Data Structures and Algorithms for Engineers

Programming Assignment 2: Bit Vectors
Task Description: Read a list of integers from an input file. Generate 0 if the integer has been seen before, and 1 if the integer has not been seen before.
Instructions
Download the code and sample data for this assignment from this location.

Unzip this file to a folder on the VM (or own environment). Organize the code and the sample input into the following locations:
/home/dsa/hw02/code/src/
/home/dsa/hw02/code/bin/
/home/dsa/hw02/sample_inputs/

Note: The input data file will have one integer on each line. The number of lines in an input data file will be <1,000,000.

Required: *Implement code to:*

1) For each input file in the sample folder, you need to output one result file. For example, for input file “sample_input_02.txt”, your result should be in sample_inputs/sample_input_02.txt_results.txt
2) There must be one line in result file for each integer.
   For example, if the input is:

   ```
   1
   4
   1
   5
   5
   1
   4
   -9

   0
   1
   13
   9
   8.7
   ```

   The output should be:

   ```
   1
   1
   0
   1
   0
   0
   0
   1
   1
   0
   1
   1
   ```

3) Your code must also handle following variations in the input file:

a) Integers in each line can have a white space before or after them.
b) If there are any lines with no inputs or white spaces, those lines must be skipped. See example input file “easy_sample_input_02.txt”
c) If there are any lines with two integers separated by white space, those lines must be skipped.
d) If there are any lines that contain a non-integer input, those lines must be skipped. See example input file “easy_sample_input_03.txt”
    - Non-integer input includes alphabets, punctuation marks, non-numeric values, floating point numbers.

How to write your code:

1) Your code needs to be implemented in the file BitVector.cpp and BitVector.h within the following two functions:
   processFile(char * inputFilePath, char * outputFilePath)
   readNextItemFromFile(FILE* inputFileStream)
2) You can create other functions/classes within the files BitVector.h and BitVector.cpp. You **must NOT** create any other source files. These are the only two source files you will submit. We will make a call to the processFile function to grade your work.
3) You are free to use libraries that can enable you to efficiently (pre) process input data. You must include a comment justifying the use of the library. However, you should not use any predefined complex data structures e.g., those found in Standard Templates Library (STL), to represent your data structures.

Note:

There are two broad methods to implement a BitVector data structure.

1) ***Trivial, but inefficient implementation:***
   The minimum and maximum integer value in C++ is given by is INT_MIN and INT_MAX respectively. That means, the min integer can be -2147483648 and the max integer can be 2147483647.
   One can trivially implement a solution for this problem by defining an array having 2 * 2147483648 +1 values. For example: Initialize an array as follows bool *elementsPresent = new bool[2 *2147483648 + 1]. When you are parsing the input file, when an integer “k” is obtained, check the following:
```
if (elementsPresent[k] == 0){
      //This integer is seen for the first time. Output the //value
      "1" to the output file.
      elementsPresent[k] = 1;
}else{
      //This integer is already seen before. Output the value "0" to
      //the output file.
}
```
   If you use this trivial solution, the Boolean vector “elementsPresent” would need to be of size: 2 *2147483648, i.e., 4 MB. While most computers can handle this memory, the problem can be solved using much less memory.
2) ***More efficient implementation:***
   The integers in the input file have been randomly selected. There are less than 1,000,000 integers in each input file. It is not possible that all integers between -2147483648 and 2147483647 are present in the input file. With this constraint, it is possible to solve the problem with a much smaller memory footprint. Here are some alternatives:
   a) *Create a data structure that can simulate an array of bits*. Implement this in a class called BitArray. This will reduce the memory requirement by 4 X. You would now need 1 MB of memory. The BitArray class can have the following (member) functions:

