Atcoder abc 384

https://atcoder.jp/contests/abc384/tasks/

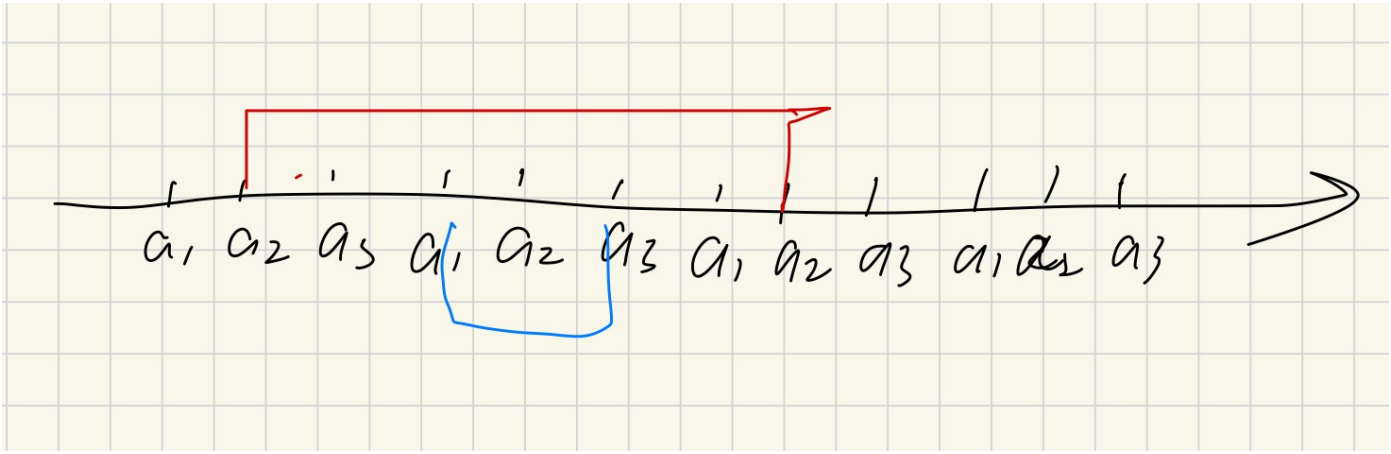
D - Repeated Sequence

这个 D 还是有点意思，写出来了但是可以作为结论记录一下

就是说一个数组，它可以无限循环，比如 $A = [1, 2, 3]$ ，
A无限循环：$[1, 2, 3, 1, 2, 3, 1, 2, 3 \ldots\ldots, 1, 2, 3, \ldots\ldots]$

给你一个值 S ，问，A 的无限循环中，是否存在一个连续的子数组使得子数组和等于 S 。

见下图即可：



红色是一个子数组，一个子数组可以这样分类：

- 包含原数组周期
- 不包含原数组周期
    - 在原数组内
    - 跨越了原数组但是在两个原数组周期内

严格来说就这三类。所以我们只需要对 $target = s \ \% \ sum(A)$ 做一个验证：对于两个原数组组成数组，是否存在一个连续子数组的和等于 $targtet$ 即可。

这里用双指针就行。

时间复杂度 $O(n)$

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    int N;
    long long S;
    cin >> N >> S;
    vector<long long> A(N);
```

