Databasesystems 2

Forum: https://forum-db.informatik.uni-tuebingen.de/c/ss18-db2

Assignment 10 (03.07.2018)

Submission: Tuesday, 10.07.2018, 10:00 AM

1. [15 Points] **Two-Way External Merge Sort**

   Given a relation of records distributed over 7 pages. Each page contains up to two records with one column of type `integer`. The pages hold the following records:

(3),(4)	(6),(2)	(9),(4)	(8),(7)	(5),(6)	(3),(1)	(2)

   Sort these pages with the help of the *Two-Way External Merge Sort algorithm*. Describe the procedure of the algorithm as follows:

   (a) For each pass, provide the runs created in the secondary memory.

   (b) Sketch the buffer content for each merge of *pass* 2 by providing every state of the three main memory pages. Represent each merge as seen in Figure 1.

   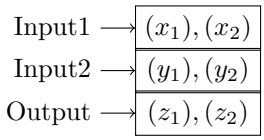

   Figure 1: Main Memory Buffer of the Two-Way External Merge Sort

2. [15 Points] **Replacement Sort**

   "*Replacement Sort* is an algorithm that can create longer runs in External Merge Sort's initial *pass* 0. Your task is to implement *Replacement Sort* in C, based on the algorithm provided in the following.

   For simplicity, we assume that each buffer page can only hold a single *record*, consisting of a single natural number. The program's input is, thus, a file containing an unsorted, line-separated sequence of singleton `integer` values, each representing a record (see file `input.seq` for example). Its output is a bunch of `.run`-files with the same structure, but each of which contain sorted sequences instead.

   `rpsort.c` provides you with a framework of the program. You have to complete it implementing function:

   ```
   /* Implementation of replacement sort ala Knuth, TAoCP, volume 3 */
   void replacement_sort(slot *buffer, int buffer_size, FILE *input_seq, FILE
       *output_run) {
       /*** YOUR CODE HERE ***/
   }
   ```

   (a) Implement the algorithm as given below and stick to the libraries imported and functions provided by `rpsort.c`.[1] Note that **solutions that do not compile are graded with zero points**. Also your solution should run on the given example file `input.seq` without unexpected errors.

[1] Of course you are still allowed to define additional functions to support your implementation of `replacement_sort()`.

Replacement Sort Algorithm:
Operates on a block of `buffer_size` memory *slots*. Each slot can store a single *record* value and has a *state* that is either `ON` or `OFF`.

Step 1: The `buffer_size` slots are filled with records from the input to be sorted.

Step 2: All slots are put into the `ON` state.

Step 3: Select the slot which has the smallest value of all `ON` slots.

Step 4: Transfer the contents of the selected slot to the output (call its value min).

Step 5: Replace the contents of the selected slot by the next input record:

- If new record value $> min$, go to **Step 3**.
- If new record value $= min$, go to **Step 4**.
- If new record value $< min$, go to **Step 6**.

Step 6: Turn the selected slot `OFF`.

- If all slots are now `OFF`, a sorted run is completed: Start a new run and go to **Step 2**.
- Else, go to **Step 3**.

(b) **Experiment:** Use your implementation to create the ouput runs for input `input.seq` (100000 records) and `buffer_size`s of 100, 1000 and 10000 slots. For each `buffer_size`, what is the average *run size*?