JONATHAN PETERSEN
A01236750
CS 5700 - DATABASE SYSTEMS

Homework 2

CS 5700 - Database Systems

17.28. A file has r = 20,000 STUDENT records of fixed length. Each record has the following fields: Name (30 bytes), Ssn (9 bytes), Address (40 bytes), PHONE (10 bytes), Birth_date (8 bytes), Sex (1 byte), Major_dept_code (4 bytes), Minor_dept_code (4 bytes), Class_code (4 bytes, integer), and Degree_program (3 bytes). An additional byte is used as a deletion marker. The file is stored on the disk whose parameters are given in Exercise 17.27.

a. Calculate the record size R in bytes.

R = 30 + 9 + 40 + 10 + 8 + 1 + 4 + 4 + 4 + 3 + 1

R = 114 bytes

b. Calculate the blocking factor bfr and the number of file blocks b, assuming an unspanned organization.

bfr = floor(blockSize / R)

bfr = floor(512 / 114)

bfr = 4

c. Calculate the average time it takes to find a record by doing a linear search on the file if (i) the file blocks are stored contiguously, and double buffering is used; (ii) the file blocks are not stored contiguously.

i. initial seek + (b / 2) - 1 * sequential seek, where b is the number of blocks

= 30 + ((20,000 / 4) / 2) - 1 * 1

= 2,529 ms

ii. random seek * (b / 2), where b is the number of blocks

= 30 * ((20,000 / 4) / 2)

= 75,000 ms

d. Assume that the file is ordered by Ssn; by doing a binary search, calculate the time it takes to search for a record given its Ssn value.

random seek * log(b), where b is the number of blocks

= 30 * log_2(2,500)

= approx. 338.631 ms

18.18. Consider a disk with block size B = 512 bytes. A block pointer is P = 6 bytes long, and a record pointer is P_R = 7 bytes long. A file has r = 30,000 EMPLOYEE records of fixed length. Each record has the following fields: Name (30 bytes), Ssn (9 bytes), Department_code (9 bytes), Address (40 bytes), Phone (10 bytes), Birth_date (8 bytes), Sex (1 byte), Job_code (4 bytes), and Salary (4 bytes, real number). An additional byte is used as a deletion marker.

a. Calculate the record size R in bytes.

R = 30 + 9 + 9 + 40 + 10 + 8 + 1 + 4 + 1

R = 112 bytes

b. Calculate the blocking factor bfr and the number of file blocks b, assuming an unspanned organization.

bfr = floor(blockSize / R)

bfr = floor(512 / 112)

bfr = 4

c. Suppose that the file is ordered by the key field Ssn and we want to construct a primary index on Ssn. Calculate (i) the index blocking factor bfr_i (which is also the index fan-out fo); (ii) the number of first-level index entries and the number of first-level index blocks; (iii) the number of levels needed if we make it into a multilevel index; (iv) the total number of blocks required by the multilevel index; and (v) the number of block accesses needed to search for and retrieve a record from the file—given its Ssn value—using the primary index.

i. Ri = size of ssn + size of block pointer

= 9 + 6

= 15 bytes

bfri = floor(blockSize / Ri)

= floor(512 / 15)

= 34 = fo

ii. first-level entries = number of entries / bfr

= 30,000 / 4

= 7,500 entries

first-level blocks = ceil(first-level entries / bfri)

= ceil(7,500 / 34)

= 221 blocks

iii. number of levels = ceil(log_fo(first-level entries)))

= ceil(log_34(7500))

= 3 levels

iv. total index blocks = level three blocks + level two blocks + level one blocks

= 1 + ceil(193 / 34) + 193

= 1 + 6 + 193

= 200 blocks

v. total blocks retrieved for primary lookup = log_2(number of first-level blocks) + 1

= 8 + 1

= 9 blocks retrieved

d. Suppose that the file is not ordered by the key field Ssn and we want to construct a secondary index on Ssn. Repeat the previous exercise (part c) for the secondary index and compare with the primary index.

i. Ri = size of ssn + size of record pointer

= 9 + 7

= 16 bytes

bfri = floor(blockSize / Ri)

= floor(512 / 16)

= 32 = fo

ii. first-level entries = number of entries

= 30,000 entries

first-level blocks = ceil(first-level entries / bfri)

