Design and implementation of a software for visualizing the external multipass sorting algorithm

Andrea Morelli 1845525
Lorenzo Romagnoli 1975517

Data Management project
Sapienza University of Rome

External sorting

External sorting is a class of sorting algorithms that can handle massive amounts of data. External sorting is required when the data being sorted do not fit into the main memory of a computing device and instead, they must reside in the slower external memory, usually a disk drive. Thus, external sorting algorithms are external memory algorithms and thus applicable in the external memory model of computation.

External sorting algorithms generally fall into two types, distribution sorting, which resembles quicksort, and external merge sort

The external memory model

The external memory model is an abstract machine similar to the RAM machine model, but with a cache in addition to main memory. The model captures the fact that read and write operations are much faster in a cache than in main memory, and that reading long contiguous blocks is faster than reading randomly using a disk read-and-write head. The running time of an algorithm in the external memory model is defined by the number of reads and writes to memory required.

The model consists of a processor with an internal memory or cache of size M, connected to an unbounded external memory. Both the internal and external memory are divided into blocks of size B. One input/output or memory transfer operation consists of moving a block of B contiguous elements from external to internal memory, and the running time of an algorithm is determined by the number of these input/output operations.

Multipass merge-sort algorithm

Merge-sort algorithm for external memory uses the sort and merge strategy to sort the huge data file on external memory. It sorts chunks that fit in main memory and then merges the sorted chunks into a single larger file. Thus it can be divided into 2 phases – Run formation Phase and Merging Phase.

In the following sections we will refer to the following Parameters:

- B – Number of pages of the relation R
- F – Number of buffer frames available
- S – Number of runs created at run formation phase, so ($S = B/F$)
- Z – Number of buffer frames used in the merging phase (usually $Z = F\text{-}1$)

Multipass merge-sort, the idea behind the algorithm

Assuming that we have F frames in the buffer the basic idea behind the algorithm is the following:

- Pass 0 : read the file in runs of F pages, and, at each run, internally sort the F pages, and write a file (called run) to disk
  - ❑ In all subsequent passes, we reserve F-1 input frames for input, and 1 frame for output
- Pass 1 : merge F-1 runs into one output runs
  - ❑ For each input run, read one page in one dedicated frame
  - ❑ When a page is used up, read the next page of the input run into the same frame
  - ❑ The result of merge is buffered in the output frame: when such frame is full, its content is written in the output run in secondary storage
- Pass n : for all the next passes we compute the same actions as the Pass 1 until we have a single sorted run that is the resulting one.
- Sorted!

Multipass merge-sort algorithm

Run formation phase

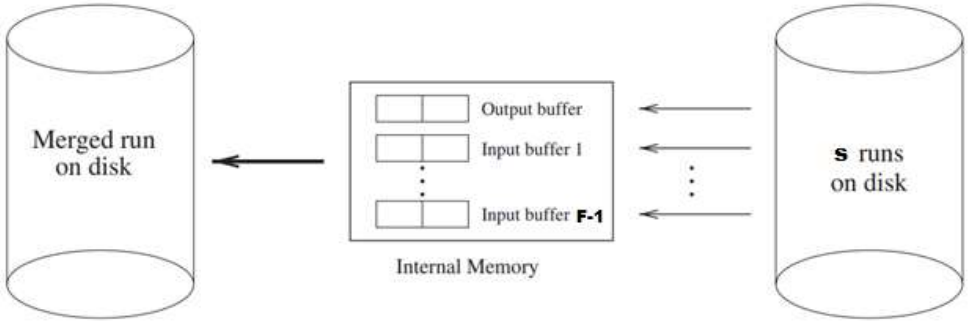
B pages of data are scanned, one memory load at a time. Every time we load F (given F equal to the number of frames available) pages in the buffers, and we sort the elements in those pages into a single run, that is written in secondary storage. At the end of this run formation phase of the algorithm (also known as Pass 0), there are S number of sorted runs, given $S = B/F$. Those S runs will be then given as input to the merging phase of the algorithm.

The run formation phase which involves creation S sorted lists takes place in $O(2 * B)$ I/O operations.

Multipass merge-sort algorithm

Merging phase

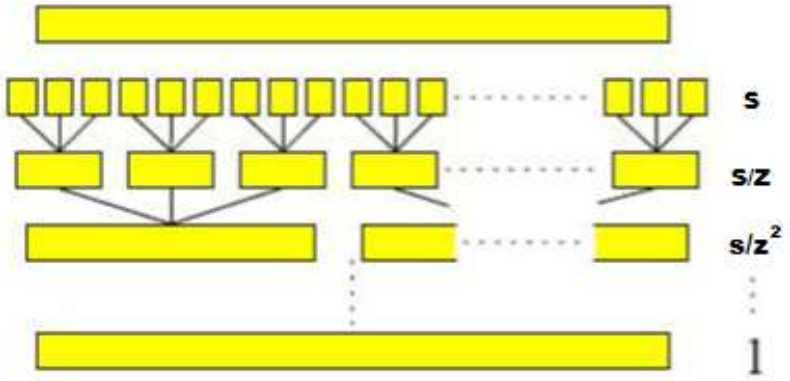
After initial runs are formed, the merging phase begins where groups of Z runs are merged. For each merge, the Z runs are scanned and merged in an online manner as they stream through the internal memory. The number of passes necessary to reach the end of the algorithm is equal to $\boldsymbol{K = log_Z S}$: then the number of runs created in each pass is S/Z^K



Multipass merge-sort algorithm

Merging phase

During the merging phase the S runs are merged together repeatedly. As seen in the I/O complexity diagram, it forms a recursion tree with S elements at leaves and height of the tree equal to $\boldsymbol{K = log_Z S}$. The algorithm stops when we reach just one final sorted list at the end, therefore $1 = S/Z^K$ and it follows that $\boldsymbol{K = log_Z S}$.



Multipass merge-sort algorithm
Overall complexity

At every pass including pass 0 we read and write every page of the relation therefore the cost in terms of page accesses is equal to $\boldsymbol{2 * B * (log_Z S + 1)}$ that is equal to $\boldsymbol{2 * B * \left(log_Z \frac{B}{F} + 1\right)}$

Since usually $Z = F - 1$ we have: $\boldsymbol{log_Z \frac{B}{F} = log_{F-1} B - log_{F-1} F}$.

Then we can approximate the overall complexity to $O\left(\boldsymbol{B * (log_{F-1} B)}\right)$

Now we will show the running software!

Thank you for the attention!