Environment

OS: Windows 8 Pro Desktop

Java: JDK version 1.6.0 38

IDE: Eclipse Juno 4.2

Ant: 1.8 2

Note: Please ensure the ant and java bin locations are added to the local PATH variable in order to compile and run the project from the command line.

How To Build

- Extract the attached zip DuplicateNumberMatcher.zip to any given folder.
- CD to the project folder DuplicateNumberMatcher once extracted.
- To clean the project and class files from the prompt type:
  ant clean
- To compile and build the project jar file from the prompt type:
  ant release

How To Run Release

- To run the project a provided dos bat file is included in the base extracted folder. It can be run in 2 modes. The first one to generate a random file of numbers for processing the second will take as an argument the numbers file to process.
  - **run.bat**
  - **run.bat numbers.txt**

How To Run Test Scenario

- To run the project test cases it can be run using ant or run using the provided run.bat dos bat file.
  - **ant runtest**
  - **run.bat runtest**

Design Decisions

The design approach taken is to use an external sorting algorithm as opposed to internal in memory sorting over the full list of numbers since we need to consider the limit on memory and processing.

1. Make a calculation based on the amount of memory available and amount of max files we will allow to be created to determine the correct size in kb each file we will create. At the moment these are fixed at 2mb of memory and 512 files max.
2. Read the input file line by line taking each number and storing that number in a list.
3. Once we reach the max size we set for each file take the current read numbers in the collection list and sort them from smallest to largest number.
4. Write this new sorted list to a new temp file and clear the memory list contents once complete.
5. Repeat steps 2-4 until the input file is completely read. Once this is complete we will have a series of output temp files where each file's numbers are sorted.
6. We now need to merge the files back again so they are fully sorted into one temp file.
7. Using a priority queue keep a sorted list of the files to be read in order as we add and remove the numbers to read from each file. The sorting algorithm on the queue is based on the first number to be read from each file. Since it will be sorted as we read from the queue the next file to read it always have its number to read in order of least to most.
8. As we read each number from the queue write that number out to a temp file. Once this is complete we will have a sorted list of numbers that were in the original file.
9. Final step is to read the temp file and check for duplications on the current and next number to be read from the file. Since the file numbers are sorted duplicates will be present in sequence (i.e. 1,2,3,3,3,4,5,6,6,6 etc…)

Test Plan Results

The output from the test plan when run produces 4 different test scenario results.

- Runs a test against a small set of test data to check the count of duplicates found matches the count of duplicates added to the test data.

  Test Case A (Correct array size, small number set): Passed

- Runs a test against a small set of test data to check each of the duplicates found matches the duplicate numbers added to the test data.

  Test Case B (Correct numbers present, small number set): Passed

- Runs a test against a large set of test data to check the count of duplicates found matches the count of duplicates added to the test data.

  Test Case C (Correct array size, large number set): Passed

- Runs a test against a large set of test data to check each of the duplicates found matches the duplicate numbers added to the test data.

  Test Case D (Correct numbers present, large number set): Passed

Performance Assessment

The performance of the solution when running should be efficient to work within the given memory constraints especially when storing each subset of numbers in memory and writing back to disk. At this stage is the most when memory is at its capacity. Special consideration should be given to the speed and access to disk when reading and writing data from the temp sorted files especially when these files are accessed and processed from the priority queue.

Another solution to using a priority queue is to read a small subset of numbers from each sorted file into memory (i.e if we have 2mb files and 5 files then read about 400k of data from each of the 5 files which equals our 2mb limit). Once we read this data sort the input and write out a batch of data from the top of the list, maybe 200k of data). Read then 200k more data across the 5 files and repeat the sorting and writing out a small subset again. This will ensure the sorted numbers will make their way up the list from least to most and be written out in that fashion. It may not be an ideal solution if there is a large set of duplicates present in the input file. We may not reach the top of the list quick enough to be in order as we are reading from the split files.