= ceil(30,000 / 32)

= 938 blocks

iii. number of levels = ceil(log_fo(first-level entries)))

= ceil(log_32(30,000))

= 3 levels

iv. total index blocks = level three blocks + level two blocks + level one blocks

= 1 + ceil(938 / 32) + 938

= 1 + 30 + 770

= 801 blocks

v. total blocks retrieved for primary lookup = log_2(number of first-level blocks) + 1

= 10 + 1

= 11 blocks retrieved

e. Suppose that the file is not ordered by the nonkey field Department_code and we want to construct a secondary index on Department_code, using option 3 of Section 18.1.3, with an extra level of indirection that stores record pointers. Assume there are 1,000 distinct values of Department_code and that the EMPLOYEE records are evenly distributed among these values. Calculate (i) the index blocking factor bfr_i (which is also the index fan-out fo); (ii) the number of blocks needed by the level of indirection that stores record pointers; (iii) the number of first-level index entries and the number of first-level index blocks; (iv) the number of levels needed if we make it into a multilevel index; (v) the total number of blocks required by the multilevel index and the blocks used in the extra level of indirection; and (vi) the approximate number of block accesses needed to search for and retrieve all records in the file that have a specific Department_code value, using the index.

i. Rindirection = size of record pointer

= 7 bytes

bfrindirection = floor(blockSize / Ri)

= floor(512 / 7)

= 73 = fo

ii. records per unique value = 30,000 / 1,000

= 30 distinct records

blocks per value = 1, since 30 records > 73 bfrindirection

blocks for indirection layer = number of distinct values * blocks per value

= 1000 * 1

= 1000 blocks

iii. Ri = size of Department_code + size of block pointer

= 9 + 6

= 15 bytes

bfri = floor(blockSize / Ri)

= floor(512 / 15)

= 34 = fo

number of first-level entries = number of distinct values

= 1000 entries

number of first-level blocks = ceil(number of first-level entries / bfri)

= ceil(1000 / 34)

= 30 blocks

iv. number of levels = ceil(log_fo(first-level entries)))

= ceil(log_34(1,000))

= 2 levels

v. total multilevel blocks = second-level blocks + first-level blocks + indirection layer

= 1 + 30 + 1000

= 1031 blocks

vi. approximate total block accesses

= log_2(first-level blocks)

+ blocks per indirection entry

+ avg entries per indirection entry

= 5 + 1 + 30

= 36 blocks

f. Suppose that the file is ordered by the nonkey field Department_code and we want to construct a clustering index on Department_code that uses block anchors (every new value of Department_code starts at the beginning of a new block). Assume there are 1,000 distinct values of Department_code and that the EMPLOYEE records are evenly distributed among these values. Calculate (i) the index blocking factor bfr_i (which is also the index fan-out fo); (ii) the number of first-level index entries and the number of first-level index blocks; (iii) the number of levels needed if we make it into a multilevel index; (iv) the total number of blocks required by the multilevel index; and (v) the number of block accesses needed to search for and retrieve all records in the file that have a specific Department_code value, using the clustering index (assume that multiple blocks in a cluster are contiguous).

i. Ri = size of Department_code + size of block pointer
= 9 + 6
= 15 bytes
bfri = floor(blockSize / Ri)
= floor(512 / 15)
= 34 = fo

ii. first-level entries = number of distinct entries
= 1,000 entries
first-level blocks = ceil(first-level entries / bfri)
= ceil(1,000 / 34)
= 30 blocks

iii. number of levels = ceil(log_fo(first-level entries)))
= ceil(log_34(1000))
= 2 levels

iv. total index blocks = level two blocks + level one blocks
= 1 + 30
= 31 blocks

v. size of clustering group = total records / number of distinct entries
= 30,000 / 1,000
= 30 blocks

total blocks retrieved for primary lookup

= log_2(number of first-level blocks) + size of clustering group

= 5 + 30

= 39 blocks retrieved

g. Suppose that the file is not ordered by the key field Ssn and we want to construct a B^+-tree access structure (index) on Ssn. Calculate (i) the orders p and p_{leaf} of the B^+-tree; (ii) the number of leaf-level blocks needed if blocks are approximately 69 percent full (rounded up for convenience); (iii) the number of levels needed if internal nodes are also 69 percent full (rounded up for convenience); (iv) the total number of blocks required by the B^+-tree; and (v) the number of block accesses needed to search for and retrieve a record from the file—given its Ssn value—using the B^+-tree.

