D2-A&B - Optimal Range Minimum Query

Time Limit: 1 sec & 8 secs.
Memory Limit: 2048 MB.

Problem Description

In this assignment you will implement the optimal algorithm for the range minimum query problem as a function library in C++.

You must implement the following two functions:

1. `void warm_up( int seq[], int n );`
2. `int query( int left, int right );`

The source code you submit will be used as a *subroutine* to solve the RMQ problem. Before any query, the function warm_up() will be called and the static data will be passed to this function. You may assume that the input data is a valid integer array that contains n elements. When this function finishes, the array `seq[]` may or may not exist. Hence, it is your responsibility to store the data for later queries.

After the function warm_up() is called, the external program will use the function query(left, right) to query the minimum value in `seq[left...right]`. You should return **the minimum value** of the element within `seq[left...right]`.

If `[ left, right ]` does not corresponds to a valid range, return `-1` instead.

Requirements and Specs

For your submission to be judged correctly, make sure that you adhere the following requirements.

- Select the language "`C++ - Function only`" when submitting your program.

  The file you submit *must not* contain the `main` function. Otherwise it will result in a compilation error.

- You must include in your source file a line containing the following comment as an identifier.

```
/* ProbId:  D2-AB-Optimal-RMQ */
```

The followings are additional requirements and specs you may assume.

For Sub-Problem D2-A,

- The size of n is at most 10^6, and there will be at most 10^6 queries.
- In other words, most known approaches for RMQ should be able to pass the test.

For Sub-Problem D2-B,

- The size of n is at most 4×10^7, and there will be at most 10^7 queries.
- The allowed time complexities for the two functions warm_up() and query() are $O(n)$ and $O(1)$, respectively.

Comments

You may assume, for example, the source code you submit will be compiled together with the following sample C++ program.

```
#include <stdio.h>

void warm_up(int[], int);
int query(int, int);

int A[3] = { 1, 2, 3 };

int main()
{
warm_up(A,3);
query(0,2);
return 0;
}
```

Note that, the above is just an example. The actual program may vary.

Hint

After some calculations, you may realize that for this problem, it is a better strategy to pick s to be slightly larger than $\log n/4$.

This will better balance the memory usage of sparse table and the table used for RMQ queries for all Cartesian trees.

To be precise, given that $n \leq 4 \times 10^7$, it is recommended to pick $s \approx 9$.

Then the table for storing RMQ answer for all Cartesian trees will have size

$$4^9 \times 9^2 \quad \approx \quad 2.13 \times 10^7 \quad \approx 85\text{MB}.$$

We have approximately $M := 4.45 \times 10^6$ groups with $\log M \approx 23$. Hence the sparse table for the groups has size

$$4.45 \times 10^6 \times 23 \quad \approx \quad 410\text{MB}.$$

Depending on auxiliary arrays you may need to use, the actual memory usage will be roughly around 1GB, which will be fine for this problem.