Задача A. Волшебные грибы

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

На планете Руук существует Большая Корпорация Маленьких Фей. Одним из видов деятельности, которым испокон веков занимаются ее сотрудницы, является посадка грядок с волшебными грибами. Каждый день, начиная с самого первого дня существования этой корпорации, феи создают одну новую грядку грибов. После этого с новой грядки два дня можно собирать споры, которыми размножаются эти грибы, а потом грядка будет поставлять уже только сам продукт — грибы.

Таким образом, если обозначить количество грибов, посаженных на грядке, созданной в день номер i, как c_i, то оно будет считаться по формуле $c_i = c_{i-1} + c_{i-2}$. Так, в первый и второй дни было посажено по одному грибу, в третий — два, в четвертый — три, в пятый — пять и так далее.

Волшебные грибы являются самым ценным продуктом, который путешественник может привезти с планеты Руук и потом продать особым ценителям. Поэтому первым, что делает любой приезжий, становится поиск грядки с волшебными грибами. Однако, в последнее время все чаще стали появляться сообщения о поддельных волшебных грибах. Тщательное расследование показало, что это является следствием действий Маленькой Корпорации Больших Фей, которая сажает грядки с грибами, внешне не отличимыми, но далеко не такими ценными, как волшебные. Причем, создавая очередную грядку, эти феи сажают туда такое количество грибов, какое их соперницы никогда не сажали и не смогут посадить.

Казалось бы, после выяснения этого факта отличать волшебные грядки от поддельных стало просто. Но обе корпорации существуют достаточно давно, количество грядок и грибов на них давно превысило все разумные пределы.

Главного программиста планеты Руук попросили написать программу, по количеству грибов на грядке сообщающую, является ли эта грядка волшебной. Но вот беда, во время написания программы он находился под воздействием тех самых настоящих волшебных грибов. Поэтому программа, которую он написал, оказалась неправильной. Бинарная версия этой программы доступна по следующей ссылке.

Для того, чтобы придумывать темы занятий, вашим преподавателям жизненно необходимы настоящие волшебные грибы. И если им будут продолжать поставлять ненастоящие грибы, возможно, занятия даже станут нормальными, понятными и полезными. Такого нельзя допустить! Для этого надо найти ошибку в коде программиста планеты Руук.

От вас требуется написать своё решение этой задачи и написать стресс, находящий тест, на котором программа работает неверно. Этот тест надо отослать в проверяющую систему. Ниже предоставлен формат входных и выходных данных программы, ошибку в которой вам надо найти.

Формат входных данных

Первая строка теста содержит одно число N ($1 \leqslant N \leqslant 1000\,000$) — количество исследуемых грядок. Следующие n строк содержат по одному целому положительному числу a_i — количества грибов на исследуемых грядках. Размер теста, который вы должны отправить, не должен превышать 1 Мб.

Формат выходных данных

Для каждого числа, данного во входном файле, выводится «`Yes`», если грядка с таким количеством грибов является волшебной, и «`No`» — если не является. Ответы разделяются переводами строк.

Пример

стандартный ввод	стандартный вывод
8 1 2 3 4 5 6 7 8	Yes Yes Yes No Yes No No Yes

Замечание

Формат отправки теста такой: вы должны отправить один zip или tar архив, содержащий единственную папку «tests». В этой папке должно лежать 2 файла: «001.dat» — сам тест, на котором программа работает неверно и «001.ans» — правильный ответ на этот тест. Более подробно вы можете прочитать тут: https://ejudge.ru/wiki/index.php/Структура_архива_с_тестами. При этом наличие файла «README» не обязательно. Файлы с «.» в начале названия будут игнорироваться. Если у вас есть какие-то вопросы на счёт этого, пишите нам в telegram.

Задача B. Максимальное расстояние

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

По ссылке доступно неправильное решение следущей задачи. Найдите тест, на котором оно работает некорректно: оно получает вердикт `RE`.

Дано число n, далее следуют n массивов, i-й из них имеет длину l_i и состоит из целых чисел. Найдите максимальную разность между соседними элементами в отсортированном массиве среди всех массивов.

Формат входных данных

Первая строка содержит целое число n ($1 \leqslant n \leqslant 10^5$).

Следующие n строк содержат описание массивов, i-я из них содержит целое число l_i ($0 \leqslant l_i \leqslant 10^5$), а затем l_i целых чисел из этого массива. Каждое из чисел массива не превосходит по модулю 10^9.

Сумма l_i — целое число до 10^5. Гарантируется, что найдётся такое i, что $l_i \geqslant 2$.

В конце строк нет пробелов!

Формат выходных данных

Выведите одно целое число — ответ на задачу.

