San Jose State University

CMPE 138/180B: Database Systems

Fall 2024

Homework 2

Please submit your solution to Canvas for this assignment, as a PDF document. Follow the homework policy document for instructions on how to submit this homework.

Problem 1 (30 pts)

Consider a disk with the following characteristics (these are not parameters of any particular disk unit): block size B=512 bytes, interblock gap size G=128 bytes, number of blocks per track=20, number of tracks per surface=400. A disk pack consists of 15 double-sided disks.

(a)[4 points]

What is the total capacity of a track and what is its useful capacity (excluding interblock gaps)?

b) [4 points]

How many cylinders are there?

c) [4 points]

What is the total capacity and the useful capacity of a cylinder?

d) [4 points]

What is the total capacity and the useful capacity of a disk pack?

e) [4 points]

Suppose the disk drive rotates the disk pack at a speed of 2400 rpm (revolutions per minute); what is the transfer rate in bytes/msec and the block transfer time btt in

msec? What is the average rotational delay rd in msec? What is the bulk transfer rate (see Appendix B)?

f) [5 points]

Suppose the average seek time is 30 msec. How much time does it take (on the average) in msec to locate and transfer a single block given its block address?

g) [5 points]

Calculate the average time it would take to transfer 20 random blocks and compare it with the time it would take to transfer 20 consecutive blocks using double buffering to save seek time and rotational delay.

Solution:

(a) Total track size = 20 * (512+128) = 12800 bytes

Useful capacity of a track = 20 * 512 = 10240 bytes

(b) Number of cylinders = number of tracks = 400

(c) Total cylinder capacity = 15*2*20*(512+128) = 384000 bytes

Useful cylinder capacity = 15 * 2 * 20 * 512 = 307200 bytes

(d) Total capacity of a disk pack = 15 * 2 * 400 * 20 * (512+128)

= 153600000 bytes = 153.6 Mbytes

Useful capacity of a disk pack = 15 * 2 * 400 * 20 * 512 = 122.88 Mbytes

(e) Transfer rate tr= (total track size in bytes)/(time for one disk revolution in msec)

tr= (12800) / ((60 * 1000) / (2400)) = (12800) / (25) = 512 bytes/msec

block transfer time btt = B / tr = 512 / 512 = 1 msec

average rotational delay rd = (time for one disk revolution in msec) / 2 = 25 / 2

= 12.5 msec

bulk transfer rate btr= tr * (B/(B+G)) = 512*(512/640) = 409.6 bytes/msec

(f) average time to locate and transfer a block = s+rd+btt = 30+12.5+1 = 43.5 msec

(g) time to transfer 20 random blocks = 20 * (s + rd + btt) = 20 * 43.5 = 870 msec

time to transfer 20 consecutive blocks using double buffering = s + rd + 20*btt

= 30 + 12.5 + (20*1) = 62.5 msec

(a more accurate estimate of the latter can be calculated using the bulk transfer rate as follows: time to transfer 20 consecutive blocks using double buffering

= s+rd+((20*B)/btr) = 30+12.5+ (10240/409.6) = 42.5+ 25 = 67.5 msec)

Problem 2 (20 pts)

Let's assume you've inherited a database of **2 billion** rows from your relatives. The rows contain a string field called *'unique name'* and a few other fields. Given the enormous size of the database, you decide to build a B+ tree indices on the *'unique name'* field.

Use the following values for calculations:

- Each page is exactly **4KB** (4096 Bytes)
- The size of each key in your B+ tree is **128 bytes**
- The size of each pointer in your B+ tree is **8 bytes**
- Your system has **48GB** of free RAM and **infinite** hard disk space
- Assume a B+ tree node must fit into a single page.
- The constant fan-out f of the B+ tree is **30**.

Let's look at how our B+ tree would look with the amount of data we need to index.

a) [4 points]

How many pages can we store in the first level? In the second level? In tenth level?

b) [8 points]

Let's plan out the space required for our index. How many levels do we need in our B+ tree? Compute the space required by each index level (round up to the first decimal place).

c) [8 points]

Assume that each level must either be completely on RAM or disk. Note that all data pages stay on the disk. What is the worst-case IO requirement (number of disk accesses) to access a record?

Solution:

a) Number of Pages in Each Level

The number of pages we can store in each level of the B+ tree is determined by the fan-out of the tree. The fan-out is the maximum number of children that a node can have. In this case, the fan-out is 66.

- First Level: The first level (root) can store 1 page.
- Second Level: The second level can store f pages, where f is the fan-out. So, it can store 66 pages.
- Tenth Level: The tenth level can store f^9 pages, where f is the fan-out. So, it can store 30^9 pages.

b) Number of Levels and Space Required

The number of levels in the B+ tree is determined by the number of rows in the database and the fan-out of the tree. The formula to calculate the number of levels is log_f(N), where N is the number of rows and f is the fan-out. So, the number of levels is log_30(2 billion).

