External Memory Sorting

Data Structures and Algorithms Lab
Assignment 5

Goal of this assignment is to implement an external file sort template to sort large collection of records that do not fit into main memory. The records are stored in a file. We will use k-way merge sort for sorting.

Outline of the approach

The input is a sequence of records stored in a file, say "inp.txt", where each record is stored in a separate line. The sorting has two phases, namely, splitting phase and merging phase.

Splitting Phase: In splitting phase, the input file is scanned sequentially and m records (corresponding to m lines in the file), where m is an input parameter, are read into main memory and stored in an array of size m. These m records are sorted in memory using Randomized quick sort RQSORT, which does in-place sorting. Recall that in RQSORT the pivot is chosen uniformly at random. After sorting, the sorted array of m records are written to a new file. Above procedure is repeated till the whole input file is scanned. Note that if the input file contains n records then splitting phase creates $\lceil n/m \rceil$ files each containing m records (except possibly the last file).

Merging Phase: In merging phase, the output files of the splitting phase are merged using Merge sort as follows. Let Q be a queue of files that initially contains all files created by splitting phase. The files in Q are merged k at a time (or less than k if Q has less than k items), where k is an input parameter, using k-way merge. In k-way merge, the head record of each of the k files is read and the minimum is output. The head of the file corresponding to the minimum advances to the next record. The merge finishes when all k files are scanned completely. To find the minimum of k head records efficiently (in $O(\log k)$ steps), a binary heap data structure of size k is used. Each merge of k files produce one new file and this file is added to the end of Q. The k-way merge process is repeated till Q contains only one file, in which case, that file is the sorted output.

Implementation Details

Name the sorting template class as 'FileSort' (Class FileSort < T >). Here 'T' denotes the user defined class for processing each record in the input file. That is, to process a record in the file (stored in a separate line) in both splitting and merging phase, the record is first stored in memory as an object of class 'T'. These objects are then compared against each other in k-way merge and RQSORT. Class 'T' is assumed to have the following operators overloaded:

1. Input stream operator >> : This operator when invoked as say "ifstream >> rec", where 'ifstream' is an input stream object (for example a file input stream) and 'rec' is an instance of type 'T', reads the next line from the stream and assigns each field of 'rec' the corresponding value from the line.
2. Output stream operator << : To output the content of the object as a string.
3. Comparison operators $<, >$ and $==$.

The FileSort template class should contain the following:

1. FileSort(String InputFileName, String OutputFileName, int m, int k) : Constructor that takes parameters m and k (described above) in addition to the input and output file names.

2. boolean Sort() : Sorting function. Return 'true' if sort was successful and 'false' in case of any error.

Splitting phase: Implement recursive Quick Sort "void RQSORT(T[ ] A, int start, int end)", where A is the array (of size m) containing the records to be sorted stored at locations A[start],...,A[end]. The sorted output will also be present in the same array segment. Define an "file input stream", say ifs, over the input file. The input array 'A' for RQSORT is filled with m records by reading each record from input file as "ifs >> A[i]". Files for storing each sorted output (of size m) are named "1", "2", "3" etc., by maintaining a file name counter, which is incremented every time a new block of m input records are sorted.

Merging Phase

Binary Heap Class 'BHeap': Implement a binary heap (Min heap) class 'BHeap' for using in k-way merge. Each heap item is an object of HeapElem class, which is a pair of elements (rec, file index), where 'rec' is an object of type T and 'file index' is an integer (in our case, a number between 1 and k) that denotes the index of the input stream (among the k input streams that are merged) to which 'rec' belongs. In BHeap, the heap items are compared based on their 'rec' fields. The BHeap Class should contain the following:

1. BHeap(int hsize) : Constructor with parameter 'hsize' denoting the maximum size of the heap. The binary heap array is allocated with size hsize.

2. Insert (HeapElem * e)

3. HeapElem ExtractMin()

KMerge Class: Implement 'KMerge' Class for using in the k-way merging. KMerge uses an instance of BHeap class (with size initialized to k). The KMerge class should support the following:

1. KMerge(int k) : Constructor with k as the parameter. This in turn creates an instance of BHeap with size k.

2. AddToMerge(T * recp, int file_index) : Include the record pointed by 'recp' for merging. The parameter file_index denotes the index (number between 1 and k) of the file to which 'rec' belongs.

3. int getNextMin(T ** recpp) : To get the next element of k-merge. This is a call by reference function. The next element will be stored in the pointer, which is pointer to by recpp (Note that recpp is a pointer to pointer). The return int denotes the file_index associated with the returned record.

Merge Loop: Let the files created by splitting phase be "1", "2", ..., "r", where $r = \lceil n/m \rceil$. The files "1", "2", ..., "k" are first merged as follows. The output of this merge is place in file "$r+1$". Create an array of k file input streams, one for each file "1", "2", ..., "k". Let KM be an instance of KMerge class. Initially add to KM the head elements of these k files using AddToMerge(). The index parameter of AddToMerge correspond to the array index of the fileinputstream from which head element was fetched. Now iteratively remove minimum element from KM using getNextMin

and write to the output file. After removing an item from KM, new element is added to KM from the filestream stored at array index returned by the latest getNextMin. Once the merge finishes, that is, when all fileinputstreams have reached end of file, remove files "1", "2", ..., "k" from the file system. Now repeat the above for the next k files "$k+1$", "$k+2$", ..., "$2k$", whose merged output is written to file "$r+2$", and so on. Thus, merging of a set of k files contribute a new file into the queue. The merge loop finishes when only one file remains. Rename this file to the output file name specified in the FileSort constructor.

Note:

- Implementation of additional ideas to further improve space and time performance, usability etc. is strongly encouraged.
- Adequate error messages should be printed during error handling.