CS29003 Algorithms Laboratory

Assignment No: 10

Last date of submission: 10–April–2018

A black box contains an array A of n bits. Most of the bits stored in A are zero. Only k of these bits are one. Let us use zero-based array indexing. There exist indices $0 \leqslant i_0 < i_1 < i_2 < \cdots < i_{k-1} \leqslant n-1$ such that a_{i_j}, $j = 0, 1, 2, \ldots, k-1$, are all the one bits stored in A. Your task is to find out the indices i_j of these one bits.

In order to help you in your search, the black box supports *range queries*. Each range query takes as arguments two integers l, r and a *type*. Here, $l \leqslant r$, so $[l, r]$ stands for an non-empty interval of integers, and *type* is one of the two bit operations: OR and XOR. A range query on (l, r, OR) returns the Boolean OR of the bit values $a_l, a_{l+1}, \ldots, a_r$. Likewise, a range query on (l, r, XOR) returns the Boolean exclusive OR (XOR) of the bits $a_l, a_{l+1}, \ldots, a_r$. The black box imposes a limit L on the number of each queries, where L is slightly larger than $\frac{1}{2} k \log_2 n$. That is, you are allowed to make at most L range queries of the OR type, and at most L range queries of the XOR type.

Write a program to determine the indices $i_0, i_1, \ldots, i_{k-1}$ of the one bits in the array A hidden in the black box, by making appropriate range queries within the specified limit L of the numbers of queries. Today, the algorithm and the data structures would be entirely yours—we provide no helps, hints, instructions, or suggestions for solving today's problem.

In the given black box, you always have $n = 65536$ and $k = 10$.

How to interact with the black box

The black box is presented to you in the form of a precompiled binary file **`blackboxA.o`**. Download the appropriate file depending on your compiler (**`gcc/g++`**). In order to link this binary file, you should compile your program as follows.

```
gcc/g++ -Wall myA10.c/myA10.cpp blackboxA.o
```

Near the beginning of your program, add the following five lines.

```
#define OR 1
#define XOR 0
extern void registerme ( );
extern int rangequery ( int, int, int );
extern void verifysoln ( int [] );
```

The first task you should do in your **`main`** function is to register yourself by calling **`registerme()`** without any arguments. There is no return value from the function. A range query on $(l, r, type)$ is made by calling **`rangequery(l,r,type)`**. It is your duty to ensure that $0 \leqslant l \leqslant r \leqslant n-1$, and **`type`** must be one of **`OR`** and **`XOR`**. When you have prepared an array I of indices storing the indices of the one bits of A, verify the correctness of your computations by calling **`verifysoln(I)`**. Before calling the function, make sure that: (1) I is capable of storing at least $k = 10$ integer values, (2) each $I[j]$, $j = 0, 1, 2, \ldots, 9$, should be an integer value between 0 and $n-1 = 65535$ (both inclusive), and (3) $I[0], I[1], I[2], \ldots, I[9]$ are sorted in strictly ascending order.

Submit a single C/C++ source file. Do not use global/static variables.

Sample output

```
+++ Registering you ... done

+++ Verifying your solutions

--- Your solution   332 is correct
--- Your solution  7121 is correct
--- Your solution 23544 is correct
--- Your solution 28705 is correct
--- Your solution 28713 is correct
--- Your solution 31129 is correct
--- Your solution 38934 is correct
--- Your solution 49219 is correct
--- Your solution 49449 is correct
--- Your solution 62143 is correct

*** You made  84   OR queries
*** You made  75  XOR queries

*** Your point is 10
```