CMPE 180A: DBMS Homework 2

Name: Sakshat Patil

SJSU ID: 018318287

Homework Number 2

Due Date: 26th March, 2025

Problem 1

a) Total capacity of a track = (Block Size + Interblock Gap) Number of blocks
   Total capacity of a track = 12800 Bytes

   Useful capacity of track = Block Size * Number of Blocks
   Useful capacity of track = 10240 Bytes

b) Number of cylinders = Number of tracks per surface

   Number of cylinders = 400

c) Total capacity of a cylinder = Total capacity of a track * Number of tracks in a cylinder
   Total capacity = 12800 * 15 * 2
   Total capacity = 384000 Bytes

   Useful capacity of a cylinder = Useful capacity of track* Number of tracks in a cylinder
   Useful capacity of a cylinder = 307200 Bytes

d) Total capacity of a disk pack = total capacity of a cylinder in the disk pack * Number of cylinders in a disk pack
   Total capacity = 307200 * 400
   Total capacity = 153600000 Bytes

   Useful capacity of a disk pack = Useful capacity of a cylinder in the disk pack * Number of cylinders in a disk pack
   Useful capacity of a disk pack = 30720 * 400

Useful capacity of a disk pack = 12288000 Bytes

e) Transfer rate = Total capacity of track / (60*1000/rotation speed)
Transfer rate = 512 bytes/msec

Block Transfer Time (btt) = Block size in bytes / track rate
Btt = 1 msec

Rotational delay = (1/2) * (1/speed of disk rotation) *1000
Rotational delay = 0.208 msec

Bulk transfer rate = (Block Size/(Block size + Gap Size)) * transfer rate
Bulk transfer rate = 409.6 bytes/msec = 0.41 MB/s

f) Total time = seek time + block transfer time + rotational delay

Total time = 30 + 1.25 + 12.5

Total time = 43.75msec

g) Time to transfer 20 blocks = 20 * time to transfer a block
Time to transfer 20 blocks = 624.16 msec

Time to transfer 20 blocks using double buffering = seek time + rotational delay + (20 * block transfer time)
Time to transfer 20 blocks using double buffering = 30 + 1 + (20*0.208)
Time to transfer 20 blocks using double buffering = 35.16

Problem 2:

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

No. of pages in Level 1= 30^ (1-1) = 1 page
No. of pages in Level 2 = 30 ^ (2-1) = 30 pages
No. of pages in Level 3 = 30 ^ (3-1) = 900 pages
No. of pages in Level 10 = 30^ (10-1) = 30^9 pages
Number of leaf nodes = (2*10^9)/30 = 66,666,667
(fan out)^(h-1) >= 66,666,667
Taking log on both sides
ln(30)*(h-1) >=ln(6666667)

H = 7
The level of numbers required are 7

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

Number of nodes at each level –

Level 7:
Space required = (2,000,000,000 / 30)* 4096 bytes
Space required = 273,066,668,032 bytes
Space required = 254 GB

Level 6:
Space required = (66,666,667/30) * 4096 bytes
Space required = 8.5 GB

Level 5:
Space required = (2,222,223/30) * 4096 bytes
Space required = 289 MB

Level 4:
Space required = (74075/30) * 4096 bytes
Space required = 9.6 MB

Level 3:
Space required = (2470/30) * 4096 bytes

Space required = 332 KB

Level 2:
Space required = 32/30 * 4096 bytes
Space required = 12 KB

Level 1 (Root Node):

c) 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?

We have 48 GB of RAM. Therefore, we can fit Levels 1-6 in the RAM. Level 7 needs to be in the hard disk space. Hence, all leaf nodes will need to be in the disk space.

The worst case I/O requirement is 2 disk accesses. One for the leaf node and one for the pages.

Problem 3:

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) 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.”

IO Cost=P(R)+T(R)×P(S)
IO Cost=20+1600×200=20+320000=320020 IOs.

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

IO Cost=P(S)+T(S)×P(R)
IO Cost=200+15000×20=200+300000=300200 IOs.

C) 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.

Joining R and S

We consider R as the outer relation as it is smaller

Avaiable buffer = B = 32

B-2 = 30

Number of Chunks for R = 20/30 =1

I/O cost for R cross S:

$P(R)+\lceil 20/30 \rceil \times P(S)=20+1\times 200=220$ I/Os.