i. (p * block pointer size) + ((p - 1) * size of ssn) ≤ block size

6p + 9p - 9 ≤ 512

15p ≤ 521

p ≤ 34.73 bytes

p = 34

(pleaf * (size of ssn + size of record pointer)) + size of next pointer ≤ block size

(9 + 7)pleaf + 6 ≤ 512

16pleaf ≤ 506

pleaf ≤ 31.625 bytes

pleaf = 31

ii. number of leaf-level blocks at 69% capacity = number of records / (0.69 * pleaf)

= 30,000 / floor(0.69 * 31)

= ceil(30,000 / 21)

= 1,429 leaf nodes

iii. fop = round(0.69 * p)

= 23 pointers

number of levels = ceil(log_fop(number of leaf nodes))

= ceil(log_23(1429))

= 3 levels

iv. total blocks = level-three blocks + level-two blocks + level-one blocks + leaf blocks

= 1 + 23 + 23^2 + 1429

= 1982 blocks

v. number of block accesses

= level-three lookup

+ level-two lookup

+ level-one lookup

+ leaf lookup

+ record lookup

= 1 + 1 + 1 + 1 + 1

= 5 blocks

19.13. Consider queries Q1, Q8, Q1B, and Q4 in Chapter 4 and Q27 in Chapter 5.

a. Draw at least two query trees that can represent each of these queries. Under what circumstances would you use each of your query trees?

SEE ATTACHMENT FOR TREES

Q1:

SELECT Fname, Lname, Address

FROM EMPLOYEE, DEPARTMENT

WHERE Dname='Research'AND Dnumber=Dno;

Q8:

SELECT E.Fname, E.Lname, S.Fname, S.Lname

FROM EMPLOYEE AS E, EMPLOYEE AS S

WHERE E.Super_ssn=S.Ssn;

Q1B:

```
SELECT E.Fname, E.LName, E.Address
FROM EMPLOYEE E, DEPARTMENT D
WHERE D.DName='Research'AND D.Dnumber=E.Dno;
```

Q4:

```
(SELECT DISTINCT Pnumber
FROM PROJECT, DEPARTMENT, EMPLOYEE
WHERE Dnum=Dnumber AND Mgr_ssn=Ssn
AND Lname='Smith' )
UNION
( SELECT DISTINCT Pnumber
FROM PROJECT, WORKS_ON, EMPLOYEE
WHERE Pnumber=Pno AND Essn=Ssn
AND Lname='Smith' );
```

Q27:

```
SELECT Pnumber, Pname, COUNT (*)
FROM PROJECT, WORKS_ON, EMPLOYEE
WHERE Pnumber=Pno AND Ssn=Essn AND Dno=5
GROUP BY Pnumber, Pname;
```

b. Draw the initial query tree for each of these queries, and then show how the query tree is optimized by the algorithm outlined in Section 19.7.

SEE ATTACHMENT

c. For each query, compare your own query trees of part (a) and the initial and final query trees of part (b).

Q1: My second query tree is similar to the final tree of part b, only I did not move the projections as far down the tree as possible.

Q8: My second query tree is the same as the tree produced by the algorithm.

Q1B: My first query tree is the same as the tree produced by the algorithm, my second tree is arguably better, but it relies on an assumption that may or may not be true.

Q4: My second query tree is similar to the final query tree for Q4, but I left off a few projections here and there.

19.17 Can a nondense index be used in the implementation of an aggregate operator? Why or why not?

It can, but extra care must be taken to insure that the proper data is retrieved. For MAX and MIN aggregates, we must be sure that the smallest and largest values are stored in our index, such as how a B+-tree would store them. For AVG or SUM aggregates we must maintain a count of the number of values per index entry so that we may use the actual total number of elements in these functions. This is also similar to how a COUNT aggregate would function on a sparse index.

19.22 Compare the cost of two different query plans for the following query:

$\sigma_{(\text{Salary} > 40000)}(\text{EMPLOYEE} \bowtie_{\text{Dno=Dnumber}} \text{DEPARTMENT})$

Use the database statistics in Figure 19.8.

SEE ATTACHMENT FOR TREES