Wild operations (wild)

Filippo vuole testare l'abilità di Francesco nel gestire operazioni selvaggie sugli array, perciò gli ha dato un array $A_0, ..., A_{N-1}$ di lunghezza N.

Ora Filippo chiederà a Francesco di eseguire alcune operazioni sull'array, dove ogni operazione può essere:

- *cambiare* il valore di A_p in x, per un qualche intero x e un indice valido p.
- *perturbare* l'intervallo $[l, r]$, cioè impostare $A_p = \max(A_p, A_{p-1})$ **simultaneamente** per tutti i p tali che $l < p \leq r$.

In qualsiasi momento Filippo può chiedere a Francesco di dirgli il valore di A_p per un qualche indice valido p.

Francesco è molto impegnato, quindi ha deciso di chiedere il tuo aiuto per rispondere alle domande di Filippo.

Implementazione

Devi inviare un singolo file con estensione `.cpp`.

> ⇦ Tra gli allegati di questo problema troverai un template `wild.cpp` con un'implementazione di esempio.

Devi implementare le seguenti funzioni:

C++
```cpp
void init(int N, vector<int> A);
```

- Questa funzione è chiamata una sola volta, all'inizio dell'esecuzione del tuo programma.
- L'intero N è la lunghezza dell'array.
- L'array A, indicizzato da 0 a $N-1$, è l'array iniziale scelto da Filippo.

C++
```cpp
void change(int p, int x);
```

- Questa funzione è chiamata più volte durante l'esecuzione del tuo programma, quando Filippo esegue un cambio.
- L'intero p è l'indice del valore cambiato nell'array.
- L'intero x è il nuovo valore da assegnare.

C++
```cpp
void perturb(int l, int r);
```

- Questa funzione è chiamata più volte durante l'esecuzione del tuo programma, quando Filippo perturba un intervallo.
- L'intero l è l'estremo sinistro dell'intervallo perturbato da Filippo.
- L'intero r è l'estremo destro dell'intervallo perturbato da Filippo.

C++
```cpp
int calc(int p);
```

- Questa funzione è chiamata più volte durante l'esecuzione del tuo programma, quando Filippo chiede il valore di un elemento dell'array.
- L'intero p è l'indice dell'elemento richiesto da Filippo.
- La funzione deve restituire il valore di A_p dopo aver applicato tutte le operazioni precedenti.

Grader di Esempio

Una versione semplificata del grader usato durante la correzione è disponibile nella cartella relativa a questo problema. Puoi usarla per testare le tue soluzioni in locale. Il grader di esempio legge i dati di input da `stdin`, chiama le funzioni che devi implementare e scrive su `stdout` nel seguente formato.

Sia Q il numero totale di cambi, perturbazioni e domande fatte da Filippo.
Quindi, il file di input è composto da $2 + Q$ righe, contenenti:

- Riga 1: gli interi N, Q.
- Riga 2: N interi $A_0, ..., A_{N-1}$, i valori iniziali dell'array.
- Riga $3 + i$ ($0 \leq i < Q$): 2 o 3 interi, in uno dei seguenti formati:
  - 1 p x: significa che Filippo cambia A_p in x.
  - 2 l r: significa che Filippo perturba l'intervallo $[l, r]$;
  - 3 p: significa che Filippo chiede il valore di A_p.

Il file di output è composto da Q_3 righe (dove Q_3 è il numero di chiamate a `calc`) contenenti i valori restituiti dalla funzione `calc`.

Assunzioni

- $1 \leq N \leq 400\,000$.
- $0 \leq Q \leq 400\,000$.
- $1 \leq A_i \leq 10^9$ per ogni $0 \leq i < N$.
- $0 \leq p < N$ in ogni chiamata a `change` e `calc`.
- $0 \leq l < r \leq N - 1$ in ogni chiamata a `perturb`.
- $1 \leq x \leq 10^9$ in ogni chiamata a `change`.

Assegnazione del punteggio

Il tuo programma sarà testato su diversi casi di test raggruppati in subtask. Per ottenere i punti di un subtask, devi risolvere correttamente tutti i suoi casi di test.

Sia Q_1 il numero di chiamate alla funzione `change` in un caso di test, allora:

- **Subtask 0 [0 punti]**: Caso d'esempio.
- **Subtask 1 [15 punti]**: La funzione `change` non viene mai chiamata; $l = 0$, $r = N - 1$ in ogni chiamata a `perturb`.
- **Subtask 2 [16 punti]**: $A_i \leq 10$ per ogni $0 \leq i < N$ e $x \leq 10$ in tutte le chiamate a `change`.
- **Subtask 3 [13 punti]**: Le chiamate alla funzione `change` non diminuiscono i valori ($x \geq A_p$), $Q_1 \leq 1000$ e $l = 0$, $r = N - 1$ in ogni chiamata a `perturb`.
- **Subtask 4 [22 punti]**: La funzione `change` non viene mai chiamata.
- **Subtask 5 [14 punti]**: Le chiamate alla funzione `change` non diminuiscono i valori ($x \geq A_p$), $Q_1 \leq 1000$.
- **Subtask 6 [20 punti]**: Nessuna limitazione aggiuntiva.

Esempi di input/output

stdin	stdout
10 28 5 1 7 8 3 2 5 6 9 4 1 1 1 1 0 1 2 0 1 2 2 6 1 6 5 2 2 9 2 2 5 2 4 5 1 4 5 2 3 8 1 8 4 3 0 1 6 3 1 4 1 2 5 7 1 0 3 2 4 5 1 6 3 3 0 3 1 3 2 3 3 3 4 3 5 3 6 3 7 3 8 3 9	1 3 1 7 8 1 8 3 6 4 9

Spiegazione

Iniziamo con l'array $A = [5, 1, 7, 8, 3, 2, 5, 6, 9, 4]$.

- Evento 1: Filippo cambia A_1 in 1 (era già 1): il nuovo array è $[5, 1, 7, 8, 3, 2, 5, 6, 9, 4]$.
- Evento 2: Filippo cambia A_0 in 1: il nuovo array è $[1, 1, 7, 8, 3, 2, 5, 6, 9, 4]$.
- Evento 3: Filippo perturba $[0, 1]$: il nuovo array è $[1, 1, 7, 8, 3, 2, 5, 6, 9, 4]$.
- Evento 4: Filippo perturba $[2, 6]$: il nuovo array è $[1, 1, 7, 8, 8, 3, 5, 6, 9, 4]$.

Dall'evento 19 in poi, Filippo chiede solo valori nell'array senza eseguire cambi o perturbazioni. A questo punto l'array è $[3, 1, 7, 8, 1, 8, 3, 6, 4, 9]$.