2^{30}$，$1\leq C<2^{30}$。

题解

题目简意

给出一个长度为 n 的正整数数列 A，对于数列中的每个数 a_i，询问 $a_i + c$ 是否属于数列 A。

朴素解法

暴力枚举 A 和 B，判断是否满足 $A - B = C$ 的条件。

时间复杂度 $O(n^2)$，无法满足 100% 数据要求。

解法一（二分查找）

二分查找第一次等于 $a_i + c$ 的位置和第一次大于 $a_i + c$ 的位置，两位置的下标差就是对于元素 a_i 来说的满足条件的数对的个数。

当然，二分查找的前提是保证数列有序排列，所以需要提前对数列进行排序（以升序为例）。

对于下面这组样例（下标从 1 启用）：

```
5 1
1 1 2 2 3
```

当遍历至 $a_1(1)$ 时，$a_i + c = 2$，按照上述思想，二分查找到数列中第一个等于 2 的位置 a_3，和第一个大于 2 的位置 a_5，两位置的下标作差得到的 2 就是对于元素 a_1 来说的满足条件的数对的个数，将其累加到最终答案中。

时间复杂度 $O(nlogn)$，瓶颈在于排序。

```cpp
#include <cstdio>
#include <algorithm>

constexpr auto maxn = 200001;

int a[maxn];

int main(void)
{
    int n, c;
    scanf("%d %d", &n, &c);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &a[i]);
    }

    std::sort(a + 1, a + 1 + n);        //排序

    long long ans = 0;
    for (int i = 1; i <= n; i++) {
        int left = std::lower_bound(a + i, a + 1 + n, c + a[i]) - a;    // 查找第一次等于a_i + c的位置
        int right = std::upper_bound(a + i, a + 1 + n, c + a[i]) - a;   // 查找第一次大于a_i + c的位置
        ans += right - left;
    }
```

```
    printf("%lld", ans);
    return 0;
}
```

解法二（STL map）

使用 $STL\ map$ 快速建立起序列中的正整数 a_i 与 其出现次数的映射，记为 f。

对于数列中每个元素 a_i，$f[a_i + c]$ 就是对于元素 a_i 来说的满足条件的数对的个数，将其累加到答案中。

时间复杂度$O(nlogn)$，但 $STL\ map$ 基于红黑树实现，常数较大。

```
#include <map>
#include <cstdio>

constexpr auto maxn = 200001;

int a[maxn];
std::map<int, int> f;

int main(void)
{
    int n, c;
    scanf("%d %d", &n, &c);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &a[i]);
        f[a[i]]++;
    }

    long long ans = 0;
    for (int i = 1; i <= n; i++) {
        ans += f[a[i] + c];
    }

    printf("%lld", ans);
    return 0;
}
```

解法三（Hash）

空间换时间，用一个桶保存每个数列中的元素 a_i 出现的次数，记为 f。

对于数列中每个元素 a_i，$f[a_i + c]$ 就是对于元素 a_i 来说的满足条件的数对的个数，将其累加到答案中。

直接使用一个大小为 2^{30} 的桶占用空间太大，不满足题目空间限制，因此我们用哈希表来提高元素密度。哈希函数采用最简单的对素数取余即可，选定的素数 p 需大于 2×10^5。

时间复杂度 $O(n)$。