P1102 A-B 数对

题目描述

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！

好吧，题目是这样的：给出一串数以及一个数字 CCC，要求计算出所有 A−B=CA − B = CA−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，两个整数 N,C。

第二行，N 个整数，作为要求处理的那串数。

输出格式

一行，表示该串数中包含的满足 A−B=C 的数对的个数。

解析

方法1

拿到题目后，我们很容易产生如下思路：首先将全部数据从小到大进行排序，然后开始计算右边界(R)与左边界(L)的差。

但是，这里存在一个问题，当差大于目标值的时候，我们是减小有边界呢？还是增加左边界呢？似乎两种可能性都可以，但是似乎又都有问题。

在这里，我们推荐另外一种做法，即将原来的减法转化为加法：A-B=C转化为A+C=B。这样，只要我们不断的枚举A的值，然后尝试在数组中找到B即可。

需要注意的是本题需要进行双边界查找。

假设输入为,10,3，{4 4 5 5 5 7 7 8 10 10}，如果我们用一般的搜索方法，那么只能得到7对，但是，实际上本题的正确结果应该为11对，即一个4对应两个7，一个7对应两个10。普通的搜索随便搜索到一个目标数字后便停止了，但是，我们希望的是把全部的都搜索到，因此需要搜索两个边界。

另外，本题的计算结果巨大，应该使用long long 类型。

编码

```cpp
#include <bits/stdc++.h>

using namespace std;

//定义全局最大数
const int MaxNum = 1e6 + 1;

//nums 待搜索数组
//target 目标数字
//length 数组长度
//return 目标所在索引位置
int BinarySearchLeft(int nums[], int target, int length) {
    int left = 1; //左边界
    //我们将长度进行了减1
    int right = length;
    //开始循环搜索
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] == target) {
            //持续向左搜索
          right = mid - 1;
        }
            //小于目标，边界右移
       else if (nums[mid] < target) {
            left = mid + 1;
        }
            //大于目标，边界左移
       else if (nums[mid] > target) {
            right = mid - 1;
        }
    }
    //寻找的目标数字不存在于当前数组
    if (left > length || nums[left] != target) {
        return -1;
    }
    return left;
}

//nums 待搜索数组
//target 目标数字
//length 数组长度
//return 目标所在索引位置
int BinarySearchRight(int nums[], int target, int length) {
    int left = 1; //左边界
    //我们将长度进行了减1
    int right = length;
    //开始循环搜索
    while (left <= right) {
```

```
        int mid = left + (right - left) / 2;
        if (nums[mid] == target) {
            //持续向右搜索,左边界增加
          left = mid + 1;
        }
            //小于目标，边界右移
       else if (nums[mid] < target) {
            left = mid + 1;
        }
            //大于目标，边界左移
       else if (nums[mid] > target) {
            right = mid - 1;
        }
    }
    if (right > length || nums[right] != target) {
        return -1;
    }

    return right;
}


int n, target;  //数组长度和目标数字
int a[MaxNum];
int num = 0;

int main() {
    //输入初始数据
   cin >> n >> target;
    for (int i = 1; i <= n; ++i) {
        cin >> a[i];
    }
    //将数据进行排序
   sort(a + 1, a + n + 1);
    //枚举左边界进行搜索
   for (int i = 1; i <= n; ++i) {
        int value = a[i] + target;
        int leftIndex = BinarySearchLeft(a, value, n);
        if (leftIndex != -1) {
            int rightIndex = BinarySearchRight(a, value, n);
            num += rightIndex - leftIndex+1;
        }
    }
    cout << num;
}
```

方法二

对于查找左右边界，实际上C++内置了一个标准库函数。

upper_bound：找到第一个大于目标值的地址，不存在则返回end地址。

lower_bound：找到第一个大于等于目标值的地址，不存在则返回end地址。

用以上的地址减去数组就能够得到它们在数组中的索引，再通过索引差，就能够知道两个目标数字之间相差的数量，而这个数量就是我们要求的对数。

编码

```
#include <bits/stdc++.h>
using namespace std;
//定义全局最大数
const int MaxNum = 1e6 + 1;
int n, target;  //数组长度和目标数字
int a[MaxNum];
long long num = 0;
int main() {
    //输入初始数据
    cin >> n >> target;
    for (int i = 1; i <= n; ++i) {
        cin >> a[i];
    }
    sort(a + 1, a + n + 1);
    //枚举左边界进行搜索
    for (int i = 1; i <= n; ++i) {
        int value = a[i] + target;
        //找到第一个大于目标值的索引
        int rightIndex = upper_bound(a + 1, a + 1 + n, value) - a;
        //找到第一个大于等于目标值的索引
        int leftIndex = lower_bound(a + 1, a + 1 + n, value) - a;
        //计算二者的间隔
        int gap = rightIndex - leftIndex;
        //加上间隔数字就是对数
        num += gap;
    }
    cout << num;
}
```