Exercises on file organizations part 1 (with solutions)

Data management
A.Y. 2024 – 2025
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.

PASS 0:

WE PRODUCE $600\,000 / 150 = 4000$ RUNS

EACH RUN IS A SORTED PORTION OF 150 PAGES

PASS 1:

$4000 \gg 150$, SO WE PERFORM $4000/149 \approx 27$ MERGE OPERATIONS.

26 MERGE OPERATIONS ON 149 RUNS $\rightarrow$ $149 \cdot 150 = 22350$ PAGES

1 MERGE OPERATION ON 126 RUNS $\rightarrow$ $126 \cdot 150 = 18900$ PAGES

PASS 2:

WE MERGE THE 27 RUNS INTO ONE FINAL (600 000 PAGES)

COST $= 2 \cdot 3 \cdot B = 3600000$ PAGE ACCESSES

Solution 1

1. In pass 0 we produce 600.000/150 = 4.000 runs. Each run is a sorted portion of the file with 150 pages.

   Since 4.000/149 = 26 with the remainder of 126, in pass 1 we perform 26 merge operations, each one on 149 runs, plus one merge operations on 126 runs. The total number of runs produced in pass 1 is 27. Each of the first 26 runs produced in a merge operation 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.

PASS 0:

THERE ARE 200 FREE FRAMES, SO WE SORT 200 PAGES OF R IN THE BUFFER, AND WE WRITE THE FIRST RUN OF 200 PAGES.
AFTER THIS WRITE THE FREE FRAMES ARE HALVED ($200/2 = 100$)
WE SORT 100 PAGES OF THE REMAINING 175 OF R IN 100 FREE FRAMES, AND WE WRITE THE SECOND RUN OF 100 PAGES.
THE FREE FRAMES ARE NOW 50. WE SORT 50 PAGES OF THE REMAINING 75 OF R IN 50 FREE FRAMES, AND WE WRITE THE THIRD RUN OF 50 PAGES.
THE FREE FRAMES ARE NOW 25. WE SORT 25 PAGES OF THE REMAINING 25 OF R IN 25 FREE FRAMES, AND WE WRITE THE LAST RUN OF 25 PAGES.

PASS 1:

WE HAVE 4 SORTED RUNS AND 12 FREE FRAMES, SO WE CAN USE 5 FRAMES TO MERGE THE 4 RUNS INTO ONE FINAL.

COST = $2 \cdot 2 \cdot 375 = 1500$ PAGE ACCESSES

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 produced in pass 0, 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 final run, i.e., 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.

WORST CASE (50 FREE FRAMES)

PASS 0.

$9000 / 50 = 180$ RUNS   180 RUNS OF 50 PAGES

PASS 1:

$180 >> 50$, SO WE PERFORM $180/49 = 4$ MERGE OPERATIONS

PASS 2:

WE MERGE THE 4 RUNS INTO ONE FINAL (9000 PAGES)

COST $= 2 \cdot 3 \cdot B = 54\,000$ PAGE ACCESSES

BEST CASE (100 FREE FRAMES)

PASS 0:

$9000 / 100 = 90$ RUNS   90 RUNS OF 100 PAGES

PASS 1:

$90 < 99$

WE MERGE THE 90 RUNS INTO ONE FINAL (9000 PAGES)

COST $= 2 \cdot 2 \cdot B = 36\,000$ PAGE ACCESSES

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×(F-1) = 2.450 and F×(F-1)×(F-1) = 120.050, we have 9.000 ≥ F×(F-1) and 9.000 ≤ F×(F-1)×(F-1), and therefore the number of passes required to sort the relation is 3. Thus, the cost is 2 × 3 × 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×(F-1) = 9.900, we have 9.000 ≥ F and 9.000 ≤ F×(F-1), and therefore the number of passes required to sort the relation is 2. Thus, the cost is 2 × 2 × 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(<u>A</u>,B,C,D) with 15.000.000 tuples, where A is the primary key, and we know that every attribute and every pointer (record id) 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.

