Problema Floppy

C header: floppy_c.h
C++ header: floppy.h

Per Roxette l'informatica è uno stile di vita. Di notte sogna alberi e di giorno immagina matrici. Oggi, passeggiando, è inciampata in un array $v_0, v_1, v_2, \ldots, v_{N-1}$ di N interi **distinti**. Trovandolo molto interessante, Roxette ha deciso di salvarlo su un floppy disk. Purtroppo, data la bassa capacità del floppy, Roxette non può salvare l'intero array: al suo posto, salverà un array di bit che le permette di rispondere a qualsiasi query del tipo:

$query(a, b)$ = l'indice del valore massimo nel sottoarray che va da a a b.

In altre parole:

$query(a, b) = a \leq id \leq b$ e $v_{id} = max(v_a, v_{a+1}, \ldots, v_{b-1}, v_b)$.

Aiuta Roxette implementando due funzioni. Una prima funzione, dato l'array interessante, calcola la sequenza di bit da salvare sul suo floppy. Una seconda funzione, data la sequenza di bit salvata sul floppy e le query alle quali Roxette vuole rispondere, restituisce la risposta corretta per ciascuna query.

Implementazione

Questo è un task a doppia interazione!
Devi implementare due funzioni. La prima è la seguente:

```
(C)   void read_array(int subtask_id, int N, int* v);
(C++) void read_array(int subtask_id, const std::vector<int> &v);
```

Questa funzione verrà chiamata **esattamente una volta** durante la prima interazione, e ti viene fornito l'array di Roxette. Durante l'esecuzione della funzione dovrai chiamare **esattamente una volta** questa funzione definita del grader:

```
(C)   void save_to_floppy(int L, char* bits);
(C++) void save_to_floppy(const std::string &bits);
```

Questa funzione dice a Roxette quale sequenza di bit salvare nel suo floppy disk. Il parametro `L` indica il numero di bit da salvare, mentre il parametro `bits` deve essere una sequenza di caratteri (solo `'0'` e `'1'` sono permessi).

La seconda funzione che devi implementare è:

```
(C)   int* solve_queries(int subtask_id,
                         int N, int L, char* bits,
                         int M, int* a, int* b);
(C++) std::vector<int> solve_queries(int subtask_id,
                                     int N, const std::string &bits,
                                     const std::vector<int> &a,
                                     const std::vector<int> &b);
```

Questa funzione viene chiamata **esattamente una volta** durante la seconda interazione, e ti fornisce la lunghezza dell'array originale e l'insieme di bit che Roxette ha memorizzato nel floppy disk. In più vengono forniti i parametri delle M query a cui devi rispondere.

I parametri della i-esima query sono `a[i]` e `b[i]`.

La funzione deve restituire un array di M interi, rappresentati la risposta alle M query.

Attenzione: Il programma verrà eseguito due volte, una per ogni interazione. Quindi qualunque risultato calcolato durante la prima esecuzione non sarà accessibile durante la seconda.

Assunzioni

- $-10^9 \leq v_i \leq 10^9$ per ogni $0 \leq i \leq N-1$
- $1 \leq L \leq 200\,000$

Subtask 1 (7 punti)

- $1 \leq N \leq 500$
- $0 \leq v_i < N$ per ogni $0 \leq i \leq N-1$
- $1 \leq M \leq 1000$
- Il punteggio di questo subtask è di 7 punti solo se tutti i testcase sono risolti correttamente, sarà 0 altrimenti.

Subtask 2 (21 punti)

- $1 \leq N \leq 10\,000$
- $1 \leq M \leq 20\,000$
- Il punteggio di ogni testcase è $\min\left(1, \frac{1}{2^{\frac{L}{N}-1-\log_2 N}}\right)$. Il punteggio del subtask sarà il minimo di quello dei testcase, moltiplicato per 21.

Subtask 3 (72 punti)

- $1 \leq N \leq 40\,000$
- $1 \leq M \leq 80\,000$
- Il punteggio di ogni testcase è $\min\left(1, \frac{1}{2^{\frac{L}{N}-2}}\right)$. Il punteggio del subtask sarà il minimo di quello dei testcase, moltiplicato per 72.

Esempio di interazione

Durante la prima interazione, la funzione `read_array` verrà chiamata con questi valori:

```
read_array(
  /* subtask_id = */ 3,
  /* v          = */ {40, 20, 30, 10});
```

Questa funzione potrebbe decidere di salvare i seguenti bit sul floppy:

```
save_to_floppy(
  /* bits = */ "001100");
```

Il programma a questo punto viene terminato (cancellando quindi tutti i dati in memoria) e rieseguito.

Nella seconda interazione, la funzione `solve_queries` verrà chiamata con questi valori:

```
solve_queries(
  /* subtask_id = */ 3,
  /* N          = */ 4,
  /* bits       = */ "001100",
  /* a          = */ {0, 0, 0, 0, 1, 1, 1, 2, 2, 3},
  /* b          = */ {0, 1, 2, 3, 1, 2, 3, 2, 3, 3});
```

La funzione dovrebbe quindi restituire un array di M interi:

```
  {0, 0, 0, 0, 1, 2, 2, 2, 2, 3}
```