Exercises on file organizations part 1 (with solutions)

Data management

A.Y. 2018 – 2019

Maurizio Lenzerini

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 produced in a merge operation has size 149 × 150 = 22.350 pages, and the 27^{th} run has size 126 × 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 × B × 3 = 6 × 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, and pass 0 is completed.
2. 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 to sort a relation R with 9.000 pages using the multipass (or, k-way) merge-sort algorithm, and we know that the number of free frames in the buffer that will be available for the algorithm is between 50 and 100. Tell which is the cost of the algorithm in terms of the number of page accesses, both in the worst case, and in the best case.

Solution 3

The worst case is obviously in the case where the number F of free buffer frames remains 50 for the whole execution of the algorithm. In this case, since $F \times (F-1) = 2.450$ and $F \times (F-1) \times (F-1) = 120.050$, we have $9.000 \geq F \times (F-1)$ and $9.000 \leq F \times (F-1) \times (F-1)$, and therefore the number of passes required to sort the relation is 3. Thus, the cost is $2 \times 3 \times 9.000 = 54.000$.

The best case is obviously in the case where the number F of free buffer frames remains 100 for the whole execution of the algorithm. In this case, since $F = 100$ and $F \times (F-1) = 9.900$, we have $9.000 \geq F$ and $9.000 \leq F \times (F-1)$, and therefore the number of passes required to sort the relation is 2. Thus, the cost is $2 \times 2 \times 9.000 = 36.000$.

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 4

We have a relation R(A,B,C,D) with 15.000.000 tuples, where A is the primary key, and we know that every attribute and every pointer has the same size. We also know that 10 tuples of R fit in one page, and there is a primary, clustering sorted index using alternative 2 for R, with A as search key. Tell which is the number of page accesses required for answering the following query

```
select B, C
from R
where A = 500
```

using the index, in the two cases of dense and sparse index.

Solution 4

The solution is based on computing the number of pages in the index, and then computing the number of page accesses required for performing the equality search using 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 index 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 reach the page with the desired tuple of R. So the total number is 18.

Exercise 5

In general, a secondary, non-unique index contains duplicates (we remind the students that a duplicate is a pair of different 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 5

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 6

We have a relation R(A,B,C,D,E,F) with 25.000.000 tuples. We assume that every attribute and every pointer has the same size. We know that 15 tuples of R fit in one page, that there is a dense, clustering sorted index using alternative 2 for R, with D as search key, and that, in the average, 21 records of R have the same value of the search key D. Tell which is the number of page accesses required for answering the following query

```
select A, B, C
from R
where D >= 61 and D <= 65
```

using the index.

Solution 6

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 index 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 containing the desired tuples. Since the range has 5 values, the number of pages of R containing the desidered tuples is the smallest integer greater than 21 × 5 / 15 = 7. It follows that the total number of page accesses is 27.

Exercise 7

We have a relation R(A,B,C,D) with 15.000.000 tuples. 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 attribute C. We also know that there is a clustering, secondary, non-unique sparse sorted index using alternative 2 for R, with attribute C as a search key.

Tell which is the number of page accesses required for answering the following query

select B, C, D
from R
where C = 70

using the index.

Solution 7

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 16 + 33 = 49.

Exercise 8

Suppose we have a relation R(A,B,C,D,E,F,G,H,L) with 6.000.000 tuples, where 100 tuples fit in one page. As usual, all attributes and pointers have the same size. Consider the Boolean query

```
select true
from R
where A = 30 and B = 60
```

Tell which is the cost (number of page accesses) of the operation in each of 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 8

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

   The number of pages of R is 6.000.000/100 = 6.000, and therefore the cost is 60.000.

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

   The cost is $\log_2 60.000 = 16$

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 values of the same size. Since 100 tuples with 9 values each fit is one page, we have that 300 data entries, each with 3 values, fit in one index page, and this means that the number of pages for the index is 6.000.000/300 = 20.000. The cost is $\log_2 20.000 = 15$.

Solution 8

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

   With respect to the previous case (primary dense index) 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 3 values. Since 100 tuples of 9 attributes fit in one page, we have that ~~450~~ index entries fit in one page. Therefore we need 20.000 / 300 = 67 pages for the second-level index, and the cost is $\log_2 67 + 1 = 6 + 1 = 7$ 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 60.000 / 300 = 200 pages for the index, and the cost is $\log_2 200 + 1 = 8 + 1 = 9$ page accesses.