DENSE

WE HAVE 15 000 000 / 10 = 1 500 000 PAGES.

A DENSE INDEX HAS AN ENTRY FOR EACH RECORD (15 000 000 DATA ENTRIES).

EACH PAGE HAS 10 TUPLE · 4 ATTRIBUTES = 40 VALUES.

EACH DATA ENTRY HAS 2 ATTRIBUTES <KEY, RID>, SO

15 000 000 · 2 = 30 000 000 VALUES IN THE INDEX.

SINCE WE HAVE 40 VALUES FOR EACH INDEX PAGE, AND 30 000 000

DATA ENTRIES IN TOTAL $\longrightarrow$ 30 000 000 / 40 = 750 000 INDEX PAGES.

WE PERFORM A BINARY SEARCH ON THE INDEX $\rightarrow \log_2(750000) \approx 19.6$

COST = 20 + 1 (TO REACH THE PAGE WITH A = 500) = 21 PAGE ACCESSES

SPARSE

WE HAVE TO STORE ONE DATA ENTRY FOR EACH PAGE

15 000 000 / 10 = 1 500 000 PAGES (DATA ENTRIES).

1 500 000 · 2 = 3 000 000 VALUES IN THE INDEX.

3 000 000 / 40 = 75 000 INDEX PAGES.

$\log_2(75\,000) \approx 16.1$ $\rightarrow$ COST = 17 + 1 = 18 PAGE ACCESSES

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 with binary 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 to reach the page with the desired tuple of R. So, the total number is 18.

Exercise 5

We have a relation R(A,B,C,D,E,F) with 25.000.000 tuples. We assume that every attribute and every pointer (record id) 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, both in the case of strongly dense and in the case of dense index.

STRONGLY DENSE

A DATA ENTRY FOR EACH TUPLE → 25 000 000 DATA ENTRIES

IN EACH PAGE WE HAVE $15 \cdot 6 = 90$ VALUES

IN EACH INDEX PAGE WE HAVE:

25 000 000 · 2 (<KEY, RID>) = 50 000 000 VALUES

EACH INDEX PAGE HAS 90 VALUES, SO:

50 000 000 / 90 = 555 555 INDEX PAGES

WE PERFORM BINARY SEARCH → $\log_2(555\,555) \approx 19{,}1 = 20$

IN "$D \geq 61$ AND $D \leq 65$" WE HAVE 5 VALUES. SINCE 21 RECORDS HAVE THE SAME VALUE OF D, WE NEED TO SEE 21 5 = 105 TUPLES

THIS MEANS THAT WE ACCESS 105 / 15 = 7 PAGES + 1 (MARGIN PAGE)

COST = 20 + 7 + 1 = 28 PAGE ACCESSES

DENSE

SINCE 21 RECORDS HAVE THE SAME VALUE OF D, WE HAVE:

$$25\,000\,000 / 21 = 1\,190\,477 \text{ DATA ENTRIES}$$

A PAGE HAS $15 \cdot 6 = 90$ VALUES

AN INDEX PAGE HAS 2 ATTRIBUTES <KEY, RID>:

$$1\,190\,477 \cdot 2 = 2\,380\,953 \text{ VALUES}$$

$$2\,380\,953 / 90 = 26\,456 \text{ INDEX PAGES}$$

WE PERFORM BINARY SEARCH $\rightarrow \log_2(26\,456) = 15$

IN "$D \geq 61$ AND $D \leq 65$" WE HAVE 5 VALUES. SINCE 21 RECORDS HAVE THE SAME VALUE OF D, WE NEED TO SEE $21\ 5 = 105$ TUPLES

THIS MEANS THAT WE ACCESS $105 / 15 = 7$ PAGES $+ 1$ (MARGIN PAGE)

COST = 15 + 7 + 1 = 23 PAGE ACCESSES

Solution 5

Let us start with the case of «strongly dense index» (i.e., one data entry for each tuple in R). 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 or equal to 21 × 5 / 15, i.e 7. Since the 21 × 5 = 105 tuples may not start at the beginning of the first page, we count the total number of page accesses as 20 + 7 +1 = 28.

