10 COSEQUENTIAL PROCESSING (SORTING LARGE FILES)

Content

- Cosequential Processing and Multiway Merge
- Sorting Large Files (External Sorting)

Cosequential Processing & Multiway Merging

- K-way merge algorithm: merge K sorted input lists to create a single sorted output list
- Adapting 2-way merge algorithm
  - Instead of naming as List1 and List2 keep an array of lists: List[1], List[2],..., List[*K*]
  - Instead of naming as item(1) and item(2) keep an array of items: item[1], item[2],..., item[*K*]

2-way Merging Eliminating Repetitions

Synchronization

- Let item[1] be the current item from list[1] and item[2] be the current item from list[2].
- ***Rules:***
  - If item[1] < item[2], get the next item from list[1].
  - If item[1] > item[2], get the next item from list[2].
  - If item[1] = item[2], output the item and get the next items from the two lists.

K-way Merging Algorithm

- An array of *K* index values corresponding to the current element in each of the *K* lists, respectively.
- Main loop of the K-Way Merge algorithm:
  1. *minItem*=index of minimum item in item[1],item[2],...,item[*K*]
  2. output item[*minItem*] to output list
  3. for i=1 to *K* do
  4. if item[i]=item[*minItem*] then
  5. get next item from List[i]
- If there are no repeated items among different lists, lines (3)-(5) can be simplified to

  get next item from List[*minItem*]

Implementation # 1

- The K-Way Merging Algorithm just described works well if $K<8$:
- Line(1) does a sequential search on item[1], item[2], ..., item[*K*]

  Running time: $O(K)$
- Line(5) just replaces item[i] with newly read item

  Running time: $O(1)$

Implementation # 2

- When the number of lists is large, store current items item[1], item[2], ..., item[K] into priority queue (heap).
- Line(1) does a min operation on the heap.

  Running time: $O(1)$
- Line(5) performs a **extract-min** operation on the heap:

  Running time: $O(\log_2 K)$
- and an **insert** on the heap

  Running time: $O(\log_2 K)$

Detailed Analysis of Both Algorithm

- Let N = Number of items in output list

  M = Number of items summing up all input lists

  (Note N≤M because of possible repetitions)
- Implementation # 1
  - Line(1): $K \times N$ steps
  - Line(5): counting all executions: $M \times 1$ steps
  - Total time: $O(K \times N + M) \subseteq O(K \times N)$
- Implementation # 2
  - Line(1): $1 \times N$ steps
  - Line(5): counting all executions: $M \times 2 \times \log_2 K$ steps
  - Total time: $O(N + M \times \log_2 K) \subseteq O(M \times \log_2 K)$

Merging as a Way of Sorting Large Files

- Characteristics of the file to be sorted
  - 8,000,000 records
  - Size of a record = 100 Bytes
  - Size of the key = 10 Bytes
- Memory available as a work area: 10 MB (Not counting memory used to hold program, OS, I/O buffers, etc.)
  - Total file size = 800 MB
  - Total number of bytes for all the keys = 80 MB
- So, we cannot do internal sorting

Solution

- Forming runs: bring as many records as possible to main memory, do internal sorting and save it into a small file. Repeat this procedure until we have read all the records from the original file
- Do a multiway merge of the sorted files
- In our example, what could be the size of a run?

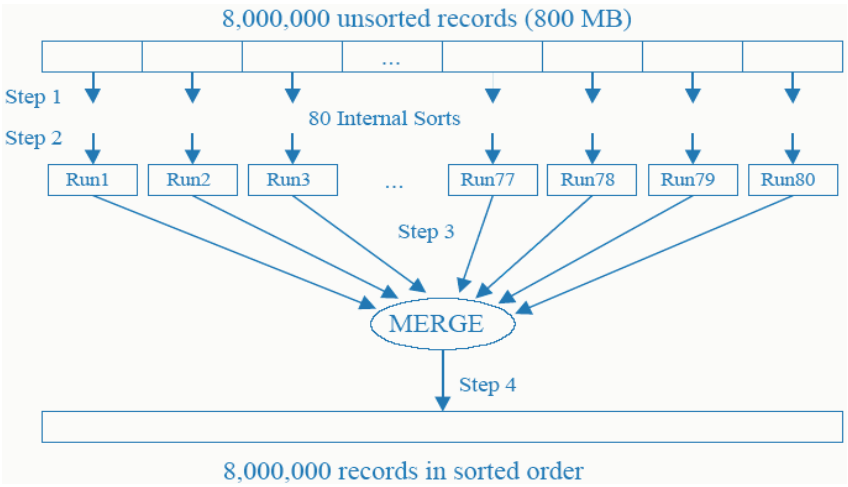
Available memory = 10 MB $\cong$ 10,000,000 bytes

Record size = 100 bytes

Number of records that can fit into available memory = 100,000 records

Number of runs = 80 runs

80 Internal Sorts



Order of I/O Operations

- I/O operations are performed in the following times:

1. Reading each record into main memory for sorting and forming the runs
2. Writing sorted runs to disk

- The two steps above are done as follows:
  - Read a chunk of 10 MB; Write a chunk of 10 MB (Repeat this 80 times)
  - In terms of basic disk operations, we spend:
  - For reading: 80 seeks + transfer time for 800 MB

    Same for writing.

Order of I/O Operations (Con't)

3. Reading sorted runs into memory for merging. In order to minimize "seeks" read one chunk of each run, so 80 chunks. Since the memory available is 10 MB each chunk can have 10,000,000/80 bytes = 125,000 bytes = 1,250 records
   - How many chunks to be read for each run?
   - size of a run/size of a chunk = 10,000,000/125,000=80
   - Total number of basic "seeks" = Total number of chunks (counting all the runs) is

     80 runs × 80 chunks/run = 80^2 chunks

Order of I/O Operations (Con't)

4. When writing a sorted file to disk, the number of basic seeks depends on the size of the output buffer: bytes in file/ bytes in output buffer.
   - For example, if the output buffer contains 200 K, the number of basic seeks is 200,000,000/200,000 = 4,000

- From steps 1-4 as the number of records (N) grows, step 3 dominates the running time

Step 3 : The Bottleneck

- There are ways of reducing the time for the bottleneck step 3

1. Allocate more resource (e.g. disk drive, memory)
2. Perform the merge in more than one step – this reduces the order of each merge and increases the run sizes
3. Algorithmically increase the length of each run
4. Find ways to overlap I/O operations