

Exercises on file organizations part 1 (with solutions)

Data management

A.Y. 2017 – 2018

Maurizio Lenzerini

Note




In the exercises on indexes, if not otherwise specified, you must assume that no page of the index is stored permanently in the buffer.

Exercise 1

Suppose we have a file stored in 600.000 pages, and we have 150 free frames available in the buffer. 🗨️

1. Illustrate in detail the algorithm for sorting the file by means of the multipass merge-sort method, specifying for each pass how many runs the algorithm produces, and which size (in terms of number of pages) have such runs.
2. Tell which is the cost of executing the algorithm in terms of number of page accesses?

Solution 1

1. In pass 0 we produce $600.000/150 = 4.000$ runs  (sorted portions of the file), each of size 150.
 -  In pass 1 we perform $4.000/149 = 26$ merge operations, each one on 149 runs, plus one merge operations on 126 runs. The total number of runs is 27. Each of the first 26 runs has size $149 \times 150 = 22.350$ pages, and the 27th run has size $126 \times 150 = 18.900$ pages.
 -  In pass 2 we merge the 27 runs into the final result, whose size is obviously 600.000.
2. Since the algorithm uses 3 passes, the cost is $2 \times B \times 3 = 6 \times 600.000 = 3.600.000$ page accesses.

Exercise 2

We have to sort a relation R with 375 pages using the multipass (or, k -way) merge-sort algorithm, and initially we have 200 free frames in the buffer. However, the system is currently very busy, and every time a run is written in secondary storage during the execution of the algorithm, after such writing the number of free frames in the buffer is halved. Describe in detail what happens during the execution of the multipass merge sort algorithm in this situation, and tell how many pages are accessed during such execution.

Solution 2

1. At the beginning of pass 0, we have 200 free frames in the buffer, and therefore we sort 200 pages of R in the buffer, and we write the corresponding first run of 200 pages. After such writing, the number of free frames is halved, and therefore we have 100 free buffer frames left. We then sort 100 pages of the remaining 175 pages of R, and we write the corresponding second run of 100 pages. After such writing, we have 50 free buffer frames left. We then sort 50 pages of the remaining 75 pages of R, and we write the corresponding third run of 50 pages. After such writing, we have 25 free buffer frames left. We then sort the last 25 pages of R, and we write the corresponding fourth run of 25 pages. After such writing, we have 12 free buffer frames left.
2. Now pass 0 is completed, and, since we have 4 sorted runs, we can simply perform pass 1 of the algorithm, by using 5 of the 12 free buffer frames for merging the 4 runs and obtaining the sorted file constituting the result.
3. The resulting algorithm has the same complexity as the two-pass algorithm, and therefore the number of page accesses required by the algorithm is $2 \times 375 \times 2 = 1.500$.

Exercise 3

We have a relation R with 15.000.000 tuples, where each tuple has 4 attributes. We assume that every attribute and every pointer has the same size. We know that 10 tuples of R fit in one page. Consider a primary, clustering sorted index using alternative 2 for R , with one attribute as a search key, which is also a key of the relation. Tell which is the number of page accesses required for searching for and accessing a tuple in R given a value of the search key, in the two cases of dense and sparse index.

Solution 3

The solution is based on computing the number of pages in the index.

If the index is dense, since we have to store 15.000.000 data entries, each data entry has 2 attributes, and we know that 10 tuples of 4 attributes fit in one page, we infer that 40 attribute values fits in one page, which means that 20 data entries fit in one page and therefore we have $15.000.000/20 = 750.000$ pages in the index. Since $\log_2 750.000 = 19.6$, we need 20 page accesses, plus the one to reach the page with the desired tuples of R. So the total number is 21.

If the index is sparse, we have to store one data entry for each page, i.e., $15.000.000/10 = 1.500.000$ data entries. Therefore we have $1.500.000/20 = 75.000$ pages in the index. Since $\log_2 75.000 = 16.1$, we need 17 page accesses plus the one to reach the page with the desired tuple of R. So the total number is 18.

Exercise 4

In general, a secondary, non-unique index contains duplicates (we remind the students that a duplicate is a pair of data entries with the same value for the search key). Are there cases where a secondary, non-unique index does not contain duplicates? If yes, which are those cases? Explain the answer in detail.