i) BitArray(int numberOfBitsNeeded): This is a constructor that initializes the BitArray to contain " numberOfBitsNeeded". All bits should be set to zero.
ii) void BitArray::setBit(int locationOfBit, bool bitValue): Set the value of the bit at a specific location.
iii) bool BitArray::getBit(int locationOfBit): Get the value of bit at specific location.
iv) References:
    (1) https://web.cs.dal.ca/~jamie/UWO/BitVectors/README.html
    (2) http://www.cs.fsu.edu/~lacher/courses/REVIEWS/cop4531/bitvectors/index.html?$$$toc.html$$$

b) *Use a linked list where each element of the linked list represents a specific range of integers.* For example, say you divide the range of integers into 10240 integers. The first element in the linked list will represent integers -2147483648 to −2147473408. The second element will represent integers from −2147473408 to −2147463168 etc. Each node in the linked list must contain an object of type BitArray. This will give you even more memory saving. If the random numbers are uniformly distributed, the memory saving you obtain could be between 100 X to 1000 X.
c) *Use an array of BitArray's*. For example, divide the range of integers into 102400 integers. The entire space between -2147483648 to −2147473408 will be divided into 41944 parts. Create an array of BitArrays as follows:

```
BitArray currBitArray = new BitArray[41944];
for (int i=0; i < 41944; i++)
      currBitArray[i] = NULL; // This means while you have
      //allotted the memory to store a BitArray
      //object, the actual object has not been created yet.
```

For an input integer, get the index of the BitArray that it belongs to. For example, if the integer is -2147483646, the BitArray index will be:

```
index = floor( ( -2147483646 - (-2147483648)) / 102400))
i.e. index = 0
```

Within the 0-th BitArray, the index for this number will be -2147483646 - (-2147483648 +(102400 * index)) = "2". Set the BitArray value as follows:

```
if (currBitArray[index] == NULL){
      currBitArray[index] = new BitArray(102400);
      currBitArray[index].setBit(2, 1);
      //This integer is seen for the first time. Output the
      //value "1" to the output file.
}
if (currBitArray[index] != NULL){
      if (currBitArray[index].getBit(2) == true)){
            //This integer is already seen before. Output the
            //value "0" to the output file.
      }else{
            currBitArray[index].setBit(2, 1);
            //This integer is seen for the first time. Output
            //the value "1" to the output file.
      }
}
```

Grading method:

1) If your code runs and generates an output file in the correct format for each input file, you get submission points.

2) If your method generates correct results for each test file, you get points for correctness.
3) We will review your source code to examine the internal documentation and award points for proper use of meaningful internal documentation.
4) We will measure the memory consumption in Bytes (submitted_memory). We will take the memory used by our implementation (our_memory). Memory score will be based on the ratio: (our_time /submitted_time) * max score for memory. If your memory usage is less than our method, you get more points. The maximum points you can get here is "max score for memory".
5) We will measure the run time in milliseconds (submitted_time). We will take the time used by our implementation (our_time). Your run-time score will be based on the ratio: (our_time /submitted_time) * max score for run-time. The maximum points you can get here is "max score for run-time".
6) See table below for max score on run-time and memory.
7) *Novelty*: You will submit a one paragraph writeup in PDF format on canvas. We will check if your score for memory is > 6 and time > 8. If your solution meets this limit, we will read your writeup and give you a score for novelty based on your writeup.

Table 1: Marking Rubrics

Item	**Criteria**	**Points Awarded**
Output: Does the code run? Are the output files generated with the filenames and formats as specified in the problem?	Points obtained	
	Max Points	***6***
Correctness: Is the right output generated?	Points obtained	
	Max Points	***18***
Quality internal documentation: Does documentation aid code understandability?	Points obtained	
	Max Points	***6***
Timing	Time taken to run (ms)	
	Comparison with other students (divide by min of others)	
	Max. Timing score	***9***
Memory	Memory consumed (Bytes)	
	Comparison with other students (divide by min of others)	
	Max. Memory score	***9***
Novelty: What was the method used to solve the problem?	*Points obtained*	
	Max Points	***12***
	Total points obtained	
	Total for Task	**60**