Пример

стандартный ввод	стандартный вывод
2 3 1 4 2 2 15 100	85

Замечание

Формат отправки теста такой: вы должны отправить один zip или tar архив, содержащий единственную папку «`tests`». В этой папке должно лежать 2 файла: «`001.dat`» — сам тест, на котором программа работает неверно и «`001.ans`» — правильный ответ на этот тест. Более подробно вы можете прочитать тут: `https://ejudge.ru/wiki/index.php/Структура_архива_с_тестами`. При этом наличие файла «`README`» не обязательно. Файлы с «`.`» в начале названия будут игнорироваться. Если у вас есть какие-то вопросы на счёт этого, пишите нам в telegram.

Задача C. Да!

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

У вас есть решение следующей задачи, которое получает `WA`. Найдите тест, на котором решение неправильно работает.

В этой задаче вам дана строка s. Ответ в этой задаче всегда «`Yes`».

Неправильное решение примерно для тысячной доли входных строк не работает.

Формат входных данных

Вход содержит единственную строку s длины от 1 до 10^5 из маленьких английских букв.

Формат выходных данных

Выведите `YES`.

Пример

стандартный ввод	стандартный вывод
sorrywearеonicpc	Yes

Замечание

Формат отправки теста такой: вы должны отправить один zip или tar архив, содержащий единственную папку «`tests`». В этой папке должно лежать 2 файла: «`001.dat`» — сам тест, на котором программа работает неверно и «`001.ans`» — правильный ответ на этот тест. Более подробно вы можете прочитать тут: `https://ejudge.ru/wiki/index.php/Структура_архива_с_тестами`. При этом наличие файла «`README`» не обязательно. Файлы с «`.`» в начале названия будут игнорироваться. Если у вас есть какие-то вопросы на счёт этого, пишите нам в telegram.

Задача F. Сложная задача

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	0.25 секунд
Ограничение по памяти:	256 мегабайт

Словарь— это множество слов. Вы должны уметь обрабатывать запросы трех типов:

- «`+ word`» — добавить слово `word` в словарь, если оно в нем не присутствует.
- «`- word`» — удалить слово `word` из словаря, если оно там присутствует.
- «`? text`» — вычислить суммарное количество вхождений всех слов из словаря в текст `text`, при этом, если слово входит в текст несколько раз, то необходимо учесть каждое вхождение.

Гарантируется, что любое слово или текст являются непустыми строками, состоящими из букв `a`, `b` и `c`, суммарная длина которых не превосходит L. Однако, для упрощения задачи перед выполнением каждого запроса необходимо поступить следующим образом: пусть x обозначает ответ на последний запрос `?`, или 0, если таких запросов еще не было. Тогда необходимо очередную строку (`word` или `text`) циклически сдвинуть x раз. Напомним, что циклическим сдвигом строки $s = s_0 s_1 \dots s_{|s|}$ называется строка $s' = s_1 \dots s_{|s|} s_0$.

Формат входных данных

В первой строке дано одно число Q — число запросов. В следующих Q строках находятся запросы. Суммарная длина строк во всех запросах не превосходит L ($L \leqslant 5\,000\,000$)

Формат выходных данных

Эта необычная задача. В ней всего 2 теста, причём ответ не обязательно должен совпадать с ответом на поставленную выше задачу. Первый тест совпадает с тестом из условия и ответ на него должен быть таким же, как в примере. Про второй тест известно только то, что ответ в нём — это одна строчная буква латинского алфавита. Какая именно — не сообщается.

Пример

стандартный ввод	стандартный вывод
11 + a + a - a - ab ? abca + ab + a ? abaaabb ? aaabbab + baa ? babaca	0 6 5 7

Задача G. Сумма минимумов отрезков

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	5 секунд
Ограничение по памяти:	256 мегабайт

В этой задаче вам надо написать не целую программу, а одну единственную функцию
`unsigned SegMinSum(const unsigned *a, const unsigned *b, int n);`
Функция принимает 2 массива a и b (эти массивы могут пересекаться в памяти) и должна посчитать

$$\sum_{i=0}^{n-1} min(a[i], b[i])$$

То есть берётся минимум по первым элементам a и b, это складывается с минимумом по вторым элементам a и b, это складывается с минимумом по третим элементам a и b и так далее, всего обрабатывается n элементов.

Например `SegMinSum({1, 2, 3}, {3, 2, 1}, 3)` = 4, потому что берётся $min(1, 3) + min(2, 2) + min(3, 1) = 4$.

Функция суммарно вызовется не более 200 000 раз, n не превышает 200 000.

