1 小 A 烤面包

小 A 烤面包

时间限制： 3000MS
内存限制： 589824KB

题目描述：

小 A 每天都要吃 a,b 两种面包名一个。而他有 n 个不同的面包机，不同面包机制作面包的时间各不相同。第 i 台面包机制作 a 面包需要花费 a_i 的时间，制作 b 面包则需要花费 b_i 的时间。为能尽快吃到这两种面包，小 A 可以选择两个不同的面包机 x,y 同时工作，并分别制作 a,b 两种面包，花费的时间将是 $\max(a_x,b_y)$。当然，小 A 也可以选择其中一个面包机 x 制作 a,b 两种面包，花费的时间将是 a_x+b_x。为能尽快吃到面包，请你帮小 A 计算一下，至少需要花费多少时间才能完成这两种面包的制作。

输入描述：

第一行一个正整数 n，表示面包机的个数。

第二行 n 个正整数 a_i，表示面包机制作面包 a 的时间。

第三行 n 个正整数 b_i，表示面包机制作面包 b 的时间。

$1 \le n \le 10^5, 1 \le a_i, b_i \le 10^5$

输出描述：

输出一行一个正整数，表示需要花费的最少时间。

样例输入 1：

```
3
2 5 9
4 3 6
```

样例输出 1：

```
3
```

样例 1 提示：直接将序列中唯一的元素删去即可

样例输入 2：

```
3
2 5 7
2 8 6
```

样例输出 2：

```
4
```

样例 2 提示：可能的一种操作为，删去最后一个元素，再使第一个元素加一，得到的序列为 2 3。

思路分析：

实现代码：

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <climits>
#include <cstdio>

using namespace std;

int main() {
    int n;

```

```
    // 使用 scanf 读取输入
    scanf("%d", &n);

    vector<int> a(n), b(n);

    // 读取 a 数组
    for (int i = 0; i < n; i++) {
        scanf("%d", &a[i]);
    }

    // 读取 b 数组
    for (int i = 0; i < n; i++) {
        scanf("%d", &b[i]);
    }

    // 初始化最小时间为较大的值
    int min_time_same = INT_MAX;
    int min_time_diff = INT_MAX;

    // 情况1: 同一台机器制作两种面包
    for (int i = 0; i < n; i++) {
        min_time_same = min(min_time_same, a[i] + b[i]);
    }

    // 情况2: 两台不同的机器制作面包
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i != j) {
                min_time_diff = min(min_time_diff, max(a[i], b[j]));
            }
        }
    }

    // 取两种情况的最小值
    int result = min(min_time_same, min_time_diff);

    // 输出最小时间
    printf("%d\n", result);

    return 0;
}
```

2 序列修改

序列修改

时间限制： 3000MS
内存限制： 589824KB

题目描述：

给一个长度为 n 的序列和一个整数 x，每次操作可以选择序列中的一个元素，将其从序列中删去，或者将其值加一。问至少操作多少次，可以使操作后的序列 (可以为空) 中数字之和是 x 的倍数。

输入描述：

第一行两个用空格隔开的正整数 n 和 x，含义如问题描述中所述。

第二行是 n 个用空格隔开的正整数 $A[1], A[2], \cdots, A[n]$，表示序列中 n 个元素的值。

$1 \leq n \leq 1000, 1 \leq x \leq 1000, 1 \leq A[i] \leq 1000$

输出描述：

一行一个整数，表示使序列中数字之和是 x 的倍数所需要的最少操作数.

样例输入 1：