```
    for(int i = 0; i < N; i ++) cin >> A[i];
    long long sum = 0;
    for(auto x : A) sum += x;
    vector<long long> B;
    B.reserve(2 * N);
    for(int i = 0; i < 2 * N; i ++) B.push_back(A[i % N]);
    long long k = S / sum;
    bool ok = false;
    long long R = S % sum;
    long long cur_sum = 0;
    int l = 0;
    for(int r = 0; r < B.size(); r ++){
        cur_sum += B[r];
        while(cur_sum > R && l <= r){
            cur_sum -= B[l];
            l ++;
        }
        if(cur_sum == R){
            ok = true;
            break;
        }
    }
    cout << (ok ? "Yes" : "No");
}
```

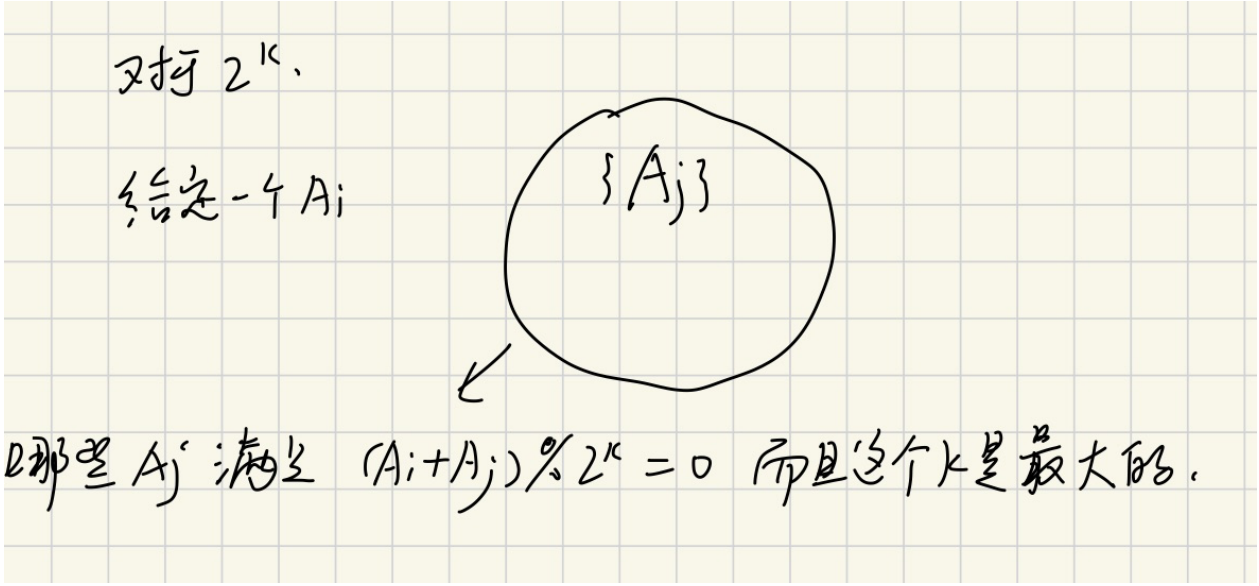
F - Double Sum 2

这题是说，定义 $f(x)$ 为让 x 一直 $\div 2$ 直到 x 是一个奇数，比如
$f(2) = f(1) = 1, f(3) = 3, f(16) = f(8) = f(4) = f(2) = f(1) = 1$ 等等。

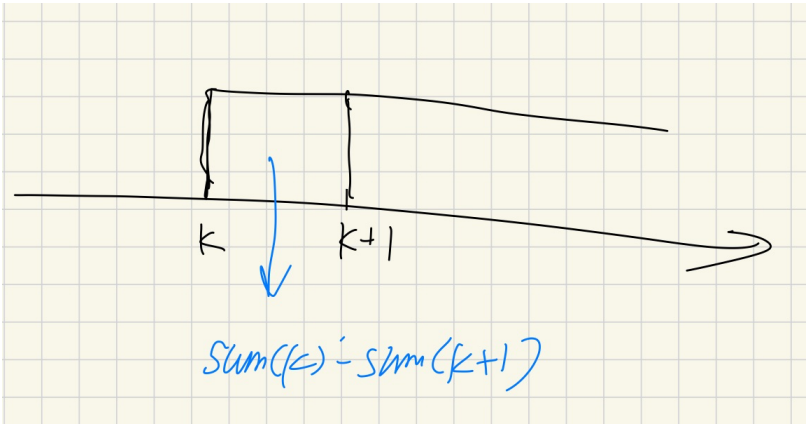
给你一个序列 A_i ，要你求 $\sum_{i=1}^{N} \sum_{j=i}^{N} f(A_i + A_j)$ 。学到东西了。

以前对于这种类型的题目的一个浅显理解，只是一个单纯的「移项」，如果式子或者关系不是那么常见，我自己就不会往这方面去想或者走不下去。但其实更进一步的一个理解是：「分组统计」的思想。

- 首先，我们可以看出一个数的 $f(num)$ 怎么快速求，其实就是 $f(num) = num \div 2^k$ ，其中 2^k 是最大的使得 $num \% 2^k = 0$ 的 2^k 。
- 那么对于 $A_i + A_j$ 我们要找出这样的 2^k ，那么 $f(A_i + A_j) = (A_i + A_j) >> k$ 。
- 不可能去用二重循环去暴力把每个和算出来然后去去除对应二进制末尾 0 然后加入答案。可以考虑做分组统计，怎么统计呢，因为其实 k 最大能到哪里去，2^k 的种类有限，那么可以这样分组，对于一个 A_i ，能和这个 A_i 构成一个和且这个和对应的是某个 2^k 的 A_j 有哪些？类似 「两数之和」的做法，可以用哈希表去维护这个信息。如下图所示：



- 直接求这个最大的 k 是不好求的。
- 虽然对于这个最大的 k 有 $(A_i + A_j)\ \%\ 2^k = 0$，也就是 $A_i \equiv -Aj\ \pmod{2^k}$，那么任何小于 k 的 k' 都满足这个性质。
- 对于上面的结论，换句话说就是，如果我的和满足 $(A_i + A_j)\ \%\ 2^k = 0$，那么说明原本让 $(A_i + A_j)\ \%\ 2^{max\_k} = 0$ 的这个最大的 max_k 一定大于等于这个 k。
- 所以我们可以遍历每个 k，计算满足 $(A_i + A_j)\ \%\ 2^k = 0$ 的所有 $(A_i + A_j)$ 的和为多少，比如为 $sum(k)$，这个和代表什么呢，就是上面所说的，这些和，它们实际的 $max\_k \geq k$。
- 那么利用差分的思想，我们计算了每个 k 对应的 $sum(k)$，那么哪些和它最大就是模 2^k 等于 0 呢？就是 $sum(k) - sum(k+1)$ 这一部分的和了。如下图所示：



- 所以这题就大概做完了。代码中维护信息和两数之和差不多，然后涉及的计算就是例如 $(a + a) + (a + b) + (a + c) = 3 * a + (a + b + c)$ 的这种数学而已。
- 感觉还是讲的很乱。。。。。。。。。。。我目前就这个描述水平了，难绷。
- 可以具体提一下，怎么计算每个 $sum(k)$，遍历每个 A_i，我们去求一下 $(-Aj + 2^k)\ \pmod{2^k}$，可以记为 mod_A_j。
  - 我们用两个哈希表记录两个信息：
  - 一个哈希表为 mod_sum，key 为 每个 mod_A_j，值为这个余数下的每个数的总和。
  - 另一个哈希表为 mod_cnt，key 为每个 mod_A_j，值为这个余数下的每个数都多少个。

- 比如 当前遍历到 A_i，得到 $mod\_A_j = (-Aj + 2^k) \pmod{2^k}$，我们假设这个 mod_A_j 对应的 A_j 比如有这些 $\{A_1, A_3, A_6\}$，那么也就意味着， A_i 分别加上这三个 A_j 都是模 2^k 为 0 的，那么要把这些和加入 $sum(k)$ 中。
- 也就是遍历到 A_i 时，我们希望，
  $sum(k) += (A_i + A_1) + (A_i + A_2) + (A_i + A_3) = 3 * A_i + (A_1 + A_2 + A_3)$ 。
- 而这个 3 可以直接由 $mod\_cnt[mod\_A_j]$ 查询到，后面的和可以直接由 $mod\_sum[mod\_A_j]$ 查询到。维护完 $sum(k)$ 后，对于当前遍历到的 A_i 维护一下两个哈希表的信息即可。

时间复杂度 $O(kn)$，k 是最小的使得 $2^k >= max(A_i * 2)$ 成立的 k 。