Замечание

Про то как работает AVX и как его примерно использовать написано здесь: https://www.codeproject.com/Articles/874396/Crunching-Numbers-with-AVX-and-AVX

Все инструкции AVX написанны здесь:

https://software.intel.com/sites/landingpage/IntrinsicsGuide/#techs=AVX,AVX2

Не забудьте в начале программы написать

```
#include <immintrin.h>
```

Никакие прагмы писать не надо, они работать всё равно не будут.

Задача H. Прибавление арифметической прогрессии

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	5 секунд
Ограничение по памяти:	256 мегабайт

В этой задаче вам надо написать не целую программу, а две функции:

1. `unsigned GetMin(const unsigned *a, int n);`

   Эта функция считает минимум в массиве a длиной n, то есть считает минимум от элементов $a[0], a[1], a[2], \ldots, a[n-1]$. Например `GetMin({2, 1, 3}, 3)` $= 1$.

2. `void AddProg(unsigned *a, int n, unsigned k);`

   Эта функция прибавляет к массиву a длиной n арифметическую прогрессию с коэффициентом k. То есть к $a[0]$ прибавляется k, к $a[1]$ прибавляется $k \cdot 2$, к $a[2]$ прибавляется $k \cdot 3$, и так далее, к $a[n-1]$ прибавляется $k \cdot n$. Функция должна модифицировать сам массив a. Прибавления делаются по модулю 2^{32}. Например $AddProg(\{1, 2, 3\}, 3, 1)$ сделает массив равным $\{2, 3, 4\}$

Функции суммарно вызовутся не более 200 000 раз, n не превышает 200 000.

Замечание

Про то как работает AVX и как его примерно использовать написано здесь: https://www.codeproject.com/Articles/874396/Crunching-Numbers-with-AVX-and-AVX

Все инструкции AVX написанны здесь:

https://software.intel.com/sites/landingpage/IntrinsicsGuide/#techs=AVX,AVX2

Не забудьте в начале программы написать

```
#include <immintrin.h>
```

Никакие прагмы писать не надо, они работать всё равно не будут.

Задача I. Сумма на отрезке

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	8 секунд
Ограничение по памяти:	256 мегабайт

В этой задаче вам надо написать не целую программу, а две функции:

1. `void Prepair(int n, const unsigned short *s);`

   Эта функция подготовится отвечать на запросы сумма на отрезке в массиве s длинной n. Массив состоит из чисел типа `unsigned short`, сумму надо выводить по модулю 2^{16}. Массив s надо копировать.

2. `unsigned short GetSum(int l, int r);`

   Эта функция ищет сумма на отрезке массива s с позиции l по позицию r. Сумму надо выводить по модулю 2^{16}.

В начале будет вызвана функция *Prepair*, а потом будут вызываться какие-то из этих функций. При этом функция *Prepair* может быть вызвана несколько раз. Например возможна такая последовательность вызовов функций:

- `Prepair(3, {1, 2, 3})`
- `GetSum(1, 3)` — надо вернуть 6.
- `GetSum(2, 2)` — надо вернуть 2.
- `Prepair(4, {1, 0, 1, 1})`
- `GetSum(1, 4)` — надо вернуть 3.

Суммарно функции вызываются не более 10^7 раз, при этом в одном вызове функции *Prepair* n не превышает 10^7, суммарно все n по функциям *Prepair* не превышают 10^{10}. По функциям *GetSum* сумма всех $r - l$ не превышает 10^{14}.

Замечание

Про то как работает AVX и как его примерно использовать написано здесь: https://www.codeproject.com/Articles/874396/Crunching-Numbers-with-AVX-and-AVX

Все инструкции AVX написанны здесь:
https://software.intel.com/sites/landingpage/IntrinsicsGuide/#techs=AVX,AVX2

Не забудьте в начале программы написать

```
#include <immintrin.h>
```

Никакие прагмы писать не надо, они работать всё равно не будут.

Задача J. Кол-во на отрезке, больших k

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	2.8 секунд
Ограничение по памяти:	4 мегабайта

Дан массив размера n, есть m запросов посчитать кол-во чисел на отрезке с l_i по r_i, больших k_i. Обратите внимание на ограничение по памяти. Чтобы не словить ML, вы должны завести один глобальный массив типа `int` размера 300 000, на дополнительные массивы памяти не хватит. Так как `iostream` требует большое количество памяти, то вы должны всё считывать и выводить с помощью `printf` и `scanf`.

Формат входных данных

В первой строке даны числа n и m ($1 \leqslant n, m \leqslant 3 \cdot 10^5$).

Во второй строке n чисел — сам массив (числа от 0 до 10^9).

