Задача C21. БАЛАНС

⏳0.2 sec. 💾 1024 MB

Кирчо разполага с редица от N цели (не непременно неотрицателни) числа: a_0, a_1, ..., a_{N-2}, a_{N-1}. Той може да прави ходове върху редицата. Един ход се характеризира с двойка числа (l, r), за която $0 \leq l \leq r < N$, като за този ход той може да избере да направи точно една от следните две операции:

- за всяко $x \in \{0, 1, \ldots, r-l-1, r-l\}$ към a_{l+x} се добавя $(-1)^x$;
- за всяко $x \in \{0, 1, \ldots, r-l-1, r-l\}$ от a_{l+x} се изважда $(-1)^x$.

Нека *балансът* на една редица се дефинира като минималния брой ходове, необходими за зануляването на всички нейни елементи. Кирчо иска да определи *баланса* на няколко редици, като всяка от тях се явява подмасив[1] на основната редица. Между отделните заявки за баланс, обаче редицата може да се променя чрез актуализации на отделни елементи. За повече информация относно промените вижте секция *Детайли по имплементацията*. За съжаление Кирчо не може сам да отговори на въпросите, затова той Ви моли да напишете програма **`balance`**, която да решава задачата.

Детайли по имплементацията

Това е интерактивна задача, което означава, че информация между Вашата програма и програмата на журито няма да се обменя чрез стандартния вход и изход, а чрез специални функции.

Трябва да имплементирате следните функции:

```
void init(const std::vector<int>& A);
```

Функцията се извиква точно веднъж в началото на изпълнението на всеки тест. Векторът A съдържа N цели числа: a_0, a_1, ..., a_{N-1}, указващи началната редица.

```
long long balance(int u, int v);
```

Функцията представлява въпрос за *баланса* на подмасива a_u, a_{u+1}, ..., a_{v-1}, a_v. Функцията трябва да върне *баланса* на подмасива. Всички извиквания на `balance()` са независими едно от друго.

```
void update(int p, int x);
```

Функцията представлява промяна на редицата – указва, че стойността на a_p става равна на x. Тази промяна важи за всички следващи извиквания на функции в рамките на текущия тест.

Трябва да предадете към системата файл `balance.cpp`, който съдържа функциите `init()`, `balance()` и `update()`. Той може да съдържа и друг код и функции, необходими за работата на програмата Ви, но **не трябва да съдържа главната функция** `main()`. В началото Вашият файл трябва да съдържа указание към предпроцесора:

[1] Редицата p е подмасив на редицата q, ако p може да бъде получена от q чрез изтриване на няколко (потенциално нула или всички) елементи от началото и няколко (потенциално нула или всички) елементи от края. В частност, една редица е подмасив на себе си.

```
#include "balance.h"
```

Вашата програма ще бъде извикана веднъж за всеки тест.

Ограничения

Нека Q е общият брой извиквания на функциите `balance()` и `update()` в рамките на даден тест.

- $1 \leq N, Q \leq 10^5$
- $-10^9 \leq a_i, x_i \leq 10^9$
- $0 \leq u_i \leq v_i < N$
- $0 \leq p_i < N$
- Гарантирано е, че има поне едно извикване на функцията `balance()`

Подзадачи

Подзадача	Точки	Необходими подзадачи	N	Q	Други ограничения
0	0	–	–	–	Примерният тест.
1	11	–	≤ 20	≤ 100	Отговорът на всеки въпрос е най-много 3.
2	7	–	–	–	Няма промени и $a_i \geq 0$ за всяко i
3	9	–	–	–	Няма промени, редицата е алтернираща[3] и $\lvert a_i \rvert \leq \lvert a_j \rvert$ за всеки $i \leq j$
4	14	–	–	$= 1$	Няма промени, $\lvert a_i \rvert = \lvert a_j \rvert$ за всеки i, j.
5	18	–	–	$= 1$	Няма промени, редицата е алтернираща.
6	15	$2 - 5$	–	–	Няма промени.
7	26	$0 - 6$	–	–	–

Точките за дадена подзадача се получават само ако се преминат успешно всички тестове, предвидени за нея.

Примерна комуникация

[2]Дефинираме алтернираща редица като такава, в която всички числа са ненулеви и знаците на числата в нея се редуват.

№	Действия на журито	Отговор на Вашата програма
1	`init({3,-4,2,-4,3})`	
2	`balance(0,4)`	`6`
3	`balance(1,3)`	`6`
4	`update(0, 2)`	
5	`balance(0,2)`	`4`

В този пример началната редица е $a = (3, -4, 2, -4, 3)$. За първата заявка Кирчо може да направи следната поредица от ходове:

$$(3, -4, 2, -4, 3) \to (3, -4, 3, -4, 3) \to (3, -4, 4, -4, 3) \to (3, -3, 3, -3, 3) \to$$
$$(2, -2, 2, -2, 2) \to (1, -1, 1, -1, 1) \to (0, 0, 0, 0, 0).$$

За втората заявка може да направи следната поредица от ходове:

$$(-4, 2, -4) \to (-4, 3, -4) \to (-4, 4, -4) \to (-3, 3, -3) \to$$
$$(-2, 2, -2) \to (-1, 1, -1) \to (0, 0, 0).$$

След промяната редицата е $a = (2, -4, 2, -4, 3)$ и Кирчо може да направи следната поредица от ходове:

$$(2, -4, 2) \to (1, -3, 1) \to (0, -2, 0) \to (0, -1, 0) \to (0, 0, 0).$$

Може да се докаже, че тези поредици от ходове са оптимални.

Локално тестване

Предоставени са Ви файловете `balance.h` и `Lgrader.cpp`, които можете да компилирате заедно с Вашата програма, за да я тествате на работния си компютър, както и команди за компилиране на двете програми.

При стартиране на локалния грейдър на първия ред на стандартния вход трябва да въведете целите положителни числа N и Q. На втория ред трябва да въведете N числа: $a_0, \ldots, a_{N-1}$, указващи първоначалното състояние на редицата. Всеки от следващите Q реда описва заявка за баканс или промяна. На този етап локалният грейдър ще извика функцията `init()` с параметър $A = \{a_0, \ldots, a_{N-1}\}$. В началото на всяка заявка въведете цялото число t_i, като $1 \leq t_i \leq 2$. Ако $t_i = 1$, то непосредствено след това въведете две числа u_i и v_i, указващи извикване на функцията `balance()` с параметри u_i и v_i. Ако $t_i = 2$, то непосредствено след това въведете две числа p_i и x_i, указващи извикване на функцията `update()` с параметри p_i и x_i.

Локалният грейдър за всяко извикане на `balance()` ще изведе върнатата от функцията стойност. Редът на извикване на функциите е както във входа.