Indian Statistical Institute

DFS LAB – COURSE PROJECT 1

MTech(CS) I year 2019–2020

Deadline: 15 December, 2019

(For BONUS marks)

SUBMISSION INSTRUCTIONS

1. Naming convention for your programs: `cs19xx-project1-progy.c` ('y' stands for the subproblem)
2. To submit a file (say `cs19xx-project1-progy.c`), go to the directory containing the file and run the following command from your account on the server (IP address: 192.168.64.35)

   ```
   cp -p cs19xx-assign1-progy.c ~dfslab/2019/project1/cs19xx/
   ```

   If you want to submit your files from a computer with a different IP address, run

   ```
   scp -p cs19xx-project1-progy.c \
   mtc19xx@www.isical.ac.in:/user1/perm/pdslab/2019/project1/cs19xx/
   ```

   (enter your password when prompted).

   To submit all `.c` and `.h` files at one go, use `cp -p *.c *.h ~dfslab/2019/project1/cs19xx/` or similar.

NOTE: All programs should take the required inputs from stdin, and print the desired outputs to stdout.

Q1. **Fillable Array Problem:** A classic problem related to dynamic data structures is managing the fillable arrays (see Exercise 2.12 in [1]). In this problem, we wish to maintain an array, say $A[1..n]$, with entries in $\{0, \ldots, 2^w - 1\}$ subject to the following three operations:

- `read`(i): returns $A[i]$
- `write`(i, d): $A[i] \leftarrow d$
- `fill`(d): $A[i] \leftarrow d$ for all $i = 1..n$

In the fillable array problem one must maintain an array $A[1..n]$ of w-bit entries such that the aforementioned operations can be performed.

It has recently been shown that with just one bit of redundancy, i.e. a data structure using $nw + 1$-bits of memory, read/fill can be implemented in worst case constant time, and write can be implemented in either amortized constant time (deterministically) or worst case expected constant (randomized) [2]. In the latter case, we need to store an additional $O(\log n)$ random bits to specify a permutation drawn from an $\frac{1}{n^2}$-almost pairwise independent family.

Write a program in C that can implement such an array, having the functions `read(int)`, `write(int, int)`, and `fill(int)`, with the said performance factors. Finally, SUGGEST AND ESTABLISH improvements (any kind of) to this implementation.

Note that, the number of elements in the array is set to be fixed in the paper by Loog et al. [2]. In fact, the array is not growing dynamically. Therefore, the size of integers, i.e., w bits, is not specified here. You can fix it by your choice.

Input Format
The first line of the input contains the initial elements of the array. This is followed by the different operations to be performed on the array. One instruction, for performing an operation, is to be given in each line. The `read(int)` instruction should start with a '=' followed by an integer denoting the array index, `write(int, int)` instruction should start with a '+' followed by a pair of integers denoting the array index and the integer to be written, and `fill(int)` instruction should start with a '@' followed by an integer denoting the integer to be filled in.

Output Format
The output (to be printed to stdout) lines will show the state of the array against each operation, except the `read(int)`, which does not change the array.

Sample Input 0

```
1 2 3 4 5
+ 5 6
= 3
@ 7
+6 8
```

Sample Output 0

```
1 2 3 4 5 6
4
7 7 7 7 7 7
7 7 7 7 7 7 8
```

References

[1] Alfred V. Aho and John E. Hopcroft. The design and analysis of computer algorithms. *Pearson Education India*, First Edition, 1974.

[2] Jacob Teo Por Loong, Jelani Nelson and Huacheng Yu. Fillable arrays with constant time operations and a single bit of redundancy. *arXiv preprint*, arXiv:1709.09574, 2017.