CS 5040: Intermediate Data Structures and Algorithms
Virginia Tech
Fall 2020
Dr. Edmison

Programming Assignment 3

Last Updated: 10-13-2020 15:48:43-04:00

Due Friday, Nov 13 @ 11:00 PM ET for 125 points
Due Wednesday, Nov 11 @ 11:00 PM ET for a 10 point bonus
Note: This project also has three intermediate milestones.
See the Piazza forum for details.

Assignment

For this project, you will implement an external sorting algorithm for binary data. The input data file will consist of a 8N blocks of data, where a block is 8,192 bytes. Each block will contain a series of records, where ***each record is 16 bytes long*** and ***contains two fields***. The **first 8-byte field** is a non-negative integer value of type `long` for the record ID and the **second 8-byte field** is the record key of type `double`, which will be used for sorting. Thus, **each block contains 512 records** (8192 bytes/16 bytes per record = 512 records per block).

Your program must sort the the records in the file, *in ascending order of the key values*, using ***replacement selection***, as described in Section 12.6 in OpenDSA in Canvas. Your program will sort sections of the file in a working memory that is **8 blocks long**. To be precise, the heap will be 8 blocks in size; in addition you will also have *a one block input buffer*, *a one-block output buffer* and any additional working variables that you need.

To process the data, read the first 8 blocks of the input file into working memory and use replacement selection to create the longest possible run. As it is being created, the run is output to the one block output buffer. Whenever this output buffer becomes full, it is written to an output file called *the run file*. When the first run is complete, continue on to the next section of the input file, adding the second run to the end of the run file. When the process of creating runs is complete, the run file will contain some number of runs, each run being at least 8 blocks long, with the data sorted within each run. For convenience, you will probably want to begin each run in a new block. You will then **use a *multi-way merge*** to combine the runs into a single sorted file.

You must also use 8 blocks of memory used for the heap in the run-building step to store working data from the runs during the merge step. Multi-way merging is done by reading

the first block from each of the runs currently being merged into your working area, and merging these runs into the one block output buffer. When the output buffer fills up, it is written to another output file. Whenever one of the input blocks is exhausted, read in the next block for that particular run. This step requires random access (using seek) to the run file, and a sequential write of the output file. Depending on the size of all records, you may need multiple passes of multiway-merging to sort the whole file.

Java's `ByteBuffer` class is useful for serializing and deserializing the record.

Invocation and I/O Files

The name of your executable must be `Externalsort`. There will be one input parameter to the program: the name of the record file to read.

The program will be invoked from the command-line as:

```
java Externalsort <record file name>
```

where:

- `Externalsort` is the name of the program. The file where you have your main() method must be called `Externalsort.java`
- `<record file name>` is the name of the file with the records to be sorted. At the end of your program, this record file (on disk) should be sorted. So this program does modify the input data file. Be careful to keep a copy of the original when you do your testing. You may assume that the specified record file does exist in our test cases.

In addition to sorting the data file, you must report some information about the execution of your program. You will need to report part of the sorted data file to standard output. Specifically, your program will print, to standard output, the first record from each 8192-byte block, in order, from the sorted data file. The records are to be printed 5 records to a line (showing both the key value and the id value for each record), the values separated by whitespace and formatted into columns. This program output must appear EXACTLY as described; ANY deviation from this requirement may result in a significant deduction in points. See the provided expected output example file to see how your output should be formatted.

You are also provided a standalone java program called GenFile.java that will generate input files with randomized data for you to use in your testing .

The GenFile program can be invoked from the command-line as:

`java GenFile <filename> <size in blocks>` where:

- `GenFile` is the name of the program.

- `<filename>` is the name of the file where the randomized test records will be stored.
- `<size in blocks>` indicates the number of blocks of data to generate for the file.

Programming Standards

You must conform to good programming/documentation standards. Web-CAT will provide feedback on its evaluation of your coding style, and be used for style grading. Beyond meeting Web-CAT 's checkstyle requirements, here are some additional requirements regarding programming standards.

- You should include a comment explaining the purpose of every variable or named constant you use in your program.
- You should use meaningful identifier names that suggest the meaning or purpose of the constant, variable, function, etc. Use a consistent convention for how identifier names appear, such as "camel casing".
- Always use named constants or enumerated types instead of literal constants in the code.
- Source files should be under 600 lines.
- There should be a single class in each source file. You can make an exception for small inner classes (less than 100 lines including comments) if the total file length is less than 600 lines.

We can't help you with your code unless we can understand it. Therefore, you should no bring your code to the GTAs or the instructors for debugging help unless it is properly documented and exhibits good programming style. Be sure to begin your internal documentation right from the start.

You may only use code you have written, either specifically for this project or for earlier programs, or code provided by the instructor. Note that the OpenDSA code is not designed for the specific purpose of this assignment, and is therefore likely to require modification. It may, however, provide a useful starting point.

Allowed Java Data Structure Classes

You are not permitted to use Java classes that implement complex data structures, or that handle the merging of multiple files. This includes `HashMap, Vector`, or any other classes that implement lists, hash tables or the like. You may use the standard array operators. You may use typical classes for string processing, byte array manipulation, parsing, etc.

If in doubt about which classes are permitted and which are not, you should ask. There will be penalties for using classes that are considered off limits.

Deliverables

You should implement your project using Eclipse, and you should submit your project using the Eclipse plugin to Web-CAT. Links to the Web-CAT client are posted at the class website. If you make multiple submissions, ***only your last submission will be evaluated***. There is no limit to the number of submissions that you may make.

You are required to submit your own test cases with your program, and part of your grade will be determined by how well your test cases test your program, as defined by Web-CAT's evaluation of code coverage. Of course, your program must pass your own test cases. Part of your grade will also be determined by test cases that are provided by the graders. Web-CAT will report to you which test files have passed correctly, and which have not. Note that you will not be given a copy of these test files, only a brief description of what each accomplished in order to guide your own testing process in case you did not pass one of our tests.

When structuring the source files of your project, use a flat directory structure; that is, your source files will all be contained in the project `src` directory. Any subdirectories in the project will be ignored.

You are permitted to work with a partner on this project. While the partner need not be the same as who you worked with on any other projects this semester, you may only work with a single partner during the course of a project unless you get special permission from the course instructor. When you work with a partner, then ***only one member of the pair will make a submission***. Make sure that you declare your partner in Web-CAT when submitting. Be sure both names are included in the documentation. Whatever is the final submission from either of the pair members is what we will grade unless you arrange otherwise with the GTA.

Honor Code Pledge

Your project submission must include this statement, pledging your conformance to the Honor Code requirements for this course. Specifically, you must include the following pledge statement ***near the beginning of the file containing the function main()*** in your program. The text of the pledge must include:

```
// On my honor:
// - I have not used source code obtained from another student,
//   or any other unauthorized source, either modified or
//   unmodified.
//
// - All source code and documentation used in my program is
//   either my original work, or was derived by me from the
//   source code published in the textbook for this course.
//
// - I have not discussed coding details about this project
//   with anyone other than my partner (in the case of a joint
//   submission), instructor, ACM/UPE tutors or the TAs assigned
//   to this course. I understand that I may discuss the concepts
//   of this program with other students, and that another student
//   may help me debug my program so long as neither of us writes
//   anything during the discussion or modifies any computer file
//   during the discussion. I have violated neither the spirit nor
//   letter of this restriction.
```

Programs that do not contain this pledge will not be graded.