Solution 4

There are at least two cases where a secondary, non-unique index does not contain duplicates:

- The index uses alternative (3), and therefore every relevant value of the search key is stored only once in the index, but with a list of rids associated to it.
- The index uses alternative (2), and is clustering. Indeed, in this case, for each relevant value k of the search key, we can store in the index only one data entry, and we can make this data entry point to the first data record r with the value k for the search key. Since the index is clustering, the other data records with value k for the search key follow immediately r in the data file, and therefore, given k , we can easily access all of them after the access to r .

Exercise 5

We have a relation R with 25.000.000 tuples, where each tuple has 6 attributes. We assume that every attribute and every pointer has the same size. We know that 15 tuples of R fit in one page. Consider a dense, clustering sorted index using alternative 2 for R , with one attribute as a search key. We know that, in the average, 20 records of R have the same value of the search key. Tell which is the average number of page accesses required for searching for and accessing the tuple in R given a range of 5 value of the search key.

Solution 5

We need to compute the number of pages in the index. Since we have to store 25.000.000 data entries, each data entry requires 2 attributes, and we know that 15 tuples of 6 attributes fit in one page, we infer that 90 attribute values fits in one page, which means that 45 data entries fit in one page and therefore we have $25.000.000/45 = 555.555$ pages in the index. Since $\log_2 555.555 = 20$, we need 20 page accesses, plus the number of pages of R to reach the desired tuples of R. Since the range has 5 values, the number of pages of R is the smallest integer greater than $(20 \text{ times } 5) / 15 = 6,6666$. So the total number is 27.

Exercise 6

We have a relation R with 15.000.000 tuples, where each tuple has 4 attributes. We assume that every attribute and every pointer has the same size. We know that 15 tuples of R fit in one page, and that R contains 31.250 different values of the search key. Consider a clustering, secondary, non-unique sparse sorted index using alternative 2 for R , with one attribute as a search key. Tell which is the number of page accesses required for accessing all tuples of R with a given value of the search key.

Solution 6

We need to compute the number of pages in the index. Since we have to store one data entry for each page of R, we need to compute the number of pages of R. Since R stores 15.000.000 data entries, and 15 tuples fit in one page, R is stored in 1.000.000 pages. So, we have to store 1.000.000 data entries, and since 30 data entries fit in one page, we conclude that we have $1.000.000/30 = 33.333$ pages in the index. Since $\log_2 33.333 = 16$, we need $16 + J$ page accesses, where J is the number of data pages of R to be accessed.

In the average, we need to access $15.000.000 / 31.250 = 480$ records in R and therefore $J = 480/15 + 1 = 32 + 1$. It follows that the total number of page accesses is 49.

Exercise 7

Suppose we have a relation R on 9 attributes with 6.000.000 tuples, and 100 tuples fit in one page. Suppose that 2 attributes form the primary key of R . Consider the operation that, given a value V (a pair), checks whether there exists a tuple in R with the value V for the primary key. Tell the cost (number of page accesses) of the operation in all the following situations:

1. R is stored as a heap file, with no index.
2. R is stored as a sorted file on the primary key, with no index.
3. R is stored as a heap file, with a primary sorted index.
4. R is stored as a heap file, with a 2-level primary sorted index.
5. R is stored as a sorted file on the primary key, with a primary sorted index.

Solution 7

1. R is stored as a heap file, with no index.

The number of pages of R is $600.000/100 = 6.000$, and therefore the cost is 6.000.

2. R is stored as a sorted file on the primary key, with no index.

The cost is $\log_2 6.000 = 13$

3. R is stored as a heap file, with a primary sorted index

The index cannot be sparse, because is unclustering. Therefore it is dense. Each data entries is constituted by 3 attributes (we assume that pointers have the same size of other attributes). Since 100 tuples with 9 values each fit in one page, we have that 300 data entries with 3 values each fit in one page, and this means that the number of pages for the index is 2.000. The cost is $\log_2 2.000 + 1 = 11 + 1 = 12$

Solution 7

4. R is stored as a heap file, with a 2-level primary sorted index.

With respect to the previous case we add one level, constituted by a sparse index on the 2000 pages of the first-level index. In this second level we have one index entry for each page in the first level, where each index entry is constituted by 2 attributes. Since 100 tuples of 9 attributes fit in one page, we have that 450 index entries fit in one page. Therefore we need $2000 / 450 = 5$ pages for the second-level index, and the cost is $\log_2 5 + 1 + 1 = 3 + 2 = 5$ page accesses.

5. R is stored as a sorted file on the primary key, with a primary sorted index.

Since now the index is clustering, it can be sparse, and therefore we need $6000 / 300 = 20$ pages for the index, and the cost is $\log_2 20 + 1 = 5 + 1 = 6$ page accesses.