В следующих m строках даны запросы, в строке вводятся l_i, r_i, k.

Формат выходных данных

Выведите m строк — ответы на запросы.

Пример

стандартный ввод	стандартный вывод
5 3 1 2 3 4 5 1 5 3 2 4 4 1 3 0	2 0 3

Замечание

Про то как работает AVX и как его примерно использовать написано здесь: https://www.codeproject.com/Articles/874396/Crunching-Numbers-with-AVX-and-AVX

Все инструкции AVX написанны здесь:

https://software.intel.com/sites/landingpage/IntrinsicsGuide/#techs=AVX,AVX2

Не забудьте в начале программы написать

```
#include <immintrin.h>}
#pragma GCC target("avx,avx2")
```

Только этих прагм не хватит, попробуйте поразвлекаться с другими прагмами и всё таки загнать эту задачу.

Задача K. Прибавление отрезка к отрезку

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	3.7 секунд
Ограничение по памяти:	256 мегабайт

Дан массив s размера n. Есть m запросов поэлементно прибавить отрезок с a_i длинной k_i к отрезку массива с b_i длинной k_i. То есть s_b увеличивается на s_a, s_{b+1} увеличивается на s_{a+1}, s_{b+2} увеличивается на s_{a+2} и так далее, s_{b+k-1} увеличивается на s_{a+k-1}. Отрезки одного запроса не пересекаются. Все прибавления надо делать по модулю 2^{32}.

Формат входных данных

В первой строке вводится числа n и m ($1 \leqslant n, m \leqslant 200\,000$).

Во второй строке вводится массив (числа от 0 до $2^{32} - 1$.

В следующих m строках по 3 числа, в i-й строке 3 числа a_i, b_i, k_i. Отрезки одного запроса не пересекаются.

Формат выходных данных

В единственной строке выведите полученный массив.

Пример

стандартный ввод	стандартный вывод
5 3 1 1 1 1 1 1 3 2 4 1 2 2 5 1	3 2 2 2 3

Замечание

Про то как работает AVX и как его примерно использовать написано здесь: https://www.codeproject.com/Articles/874396/Crunching-Numbers-with-AVX-and-AVX

Все инструкции AVX написанны здесь:

https://software.intel.com/sites/landingpage/IntrinsicsGuide/#techs=AVX,AVX2

Не забудьте в начале программы написать

```
#include <immintrin.h>}
#pragma GCC target("avx,avx2")
```

Только этих прагм не хватит, попробуйте поразвлекаться с другими прагмами и всё таки загнать эту задачу.

Задача L. Сумма на отрезке с изменениями

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	15 секунд
Ограничение по памяти:	256 мегабайт

Эта задача почти как задача «Сумма на отрезке», однако добавляется функция изменения. В этой задаче вам надо написать уже 3 функции:

1. `void Prepair(int n, const unsigned short *s);`

   Эта функция подготовится отвечать на запросы сумма на отрезке в массиве s длинной n. Массив состоит из чисел типа `unsigned short`, сумму надо выводить по модулю 2^{16}. Массив s надо копировать.

2. `unsigned short GetSum(int l, int r);`

   Эта функция ищет сумма на отрезке массива s с позиции l по позицию r. Сумму надо выводить по модулю 2^{16}.

3. `void Change(int a, unsigned short x);`

   Эта функция должна изменить число на позиции a на значение x.

В начале будет вызвана функция *Prepair*, а потом будут вызываться какие-то из этих функций. При этом функция *Prepair* может быть вызвана несколько раз. Например возможна такая последовательность вызовов функций:

- `Prepair(3, {1, 2, 3})`
- `GetSum(1, 3)` — надо вернуть 6.
- `Change(2, 1)` — теперь массив имеет вид $\{1, 1, 3\}$.
- `GetSum(2, 2)` — надо вернуть 1.
- `Prepair(4, {1, 0, 1, 1})`
- `GetSum(1, 4)` — надо вернуть 3.

Суммарно функции вызываются не более 10^7 раз, при этом в одном вызове функции *Prepair* n не превышает 10^7, суммарно все n по функциям *Prepair* не превышают 10^{10}. По функциям *GetSum* сумма всех $r - l$ не превышает 10^{14}.

Замечание

Про то как работает AVX и как его примерно использовать написано здесь: https://www.codeproject.com/Articles/874396/Crunching-Numbers-with-AVX-and-AVX

Все инструкции AVX написанны здесь:

https://software.intel.com/sites/landingpage/IntrinsicsGuide/#techs=AVX,AVX2

Не забудьте в начале программы написать

```
#include <immintrin.h>
```

Никакие прагмы писать не надо, они работать всё равно не будут.