【位运算】-分苹果

题目描述与示例

题目描述

A，B两团体想把苹果分为两堆。

A盼望依照它的计算规则**平分苹果**，他的计算是依照二进制加法进行计算，而且不计算进位。

以 `12 + 5` 为例，按照A的计算规则有 `12 + 5 = bin(1100) + bin(0101) = bin(1001) = 9` 成立。

B的计算是最常见的十进制加法，包含进位。B期望在满足A的情形下获取苹果分量最多。

输入苹果的数目跟每个苹果的重量，输出满意A的情形下获取的苹果总重量；假如无法满意A的请求，输出 `-1` 。

输入描述

苹果的数目跟每个苹果分量

输出描述

B在满意A的情形下获取的苹果总分量，假如B无法满意A的请求，输出 `-1` 。

示例一

输入

```
2
12 5
```

输出

```
-1
```

示例二

输入

```
2
12 12
```

输出

```
12
```

示例三

输入

```
3
3 5 6
```

输出

```
11
```

说明

按照A的计算方法 `5 + 6 = 3` ，不进行二进制进位， `bin(101)+ bin(110) = bin(011) = 3` 。再按照B的方法计算， `5 + 6 = 11` 。

解题思路

题干阅读理解

本题的题意非常费解，说人话就是：

- 把数组 `apples` 分成两个部分 `apples1` 和 `apples2` ，分别作为A和B获得的苹果数。
- 分别对 `apples1` 和 `apples2` 求异或和，得到 `xorsum1` 和 `xorsum2`
- `xorsum1` 和 `xorsum2` 需要满足两者相等 `xorsum1 == xorsum2` （即所谓的按照A的方法进行苹果平分）
- 对 `apples1` 和 `apples2` 分别进行**十进制求和**，得到 `apples1_sum` 和 `apples2_sum` 。
- 要求找到一种分苹果的方法，使得 `apples2_sum` 最大，作为B获得的苹果数。

如何满足A的分配规则

对于给定的任意一个数组 `apples` ，我们需要思考**数组本身满足什么条件时，A的分配规则会得到满足。**

由上一步的分析得知，如果A的分配规则满足，那么 `apples` 可以被分成 `apples1` 和 `apples2` 两部分，这两部分的异或和 `xorsum1` 和 `xorsum2` 满足两者相等的条件

```
xorsum1 == xorsum2
```

根据异或操作的性质，很容易得到

```
xorsum1 ^ xorsum2 == 0
```

如果把 `xorsum1` 和 `xorsum2` 分别展开并使用异或操作交换律，由于 `apples1` 和 `apples2` 正好组成了 `apples` ，我们可以得到

```
apples[0] ^ apples[2] ^ apples[1] ^ ... ^ apples[i] ^ ... ^ apples[n-1] == 0
```

上式的左边部分其实是 `apples` 数组的异或和。

换句话说，如果 `apples` 数组的异或和为 `0` ，那么 `apples` 数组一定可以拆成 `apples1` 和 `apples2` 两部分，满足A的分配规则。**进一步地，无论 `apples` 拆成怎么样的两部分，都能够满足A的分配规则。**

所以判断A的分配规则是否能满足的依据非常简单，即判断 `apples` 的异或和是否等于 `0` 即可。

如何贪心地让B获利

如果上述步骤想明白了，剩下的操作实际上非常简单了。由于无论 `apples` 拆成怎么样的两部分，都能够满足A的分配规则，为了让B尽可能多地获得苹果，我们只需要**贪心地**让A获得的那一部分 `apples1` 在**十进制的数值上**尽可能地小即可。A取最小的结果即为 `min(apples)` ，此时B获得的苹果数量为 `sum(apples) - min(apples)` ，即为答案。

上述核心思路整理成代码，实际上非常简短

```
xorsum = 0
for num in nums:
    xorsum ^= num

if xorsum == 0:
    print(sum(nums) - min(nums))
else:
    print(-1)
```