```
#include<bits/stdc++.h>
using namespace std;

long long cal(vector<int>& a, int k) {
    long long res = 0;
    unordered_map<int, long long> mod_sum;
    unordered_map<int, int> mod_cnt;
    int p = 1 << k;
    for(int i = 0; i < a.size(); i ++) {
        int mod_j = (-a[i] % p + p) % p;
        if((2 * a[i]) % p == 0) res += 2 * a[i]; //自己和自己配对
        //自己和之前遍历过的元素配对
        res += 1LL * a[i] * mod_cnt[mod_j] + mod_sum[mod_j];
        mod_cnt[a[i] % p] ++;
        mod_sum[a[i] % p] += a[i];
    }
    return res;
}

int main() {
    int n;
    cin >> n;
    vector<int> a(n);
    for(int i = 0; i < n; i ++) cin >> a[i];
    //从数据范围估算 k, 2*1e7 25位大概就可以 保险试下26
    vector<long long> sum(27, 0);
    for(int i = 0; i <= 26; i ++) sum[i] = cal(a, i);
    long long ans = 0;
    for(int i = 0; i < 26 - 1; i ++) ans += (sum[i] - sum[i + 1]) >> i;
    cout << ans << endl;
    return 0;
}
```

G

等我把莫队学一下和线段树加强一下再看吧。。。