Efficient Russian Multiplication
S19 CSCI332 Design and Analysis of Algorithms

Assigned: 2019-02-15
Reference: §4.4; 5.0 Due Date: 2019-03-09 (midnight)

Summary:

Abstract: Create a program implementing the A La Russe multiplication algorithm to multiply a list of numbers stored in an Balanced Search Tree (BST)

Objectives:

1. Decrease and Conquer by a Constant Factor
2. A La Russe Multiplication Algorithm
3. Balanced Search Trees (BST)
4. AVL Self-Balancing Search Trees via Rotations
5. Master Theorem
6. Algorithm Analysis

Grading: 45 pts - A $\geq$ 41.85; A- $\geq$ 40.50; B+ $\geq$ 39.15; B $\geq$ 37.35; B- $\geq$ 36; C+ $\geq$ 34.65; C $\geq$ 32.85; C- $\geq$ 31.75; D+ $\geq$ 30.15; D $\geq$ 28.35; D- $\geq$ 27

Outcomes: R1 (CAC-c,i,j,k; EAC-e,k,1); R2 (CAC-a,b,j; EAC-a,e,1); R6 (CAC-a,c,j; EAC-a,e,1) (see syllabus for description of course outcomes)

Project Description

Both the A La Russe algorithm for multiplying positive integers and the balanced search tree data structure (such as the AVL binary search tree) perform their operations in $\Theta(\log n)$ time efficiency. However, it is not trivial to get either of these to perform with this efficiency unless the input is well understood and managed.

For example, in the case of the A La Russe algorithm, the smallest operand of the multiplication should be the operand used to reduce the problem by a constant factor of two (2). The computed result of the multiplication will be the same if the larger of the two operands is used, but the time efficiency might be considerably worse. Similarly, if an unbalanced binary search tree is used to hold the collection of numbers to be multiplied together by the A La Russe algorithm, the ability to search the tree for the numbers becomes increasingly inefficient as the tree becomes out of balance.

Therefore, the task is to obtain an implementation able to exploit the efficiency of these algorithms to generate a solution to finding the product of a list of positive integers.

The project provides two (2) classes, a `BSTnode` and a `BST` (which is an AVL binary search tree). The `BSTnode` is completely implemented, while the class definition for the `BST` is all that is provided. In addition, two other data structures are provided:

`BSTnode_ptr` Which is an alias (similar to `typedef` in C) for `shared_ptr<BSTnode<ItemType> >`. This simplifies the otherwise repetitive nature of having to type the full type specification without sacrificing clarity.

`TVal` Which is a union type as follows,

```
union
{
   int iVal = -1;
   unsigned long long ullVal;
}
```

which holds the values to be placed in the `BST` tree. Unions allocate memory for the largest union member and allows the user to store and retrieve values through any of the member fields, each of which stores and retrieves from the same memory location. This allows a memory location value to

be used as any of the field types depending on which field is used. Since the values in the `BST` may be used as `int` values as well as `unsigned long long` values (due to overflow of the `int` data type as multiplication is taking place), the union is used to allocate a single block of memory that can hold either of these data types.

The `main` routine of your program will be given a list, L, of $n = |L|$ command line arguments as input, as follows:

$$\forall\ n_i \in L \mid 0 \leq n_i \leq (2^{31} - 1) \text{ and } \prod_{0}^{n-1} n_i \leq (2^{64} - 1)$$

Your output is to be

$$\forall\ n_i \in L \mid \prod_{0}^{n-1} n_i$$

and you will need to also output two (2) counts, one for the number of times the basic operation for the A La Russe algorithm, and one for the number of times the basic operation for the BST tree algorithm. An example would look like the following:

```
Multiplication of the operands below is 67200
25 2 16 7 4 3
Shifts performed is 20 (via A La Russe)
Accesses for operands is 12 (via AVL BST)
```

Obtaining Project Files:

1. Log into your account on the `gitlab.cs.mtech.edu` department server and for the project into your own account.
2. Go to your cloned project in your own account on the GitLab department server and select [settings] and then [members] for the project and add your instructor (and any teaching assistants for the course) as a `Developer` member.
3. (optionally - only need to perform this step once) If you are going to use the `ssh` protocol to obtain your project files from the GitLab department server, you need to make sure the `ssh` key from the machine on which you will be working with the project are copied to the list of valid keys in your account.
4. copy either the `ssh` or `http` url paths to your clipboard
5. Log into the `lumen.mtech.edu` department linux server with your department credentials. If this is the first time you have logged into the server, your username will be the first part of your campus email account and your default password will be your student id; make sure to change your password the first time you login using the `passwd` linux command.
6. Create a projects folder for the course using the command `mkdir -p ~/CSCI332/Projects`, and then change into this directory using the command `cd ~/CSCI332/Projects`.
7. Clone the project to your course project folder using the command `git clone <url>`, where `<url>` is the project url you copies to your clipboard. This will create a new directory for the project.
8. Your should use the command `cd` to change into the new project folder you just cloned.
9. Now proceed to the project activities in the next section.

