题解：爱福若歌·佳木谱斯

问题描述

小 F 是一只青蛙（a frog），小 G 是一只鹅（a goose）。池塘里有 `n` 片荷叶自左向右排成一排，小 F 准备从第 1 片荷叶开始向右跳跃，直到跳出这 `n` 片荷叶。每当小 F 跳过第 `i` 片荷叶时，小 G 会给小 F `a_i` 本《算法导论》，如果 `a_i < 0`，则表示小 F 需要给小 G `-a_i` 本《算法导论》。

小 F 的跳跃方式是：在每一轮中，小 F 可以跳跃多步，第 `i` 步跳跃 `i` 片荷叶。也就是说，当小 F 在第 1 片荷叶上时，接下来会跳到第 2、4、7、11 片荷叶。小 F 可以在任意荷叶上开启下一轮跳跃，但需要支付 `w` 本《算法导论》来重置步数。

小 F 希望知道最多能从小 G 处得到多少本《算法导论》。如果小 F 不得不给小 G 《算法导论》，那么视为小 F 从小 G 处得到负数本《算法导论》。

输入

- 第一行包含一个整数 `t`，表示测试数据的数量。
- 接下来 `t` 组测试数据，每组数据包含两行：
  - 第一行包含两个正整数 `n` (2 ≤ n ≤ 100000) 和 `w` (1 ≤ w ≤ 100000)，分别表示荷叶的数量和重置步数的代价。
  - 第二行包含 `n` 个整数 `a_0, a_1, ..., a_{n-1}` (−100000 ≤ a_i ≤ 100000)，表示每片荷叶的重要程度。

输出

对于每组测试数据，输出小 F 能获得的《算法导论》的最大数量。

解题思路

动态规划

为了解决这个问题，可以使用动态规划的方法。定义状态 `dp[i]` 为在跳过 `i` 片荷叶后，能够获得的《算法导论》的最大数量。

状态转移方程

1. **跳跃状态**：
   - 当小 F 从荷叶 `i` 开始跳跃时，他可以跳到 `j = i + step`，其中 `step` 是当前的跳跃步数。
   - 每跳一步就将当前荷叶的价值加到 `sum` 中。
2. **重置状态**：
   - 小 F 可以选择重置跳跃状态，支付 `w` 本书籍，因此可以从 `dp[i]` 中减去 `w`。
   - 状态转移公式如下： [dp[j] = \max(dp[j], sum)]

边界条件

- 初始时 `dp[0] = a[0]`，表示从第 1 片荷叶开始。
- `sum` 用于记录当前跳跃状态下的总价值。

复杂度分析

- **时间复杂度**：O(n)，每组数据最多需要 O(n) 的遍历。
- **空间复杂度**：O(n)，用于存储 `dp` 和 `flag` 数组。

代码实现

以下是完整的代码实现：

```
#include <stdio.h>
#define I long long
#define MAX(a,b) (((a)>(b))?(a):(b))
#include <string.h>

I a[100100];
I dp[101000];
int flag[100100];

int main() {
    int t;
    scanf("%d", &t);
    while (t--) {
        long long res = -1e9;
        int n, w;
        scanf("%d %d", &n, &w);
        for (int i = 0; i < n; i++)
            scanf("%lld", &a[i]);
        dp[0] = a[0];
        dp[1] = a[1] + a[0];
        flag[0] = flag[1] = 1;
        for (int i = 0; i < n; i++) {
            long long sum = dp[i] - w;
            if (i == 0) sum = dp[i];
            int step = 1;
            int j = i + step;
            while (j < n) {
                sum += a[j];
                if (flag[j])
                    dp[j] = MAX(dp[j], sum);
                else dp[j] = sum;
                flag[j] = 1;
                step++;
                j += step;
            }
            if (sum > res) res = sum;
            dp[i] = MAX(dp[i], dp[i]-w);
        }
        printf("%lld\n", res);
        memset(dp, 0, sizeof(dp));
```

```
            memset(flag, 0, sizeof(flag));
        }
        return 0;
    }
```