The space required by each index level is determined by the number of pages in that level and the size of each page. The formula to calculate the space required is number of pages * size of each page.

c) Worst-Case IO Requirement

The worst-case IO requirement is the number of disk accesses required to access a record. This is determined by the number of levels in the B+ tree, as each level requires a disk access. So, the worst-case IO requirement is equal to the number of levels in the B+ tree.

Problem 3 (30 pts)

This problem will explore different join implementations and the associated IO costs for different models. Let R(a, b), S(b, c), and T(c, d) be tables. For the purpose of this question, use the values provided below.

- P(R) = number of pages of R = 20
- T(R) = number of tuples of R = 1600
- P(S) = number of pages of S = 200
- T(S) = number of tuples of S = 15000
- P(T) = number of pages of T = 2000
- P(R, S) = number of pages in output RS = 100
- P(S, T) = number of pages in output ST = 1000
- P(R, S, T) = number of pages in output RST = 500
- B = number of buffer pages = 32

a) [3 point]

Let us start by considering a simple nested loop join. Compute the IO cost for a simple nested loop join if R is the ”outer loop” and S is the ”inner loop.”

b) [3 point]

Compute the IO cost for a simple nested loop join if S is the ”outer loop” and R is the ”inner loop.”

c) [8 points]

Now consider using a block nested loop join. Compute the IO cost for joining R, S and then joining the result with T. Then compute the IO cost for joining S, T and then joining the result with R.

d) [3 points]

Now consider using a sort-merge join. Compute the IO cost for joining R, S and then joining the result with T. Assume that the tables are not sorted before starting. Also assume that we do not need to do any back up as described in lecture.

e) [4 points]

Again, using a sort-merge join, compute the IO cost for joining S, T and then joining the result with R. Assume that the tables are not sorted before starting. Also assume that we do not need to do any back up as described in lecture.

f) [3 points]

Now suppose we only want to join R and S (with sort-merge join) but this time all values for the join attribute are the same. What would be the IO cost now?

g) [3 points]

Now consider using a hash join. Compute the IO cost for joining R, S and then joining the result with T.

h) [3 points]

Again, using a hash join, compute the IO cost for joining S, T and then joining the result with R.

Solution:

a) IO = P(R)+T(R)*P(S)+out = P(R)+T(R)*P(S)+P(R,S) = 20+1,600*200 +100 = 320,120

b) IO = P(S) + T(S)*P(R)+out = P(S) + T(S)*P(R)+P(R,S) = 200+15,000*20 +100 = 300,300

c) $IO_{R,S}$ = P(R)+P(R)*P(S)/B+P(R,S) =20+20*200/32+100=245

   $IO_{RS,T}$ = P(R,S) + P(R,S)*P(T)/B +P(R,S,T) = 100+100*2000/32+500 = 6850

   $IO_{S,T}$ = P(S)+P(S)*P(T)/B+P(S,T) = 200+200*2000/32+1000 = 13600

   $IO_{ST,R}$ = P(S,T) +P(S,T)*P(R)/B+P(S,T,R) = 1000+1000*20/32+500 = 2125

d) $IO_{R,S}$ = Sort(P(R)) + Sort(P(S))+P(R)+P(S)+P(R,S) = 4*20 +4*200 +20+200 +100 =1200

   Sort(N pages) = 2N*ceil($\log_B$(N/2B))+2N

   ceil($\log_B$(N/2B)) is replaced with 1 if N<B

   $IO_{RS,T}$ = Sort(P(R,S)) + Sort(P(T))+P(R,S)+P(T)+P(R,S,T)

e) Similar to d)

f) $IO_{R,S}$ = Sort(P(R)) + Sort(P(S))+P(R)*P(S) +P(R,S)

g) $IO_{R,S}$ =3(P(R)+P(S))+P(R,S) = 660+100=760

$IO_{RS,T}$ = 3(P(R,S)+P(T))+ P(R,S,T) = 6300+500=6800

Problem 4 (10 pts)

Suppose we want to create a linear hash file with a file load factor of 0.7 and a blocking factor of 20 records per bucket, which is to contain 112,000 records initially.

(a) How many buckets should we allocate in primary areas?

(b) What should be the number of bits used for bucket addresses?

Solution:

(a) No. of buckets in primary area.

= 112000/(20*0.7) = 8000.

(b) K: the number of bits used for bucket addresses

2K < = 8000 < = 2 k+1

2 12 = 4096

2 13 = 8192

K = 12

Boundary Value = 8000 - 2 12

= 8000 - 4096

= 3904

10 points will be awarded for following the homework guidelines document.