Project Activities:

1. Build the project by executing the `make` program to create the `main` executable to make sure the files are correct and lead to a completely compiled executable.
2. Review and understand the code that comprises the project. Make sure you understand the syntax associated with creating the header files, with creating the implementation files, with using the templates, with creating the aliases, and with creating the `union` structure.
3. Once you have a good understanding of the project files, process to create new C++ software components to complete the project.
4. Review the `BST.h` header file for the AVL Balanced Search Tree and implement all of the class methods in the `BST.cpp` file.
5. Use the `AlaRusse.h` and `AlaRusse.cpp` files to design and implement the A La Russe algorithm to obtain the $\Theta(\log n)$ efficiency.
6. Modify the `main.cpp` file to instantiate the `BST` used to store the positive integers from the command line, instantiate the `AlaRusse` that will then:
   (a) obtain operands from the `BST`
   (b) perform the shifts needed to obtain the product of the operands
   (c) insert the result into the `BST`
7. Continue until there is only a root node in the `BST`
8. Output the results of the program as shown above.
9. Answer the following questions:
   (a) Provide a closed form equation for the A La Russe algorithm as implemented in your program.
   (b) Provide a closed for equation for the AVL Balanced Search Tree as implemented in your program.
   (c) Discuss if there is a better algorithm for storing the list L of numbers provided on the command line and the strengths and weaknesses over the AVL BST algorithm.
   (d) How would you represent numbers larger than $(2^{64} - 1)$ if multiplication of the list of numbers were to exceed the storage capacity of the C++ `unsigned long long` data type.
10. Place the answers to the above questions in a file named `questions.txt` to be included with your project and will be pushed to the GitLab repository.

Figure 1: Programming Project Grading Rubric

Attribute (pts)	Exceptional (1)	Acceptable (0.8)	Amateur (0.7)	Unsatisfactory (0.6)
Specification (10)	The program works and meets all of the specifications.	The program works and produces correct results and displays them correctly. It also meets most of the other specifications.	The program produces correct results, but does not display them correctly.	The program produces incorrect results.
Readability (10)	The code is exceptionally well organized and very easy to follow.	The code is fairly easy to read.	The code is readable only by someone who knows what it is supposed to be doing.	The code is poorly organized and very difficult to read.
Reusability (10)	The code could be reused as a whole or each routine could be reused.	Most of the code could be reused in other programs.	Some parts of the code could be reused in other programs.	The code is not organized for reusability.
Documentation (10)	The documentation is well written and clearly explains what the code is accomplishing and how.	The documentation consists of embedded comments and some simple header documentation that is somewhat useful in understanding the code.	The documentation is simply comments embedded in the code with some simple header comments separating routines.	The documentation is simply comments embedded in the code and does not help the reader understand the code.
Efficiency (5)	The code is extremely efficient without sacrificing readability and understanding.	The code is fairly efficient without sacrificing readability and understanding.	The code is brute force and unnecessarily long.	The code is huge and appears to be patched together.
Delivery (total)	The program was delivered on-time.	The program was delivered within a week of the due date.	The program was delivered within 2-weeks of the due date.	The code was more than 2-weeks overdue.

The *delivery* attribute weights will be applied to the total score from the other attributes. That is, if a project scored 36 points total for the sum of *specification*, *readability*, *reusability*, *documentation* and *efficiency* attributes, but was turned in within 2-weeks of the due date, the project score would be $36 \cdot 0.7 = 25.2$.

Project Grading:

The project must compile without errors (ideally without warnings) and should not fault upon execution. All errors should be caught if thrown and handled in a rational manner. Grading will follow the *project grading rubric* shown in figure 1.

Collaboration Opportunities:

The is no opportunity to work collaboratively on this project. All work must be original to the student.