D0 - StringLength

Time Limit: 1 sec.

Problem Description

In this practice problem, you are asked to implement the `StringLength` function, which takes as input a string stored in a `char` array and returns the length of the string.

Submit the following program to the judge system.

```
 /* ProbId:  D0-StringLength */
#include <string.h>
int StringLength( char str[] )
{
    return strlen(str);
}
```

In order for your submission to be judged correctly, make sure that you adhere the following instructions.

- 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 to this problem.

  ```
  /* ProbId:  D0-StringLength */
  ```

Comments

The goal of this practice problem is to help you become familiar with the "Function only" type submissions.

D1 - Sequence and Cartesian Trees

Time Limit: 1 sec.

Problem Description

In this problem, you are asked to

- Encode the set of all possible Cartesian trees with k vertices into non-negative integers (represented by a binary string) between 0 and 4^k.
- Map the input sequences to the particular Cartesian trees they correspond to.

Input Format

The first line consists of two integers k and m, where k is the size of the Cartesian tree and m is the number of input sequences. Then there are m lines, each containing a sequence of k integers.

You may assume that

- $1 \leq k \leq 2 \times 10^5$.
- $m \times k$ is at most 2×10^5.

Output Format

For each input sequence, compute the encoding of the Cartesian tree it corresponds to and print the binary representation of the encoding in a line. Make sure that the length of the binary string is equal to $2k$.

Note that, the encoding is not unique. You can use any valid way to encode the trees.

Sample Input

```
2 3
1 3
-1 -2
20 30
```

Sample Output

```
0001
0010
0001
```

Note.

This problem is a subroutine for the optimal RMQ algorithm to be used in Problem D2.

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.

D2-C - Optimal Range Minimum Query

Time Limit: 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 index** of the minimum element within `seq[left...right]`. If `[ left, right ]` does not corresponds to a valid range, return `-1` instead.

The indexes of the array follows the standard spec. Hence, they range from 0 to $n - 1$.

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-C-Optimal-RMQ */
  ```

The followings are additional requirements and specs you may assume.

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