The join output has P(R,S)=100 pages.

Joining (R,S) with T:

Number of chunks for (R,S):

[100/30] = [3.33] = 4

I/O cost for for (R,S) cross T:

$P(R,S)+4\times P(T)=100+4\times 2000=100+8000=8100$ I/Os.

Total I/O Cost for Plan 1:

220+8100=8320 I/Os

Joining S with T:

Choosing S as the outer relation as it is less:

Number of chunks for S

[200/30] = [6.67] = 7

IO Cost = P(S)+7×P(T)=200+7×2000=200+14000=14200 I/Os.

The join output has P(S,T)=1000 pages.

Joining (S,T) with R:

Using the previous result with R.

Number of chunks for R = [20/30] = 1

I/O Cost = P(R)+1×P(S,T)=20+1000=1020 I/Os

Total Cost = 14200+1020=15220 I/Os.

D) 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.

Join R and S

Sorting S

P(R) = 20 pages

Cost to Sort = read 20 + write 20 = 40 I/Os

Sorting R

P(s) = 200 pages. The size is greater than 32 so we use external sorting

Phase 1: Read 200 pages and write out sorted runs: 200 + 200 = 400 I/Os.
Phase 2 (Merge Pass): Read the 200 pages and write the final sorted file: 200 + 200 = 400 I/Os.
Total sort cost = 400 + 400 = 800 IOs

Merge -Join R and S

I/O cost for scanning: 20+200=220 I/Os.
Merge-join cost: 220+100=320 I/Os.

Total Cost for Join R and S = 40 + 800 + 320 = 1160 IOs

Join (R,S) with T

P(T) = 2000 pages

T does not fit in memory, so we must use external sorting.

Phase 1: Read 2000 pages and write sorted runs: 2000 + 2000 = 4000 I/Os.

Phase 2 (Merge Pass):Read 2000 pages and write the final sorted file: 2000 + 2000 = 4000 I/Os.

Total sort cost for T: 4000 + 4000 = 8000 I/Os.

Merge-Join of (R,S) and T

Now both inputs are sorted:

Sorted (R,S) is 100 pages.

Sorted T is 2000 pages.

Cost to scan inputs: 100+2000=2100 I/Os.

The join output has P(R,S,T)=500 pages, costing 500 I/Os to write.

Merge-join cost: 2100+500=2600 I/Os.

Total Cost for Join (R,S) and T Sort T: 8000 Merge join: 2600 8000+2600=10600 I/Os.

Overall Total IO Cost

(Join R and S): 1160 I/Os

(Join (R,S) withT): 10600 I/Os

Grand Total: 1160+10600=11760 I/Os.

E) 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.

Joining S with T:

1. Sorting S

   P(S) = 200 pages

   Since P(S) > B, we require extra sorting

   Cost:

   a) Initial Pass: 200 r + 200 w = 400 I/Os
   b) Merge Pass: 200 r + 200 w = 400 I/Os
   c) Total for S: 400 + 400 = 800 I/Os

2. Sorting T

   P(T) = 2000

   Cost:

   d) Initial Pass: 2000 r + 2000 w = 4000 I/Os
   e) Merge Pass: 2000 r + 2000 w = 4000 I/Os
   f) Total for S: 4000 + 4000 = 8000 I/Os.

3. Merge – Join S and T

   a) Scan Cost = P(s) + P(T) = 200 + 2000 = 2200 I/Os
   b) Output write = P(S,T) = 1000 pages
   c) Merge join cost = 2200 + 1000 = 3200 I/Os

4. Total Cost = Cost of everything = 12000 IOs

Joining (S,T) with R

1. Sorting R
   a. P(R) = 20 pages
   b. Cost to read and write = 20 + 20 = 40 I/Os
2. Merge-Join R and ST

a. Scan cost

   R: 20pages

   ST: 1000 pages

   Total scan cost = 1020 I/Os

b. Merge join cost = 1020 + 500 I/Os

Cost(R, ST) = 40 + 1520 = 1560 I/Os

Grand Total = 13560 I/Os

F) 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?

