CS 351 Data Organization and Management

FALL 2011 - Programming Project #2

External Sorting

Part 1 – Creating Sorted Segments (Dec. 4, 2011)

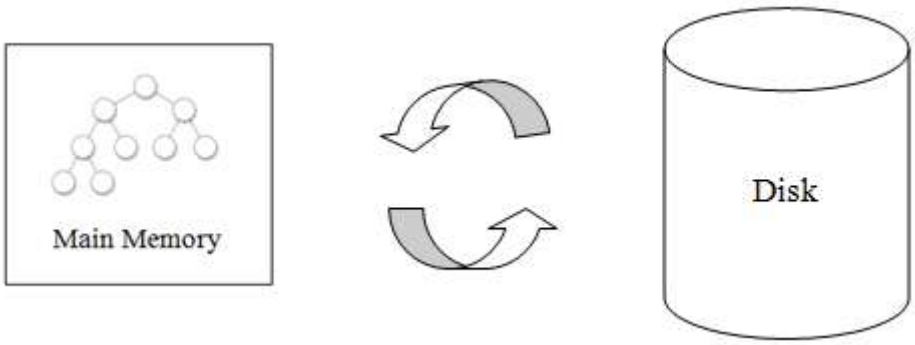
Due: December 19, 2011 at 11:59 PM

Sorting a collection of records on some (search) key is a very useful operation. Sorting is important because users may want answers in some order, or we want to eliminate duplicate copies in a collection of records.

Although main memory sizes are growing rapidly, we may have files larger than the memory. When the data to be sorted is too large to fit into available memory, we need an external sorting algorithm.

The idea behind external sorting is as follows: We break the entire file into smaller segments, sorting these segments, writing them to disk and merging them using some amount of memory.

In this project you are asked to implement a Java console application which aims to create the sorted segments of students file. You must think the memory as min-heap of certain size, and read records from the file as long as the heap is not full. After filling the heap, now you must create corresponding sorted segment through the heap, and write it to the output file. You must continue this procedure as long as the end of file is not reached.



1. File Structure

Students file consists of student records which have the following format:

Student ID	Student Name	Student Surname	Department

Each record is uniquely identified by its student id field. Therefore, all records are unique (No duplicate records exist).

A record as a whole can be read into a string (i.e. you can read records line by line from the input file).

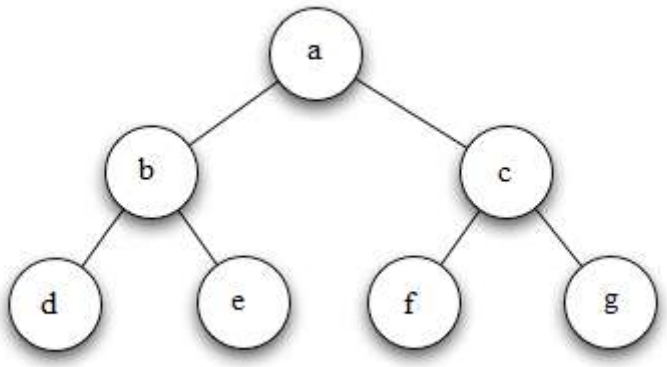
You are supposed to do read/write file operations in Java as explained in the tutorial section.

You can find the students file in the link given below:

http://cs.bilkent.edu.tr/~ctoraman/cs351fall11/project2/students.txt

2. Memory

Memory size allocated to this program is limited (as in the discussion of Salzberg's book). You must think the limited memory as a complete binary min-heap. You must also think the heap as an array (you are supposed to sort in ascending order so the root contains the minimum element). Consider the following example:



Heap Array:

a	b	c	d	e	f	g

Heap has the following properties:

- Complete binary tree property: Complete binary tree is a binary tree in which every level, except possibly the last, is completely filled. The leaves can only appear in the last two levels.

- Min heap property: The key value of each node must be smaller than its children. For the heap given above, the following heap properties must be satisfied:

For the first node: a < b and a < c
For the second node: b < d and b < e

For the third node: c < f and c < g

Note that each node in the heap holds a student record and the size of the heap is 100 student records. You must take heap size as constant in your code.

3. Creating Sorted Segments

While creating sorted segments, you must use the following algorithms:

Algorithm #1. CREATING_SORTED_SEGMENTS
1. Open students file F for reading 2. numSortedSegments ← ⌈total number of records in F / heap size in terms of records⌉ 3. numSortedSegmentsCreated ← 0 4. **while** numSortedSegmentsCreated < numSortedSegments 5. CONSTRUCT_HEAP 6. CONSTRUCT_SORTED_SEGMENT 7. numSortedSegmentsCreated ← numSortedSegmentsCreated + 1 8. **endwhile**

3.1. Constructing Heap

Algorithm #2. CONSTRUCT_HEAP
2. heapIndex ← 0 3. **while** heapIndex < HEAP_SIZE 4. studentRecord ← Read the next record from students file (students.txt) 5. **if** studentRecord is not NULL 6. heap[heapIndex] ← studentRecord 8. Apply heap condition test to continue with a valid heap 9. heapIndex ← heapIndex + 1 10. **else** 11. end of students file reached so exit the while loop 12. **endwhile**

3.2. Constructing Sorted Segment

Algorithm #3. CONSTRUCT_SORTED_SEGMENT
1. last ← HEAP_SIZE - 1 1. **while** last is not equal to 0 2. write heap[0] to the sortedSegments output file (sortedSegments.txt) 3. heap[0] ← heap[last] 4. heap[last] ← NULL 5. last ← last – 1 6. Apply heap condition test to continue with a valid heap 7. **endwhile** 8. Write heap[0] to the sortedSegments output file (sortedSegments.txt)

4. Tutorial: How to R/W File in Java

In Java, there are various ways to handle r/w a file; however, you must use the following way. Other implementations will **not** be accepted.

You have to use BufferedReader and BufferedWriter classes to read from and write to a file, respectively. Consider using the following methods of BufferedReader:

- BufferedReader(Reader in): Opens file for reading. The argument is an instance of Reader class. See Java API for more detailed information.
- readLine( ): Reads a line from the file.
- close( ): Closes the file.

Use the following methods of BufferedWriter:

- BufferedWriter(Writer out): Opens file for writing. The argument is an instance of Writer class. See Java API for details.
- write(String s): Writes a string to the file.
- close( ): Closes the file.

Note that in the sortedSegments.txt, separate each sorted segment by using an empty line between them. In other words, all sorted segments are stored in one file.

5. Submission Rules

- Name your program as **ExternalSortingPart1.java**, and this will be run for program execution.

- You are free how to test your implementation. We will use our own method to grade your projects.

- The input files will always be **students.txt**. The output file which your program creates at the end of execution should be named as **sortedSegments.txt**. The output file should be placed in the same folder that your Java project folder exists.

- Note that we may test your program with a different input file. Additionally, we plan to use a supportive tool to be able to detect plagiarism.

- Put your java file(s) into a rar file. Do not put any other file.

- Name your rar file as **StudentID_cs351p2.rar** (e.g. 20802629_cs351p2.rar). Files with other formats will be **ignored** automatically.

- Use the submission page to upload your rar file:
  http://cs.bilkent.edu.tr/~ctoraman/cs351fall11/project2/submission/
  Read the upload rules in the submission page as well. Late submissions will **not** be accepted!

- You can resubmit until the due date and the previous file will be replaced.

- You must follow the submission rules.

- For questions, contact with koroglu@cs.bilkent.edu.tr