代码

Python

```
# 题目：2024E-分苹果
# 分值：100
# 作者：闭着眼睛学数理化
# 算法：异或位运算
# 代码看不懂的地方，请直接在群上提问


n = int(input())
nums = list(map(int, input().split()))

# 初始化nums数组的异或和xorsum为0
xorsum = 0
# 遍历nums中的每一个元素num，计算所有num的异或和
for num in nums:
    xorsum ^= num

# 如果nums的异或和结果为0
# 说明nums可以按照A的方式进行分配
```

```python
# 被分成两个部分分别分配给A和B
# 为了使得B获得的苹果数量尽可能地多
# 贪心地选择nums中的最小那个数分配给A
# 剩余所有苹果分配给B
if xorsum == 0:
    print(sum(nums) - min(nums))
# 如果nums的异或和结果不为0
# 则无法按照A的方式进行分配
else:
    print(-1)
```

Java

```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        int[] nums = new int[n];

        for (int i = 0; i < n; i++) {
            nums[i] = scanner.nextInt();
        }

        int xorsum = 0;
        for (int num : nums) {
            xorsum ^= num;
        }

        if (xorsum == 0) {
            int sum = 0;
            int min = Integer.MAX_VALUE;
            for (int num : nums) {
                sum += num;
                min = Math.min(min, num);
            }
            System.out.println(sum - min);
        } else {
            System.out.println(-1);
        }
    }
}

```

C++

```cpp
#include <iostream>
#include <vector>
#include <climits>
using namespace std;

int main() {
    int n;
    cin >> n;
    vector<int> nums(n);

    for (int i = 0; i < n; i++) {
        cin >> nums[i];
    }

    int xorsum = 0;
    for (int num : nums) {
        xorsum ^= num;
    }

    if (xorsum == 0) {
        int sum = 0;
        int min = INT_MAX;
        for (int num : nums) {
            sum += num;
            min = min < num ? min : num;
        }
        cout << sum - min << endl;
    } else {
        cout << -1 << endl;
    }

    return 0;
}

```

时空复杂度

时间复杂度：`O(N)`。仅需一次遍历数组。

空间复杂度：`O(1)`。仅需若干常数变量。

相同问题不同描述

2023B-分积木

题目描述

`solo` 和 `koko` 是两兄弟，妈妈给了他们一大堆积木。每块积木上都有自己的重量。现在他们想要将这些积木分为两堆。哥哥 `solo` 负责分配，弟弟 `koko` 要求两个人获得的积木总重量相等（根据 `koko` 的逻辑），个数可以不同，不然就会哭。但 `koko` 只会先将两个数转成二进制再进行加法，而且总会忘记进位（每个进位都会忘记）如当 `25(11101)+11(1011)` 时， `koko` 得到的计算结果是 `18(10010):11001+01011=10010` ， `solo` 想要尽可能让自己得到的积木总重量最大，且不让 `koko` 哭。

输入描述

第一行是一个整数 `N` `(2 <= N <= 100)` 表示有多少块积木

第二行为空格分开的 `N` 个整数 `Ci` `(1 <= Ci <= 10^6)` 表示第 `i` 块积木的重量

输出

让 `koko` 不哭,输入 `solo` 所能获得积木的最大总重量，否则输出 `"No"`

示例

输入

```
3
3 5 6
```

输出

```
11
```

说明

`solo` 能获得重量为 `5` 和 `6` 的两块积木

`5` 转成二进制为 `101`

`6` 转成二进制为 `110`

按照 `koko` 的计算方法(忘记进位)

结果为 `11` (二进制)

`koko` 获得重量为 `3` 的积木转成二进制为 `11`

`solo` 和 `koko` 得到的积木的重量都是 `11` (二进制)

因此 `solo` 可以获得的积木的总重量是 `5+6=11` (十进制)