Sorting R

R: 20 pages

- Memory available: 32 pages
- 20 < 32, so R fits in memory

Calculation:

- Read R: 20 I/Os
- Write sorted R: 20 I/Os
- Total R sorting: 20 + 20 = 40 I/Os

Sorting S

S: 200 pages

- Memory available: 32 pages
- 200 > 32, requires external sorting

Initial pass calculation:

- Read S: 200 I/Os
- Write sorted runs: 200 I/Os
- Initial pass total: 200 + 200 = 400 I/Os

Merge pass calculation:

- Read sorted runs: 200 I/Os
- Write merged result: 200 I/Os
- Merge pass total: 200 + 200 = 400 I/Os

Total S sorting: 400 + 400 = 800 I/Os

Merge-Join R and S

Scanning input:

- Read sorted R: 20 I/Os
- Read sorted S: 200 I/Os
- Scanning total: 20 + 200 = 220 I/Os

Output writing:

- Join result: 1,200,000 pages
- Writing output: 1,200,000 I/Os

Merge phase total:

- Scanning: 220 I/Os
- Writing output: 1,200,000 I/Os
- Total merge phase: 220 + 1,200,000 = 1,200,220 I/Os

Total I/O Cost

- Sort R: 40 I/Os

- Sort S: 800 I/Os
- Merge join: 1,200,220 I/Os

Total: 40 + 800 + 1,200,220 = 1,201,060 I/Os

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

Joining R and S

Given:

- R: 20 pages
- S: 200 pages
- Output (R,S): 100 pages

Cost Breakdown: Partition Phase:

- Read R and S: 20 + 200 = 220 I/Os
- Write partitions: 20 + 200 = 220 I/Os

Probing Phase:

- Read partitions: 220 I/Os

Output Writing:

- Write join result: 100 I/Os

Calculation:

- Total I/O cost: 220 + 220 + 220 + 100 = 760 I/Os

Verification:

- Shortcut formula: 3 × (P(R) + P(S)) + P(R,S)
- 3 × (20 + 200) + 100

- 3 × 220 + 100
- 660 + 100 = 760 I/Os

Joining (R,S) with T

Given:

- (R,S): 100 pages
- T: 2000 pages
- Output (R,S,T): 500 pages

Cost Breakdown: Partition Phase:

- Read (R,S) and T: 100 + 2000 = 2100 I/Os
- Write partitions: 100 + 2000 = 2100 I/Os

Probing Phase:

- Read partitions: 2100 I/Os

Output Writing:

- Write join result: 500 I/Os

Calculation:

- Total I/O cost: 2100 + 2100 + 2100 + 500 = 6800 I/Os

Verification:

- Shortcut formula: 3 × (P(R,S) + P(T)) + P(R,S,T)
- 3 × (100 + 2000) + 500
- 3 × 2100 + 500
- 6300 + 500 = 6800 I/Os

Overall I/O Cost for (R,S) and T

- R join S: 760 I/Os
- (R,S) join T: 6800 I/Os
- Total: 760 + 6800 = 7560 I/Os

Joining S, T, and R

S: 200 pages

- T: 2000 pages
- Output (S,T): 1000 pages
- R: 20 pages
- Output (R,S,T): 500 pages

Phase 1: Join S and T

Calculation:

- 3 × (P(S) + P(T)) + P(S,T)
- 3 × (200 + 2000) + 1000
- 3 × 2200 + 1000
- 6600 + 1000 = 7600 I/Os

Phase 2: Join R with (S,T)

Calculation:

- 3 × (P(R) + P(S,T)) + P(R,S,T)
- 3 × (20 + 1000) + 500
- 3 × 1020 + 500
- 3060 + 500 = 3560 I/Os

Total I/O Cost

- S join T: 7600 I/Os
- R join (S,T): 3560 I/Os
- Total: 7600 + 3560 = 11,160 I/Os

Problem 4

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?

Determining Hash Table Capacity

Load Factor Calculation:

- Load factor (α) = Number of records / Total capacity
- Given:
  - Load factor (α) = 0.7
  - Blocking factor = 20 records per bucket
  - Total records = 112,000

Number of Buckets Calculation:

- Equation: 112,000 = 0.7 × (Number of buckets × 20)
- Solve for number of buckets:
  - Number of buckets = 112,000 ÷ (0.7 × 20)
  - Number of buckets = 112,000 ÷ 14
  - Number of buckets = 8,000

Bucket Address Bits:

- Requirement: $2^b \geq 8,000$
- Calculation:

    - $\log_2(8{,}000) \approx 12.97$
    - Round up: 13 bits

Results:

- Primary area buckets: 8,000
- Bucket address bits: 13