Programming Assignment 5

CSCD300 Data Structure

Instructor: Dr. Bojian Xu
Eastern Washington University, Cheney, Washington

Due: 11:59pm, July 20, 2014 (Sunday)

Please follow these rules strictly:

1. Verbal discussions with classmates are encouraged, but each student must independently write his/her own work, without referring to anybody else's solution.
2. No one should give his/her code to anyone else.
3. The deadline is sharp. Late submissions will **NOT** be accepted (it is set on the Canvas system). Send in whatever you have by the deadline.
4. Every source code file must have the author's name on the top.
5. All source code should be commented reasonably well.
6. Sharing any content of this assignment and its keys in any way with anyone who is not in this class of this quarter is NOT permitted.

Simulating the merging process of external sorting

External sorting [1] is to sort a data set which is too massive to fit into the random access main memory, forcing normal RAM model-based sorting algorithms [2] to fail. The basic idea of external sorting is to cut the data set into smaller enough subsets and then sort each subset using a RAM-based sorting algorithm, then merge all the subsets into a single large sorted data set. The efficiency of the merging process is critical and largely determines the overall efficiency of the external sorting. In this assignment, we want to simulate the merging process by using the FIFO queue data structure, including **both the array-based and the singly linked list-based implementations**. [3]

[1] `http://en.wikipedia.org/wiki/External_sorting`

[2] Such as the *selection sort* that you have learned before as well as several others that we will discuss later in this course.

[3] The merging process described in this assignment is trivial and slow. There exists an order of magnitudes faster method for this merging process by using the priority queue data structure, which we will be discussed in CSCD320.

Specification of your program

1. Your program should be named as: `Test_Merge.java`
2. The input and output of your program.

   The input are a number of files, each of which contains a sequence of integers, one integer per line, in non-descending order. **Your program should be working for any number (≥ 2) of input files**.

   The output is the screen print of the whole data set from merging all the integers in the input files in the non-descending order.

An example

Suppose we supply to your program with three files named `data1.txt`, `data2.txt`, and `data3.txt`, with the following content:

```
4          1          2
6          3          5
7          5          9
9          6
           8

data1.txt      data2.txt      data3.txt
```

Then, the command line you should type would be:

```
$java Test_Merge data1.txt data2.txt data3.txt
```

Then your program will merge all the integers from the three sorted files and print them on the screen:

```
1
2
3
4
5
5
6
6
7
8
9
9
```

The merging process and its implementation

Suppose we are given k files $F_1, F_2, \ldots, F_k$, for some $k \geq 2$. Each contains a sequence of sorted integers, one per line in non-descending order. The number of integers in all these files do NOT have to be the same. We want to merge the integers from all these files, so that all the integers after merging are in non-descending order. The following pseudocode shows a trivial way for doing this merging process and you are required to implement it.

Note: The capacity of the array-based FIFO queue in the code below **must be set to be 10** in order to give yourself the chance to play the enqueue and dequeue operations alternatively during the run of your program.

Merge($F_1, F_2, \ldots, F_k$)

Input: k text files. Each contains a sequence of integers, one per line in non-descending order.
Output: The print of all the integers from the k input files in an non-descending order.

for $i = 1, 2, \ldots, k$ **do**
    Initialize an empty **array-based** FIFO queue of **capacity 10**: $ArrayQ_i$
    Enqueue the first 10 integers from F_i into $ArrayQ_i$. If F_i contains less than 10 integers, all the integers in F_i will be enqueued into $ArrayQ_i$. `/* So all elements in` $ArrayQ_i$ `are in non-descending order from the head to the tail. */`

Initialize an empty **singly linked list-based** FIFO queue: $ListQ$

while $ArrayQ_1, ArrayQ_2, \ldots, ArrayQ_k$ *are not all empty* **do**
    Let $ArrayQ_m$ be the queue whose head number is the smallest among all the $ArrayQ_i$'s that are not empty, for some $m \in \{1, 2, \ldots, k\}$. `/* If multiple queues share the same smallest head number, it is fine to pick anyone. */`
    $ListQ$.enqueue($ArrayQ_m$.dequeue())
    $ArrayQ_m$.enqueue(the next number from F_m). `/* Do nothing if all the numbers in` F_m `have been enqueued. */`

while $ListQ$ *is not empty* **do**
    print $ListQ$.dequeue()

Hint: For a better code organization, you may want to organize all those array-based FIFO queues into another array, where every element of that array is an array-based FIFO queue. For example, if there are 10 input files, we will allocate an array $ArrayQ[0 \ldots 9]$ of size 10, where each array element $ArrayQ[i]$ itself is an array-based FIFO queue.

Submission

- All your work files must be saved in one folder, named: **firstname_lastname_EWUID_cscd300_prog5**
  (1) We use the underline '_' not the dash '-'.
  (2) All letters are in the lower case including your name's initial letters.
  (3) If you have middle name(s), you don't have to put them into the submission's filename.
  (4) If your name contains the dash symbol '-', you can keep them.
- You need to include a pure ascii text file in the above folder, which contains the description of your implementation of the FIFO queues and how to use them for merging all the sorted individual files together into a sorted data set.
- You then compress the above whole folder into a .zip file.
- Submit .zip file onto the Canvas system by the deadline.