```
1 3
4
```

样例输出 1：

```
1
```

样例 1 提示：直接将序列中唯一的元素删去即可

样例输入 2：

```
3 5
1 3 3
```

样例输出 2：

```
2
```

样例 2 提示：可能的一种操作为，删去最后一个元素，再使第一个元素加一，得到的序列为 2 3。

思路分析：

首先判断出这是一道动态规划的算法题，对于动态规划问题，在考虑复杂度的前提下，一般有以下方法：

(1) 基于备忘录的递归方法；

(2) 迭代法；

我个人一般倾向于使用基于备忘录的递归方法，首先递归方法即是穷举法，只是空间复杂度低了，剩下就是对树的裁剪优化，使用备忘录是一个很好的优化方法，相较于迭代法，严格依赖于状态转移方程的正确性，而递归只需要考虑退出递归的条件和每一步状态转移的操作。

1. 问题定义：

   - 我们有一个序列 A 和目标倍数 x。我们要通过删除元素或者加一操作，修改序列使其数字之和是 x 的倍数。
   - 最终需要最少的操作次数。每次操作的可能性包括：
     - 删除元素。

- 将元素加 1。

2. 递归思路：
   - 对于每个元素，我们有三个决策：
     1. 删除当前元素，即跳过该元素并继续递归计算后续元素的操作。
     2. 保留当前元素，并将其值加 1 后递归处理后续元素。
     3. 保留当前元素，不改变它的值，递归处理后续元素。
   - 我们通过递归来遍历所有可能的操作路径，直到处理完所有元素。
   - 在每个递归步骤中，记录当前的序列和，并判断是否已经是 x 的倍数。
3. 备忘录优化：
   - 为了避免重复计算相同的状态，使用一个二维数组 dp 来存储已经计算过的状态。
   - 状态由两个变量决定：
     - 当前处理的序列位置 i。
     - 当前序列的数字之和 `sum % x`（我们只关心序列和对 x 的模，因为我们只需要知道和是否是 x 的倍数）。
   - 如果在某个状态下已经计算过最少操作次数，直接从备忘录中读取结果，避免重复计算。
4. 递归终止条件：
   - 当遍历完所有元素时（即下标 i 达到 `A.size()`），检查当前序列和是否是 x 的倍数。如果是，返回 0 表示不需要更多操作；否则返回 `INT_MAX`，表示该路径不可行。
5. 递归步骤：
   - 我们在每个递归步骤中，计算三种操作的结果：
     1. 删除操作：调用递归处理后续元素，操作数加 1。
     2. 加 1 操作：当前元素加 1 后，调用递归处理后续元素，操作数加 1。
     3. 保留操作：直接保留当前元素，递归处理后续元素。
   - 比较三者的操作次数，取最小值作为当前最优选择。

实现代码：

```cpp
#include <iostream>
#include <vector>
#include <climits>

using namespace std;

// 递归函数，计算当前 idx 和 sum 下的最小操作数
int dfs(int i, int sum, int x, vector<int>& A, vector<vector<int>>& dp) {
    // 基本情况：如果已经检查到序列末尾
    if (i == A.size()) {
        return (sum % x == 0) ? 0 : 100001; // 判断当前和是否为 x 的倍数
    }

    // 检查备忘录，是否已经计算过该状态
    if (dp[i][sum % x] != -1) {
        return dp[i][sum % x]; // 返回已保存的结果
    }

    // 选择 1: 删除当前元素
```

```
    int del_op = dfs(i + 1, sum, x, A, dp);

    // 选择 2: 把当前元素加 1
    int add_op = dfs(i + 1, sum + A[i] + 1, x, A, dp);

    // 选择 3: 保留当前元素（不加一）
    int keep_op = dfs(i + 1, sum + A[i], x, A, dp);

    // 取三种选择的最小操作数
    int res = min(del_op + 1, min(add_op + 1, keep_op));

    // 保存当前状态到备忘录
    dp[i][sum % x] = res;
    return res;
}

int main() {
    int n, x;
    cin >> n >> x;

    // 输入序列
    vector<int> A(n);
    for (int i = 0; i < n; i++) {
        cin >> A[i];
    }

    // 初始化备忘录数组，-1 表示尚未计算
    vector<vector<int>> dp(n, vector<int>(x, -1));

    // 调用递归函数，从索引 0 开始，初始和为 0
    int result = dfs(0, 0, x, A, dp);

    // 输出结果，如果无法达成目标，则返回 -1
    if (result == 100001) {
        cout << -1 << endl;
    } else {
        cout << result << endl;
    }

    return 0;
}
```