Solution 5

Let us consider the case of (not strongly) dense index (i.e., the case where we have one data entry for each value stored in the search key in R).

Since in the average 21 records of R have the same value of the search key D, in this case we have to store 25.000.000/21 = 1.190.477 data entries. Since 45 data entries fit in one index page, we have 1.190.477/45 = 26.456 pages in the index. Since $\log_2$ 26.456 = 5 and since the 21 × 5 = 105 tuples may not start at the beginning of the first page the cost is 5 + 7 + 7 + 1 = 13 page accesses.

Exercise 6

We have a relation R(A,B,C,D) with 15.000.000 tuples. We assume that every attribute and every pointer (record id) 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.

WE HAVE A SPARSE INDEX, SO A DATA ENTRY FOR EACH PAGE.

THERE ARE $15\,000\,000 / 15 = 1\,000\,000$ PAGES.

IN EACH PAGE THERE ARE $15 \cdot 4 = 60$ VALUES.

IN EACH INDEX PAGE WE HAVE:

$$1\,000\,000 \cdot 2(\langle KEY, RID \rangle) = 2\,000\,000 \text{ VALUES}$$

$$2\,000\,000 / 60 = 33\,334 \text{ INDEX PAGES}$$

WE PERFORM BINARY SEARCH. $\log_2(33\,334) \approx 16$

31 250 DIFFERENT VALUE OF C. WE NEED TO ACCESS

$$15\,000\,000 / 31250 = 480 \text{ TUPLES}$$

$$480 / 15 = 32 \text{ PAGES} + 1 \text{ (MARGIN PAGE)}$$

$$\text{COST} = 16 + 32 + 1 = 49 \text{ PAGE ACCESSES}$$

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 contains 15.000.000 tuples, 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 7

Suppose we have a relation R(<u>A</u>,<u>B</u>,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.

1 6 000 000 / 100 = 60000 PAGES
WE HAVE TO SEE ALL PAGES → 60 000 PAGE ACCESSES

2. WE PERFORM BINARY SEARCH → $\log_2 60\,000 = 16$

3 SINCE THE R IS STORED AS A HEAP, THE INDEX IS DENSE AND NOT SPARSE, SO A DATA ENTRY FOR EACH TUPLE.

IN EACH PAGE. $100 \cdot 9 = 900$ VALUES

IN EACH INDEX PAGE. 6000000 3 (<KEY A, KEY B, RID>) = 18 000 000 VALUES

18 000 000 / 900 = 20 000 INDEX PAGES

WE PERFORM BINARY SEARCH → $\log_2 20\,000 = 15$

4. WE ADD A SPARSE INDEX ON THE 20000 INDEX PAGES OF THE FIRST LEVEL. IN THE SECOND LEVEL WE HAVE AN INDEX ENTRY FOR EACH INDEX PAGE OF THE FIRST LEVEL.

20000 (INDEX PAGES - FIRST) $\cdot$ 3 = 60 000 VALUES (SECOND)

60000 / 900 (VALUES IN ONE PAGE) = 67 INDEX PAGES (SECOND)

COST = $\log_2 67$ + 1 (FIRST LEVEL ACCESS) = 7 PAGE ACCESSES

5. CLUSTER INDEX, WE CAN USE A SPARSE INDEX. A DATA ENTRY FOR EACH PAGE (60 000).

$60\,000 \cdot 3 = 180\,000$ VALUES → 180 000 / 900 = 200 INDEX PAGES

COST = $\log_2 200$ + 1 (ACCESS R) = 9 PAGE ACCESSES

Solution 7

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

   The number of pages of R is 6.000.000/100 = 60.000, 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, as R is stored as a heap, it is unclustering. Therefore, it is dense. Each data entry 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 (by index-only evaluation, since we do not have to access R).

Solution 7

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 20.000 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 300 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 (note that we have to access R in this case).