Assignment 8

Part – II Physical Database Organization and Algorithms

Question 6

We have the following parameters.

block size = 4096 bytes

block-address size = 9 bytes

block access time = 10 ms (microseconds)

record size = 200 bytes

record key size = 12 bytes

6.a)

For minimum time, we need maximum branching factor. First, we will find order n of the B+ tree.

(n+1) * | blockaddress | + n* |Key| <= blocksize (from lectures)

(n+1) * 9 + n* 12 <=4096

9n+9+12n<=4096

21n<=4096-9

n <=194.6

n ≈ 194

Max_keys= 194, Max_ptr= 195

The min time = $(\log_{n+1}10^8 + 1) * 10ms = (\log_{195}10^8 + 1) * 10 \approx 50$ ms

The plus 1 is for extra read for a record in data file, which spends a block access.

6.b)

For maximum time, we need minimum branching factor.

Min branching factor = n/2 + 1 = 194/2 +1 = 98

At root, there is min 2 branching factor. We need one block access for that. So, after that we will have $10^8 / 2 = 5 * 10^7$ nodes.

Height of the tree = $\text{Log}_{98}(5 * 10^7) \approx 4$. So, 4 block access for the depth of tree.

Once again, we will need 1 block access to get the record in data file.

Hence maximum time is: (4+1+1) * 10 ms = 60 ms

Question 9

We have,

R = 1500000

S = 5000

B(R) = 1500000 / 30 = 50000

B(S) = 5000 / 10 = 500

Also,

M + delta M = 101

Delta M is 1 usually, which is fudge factor and is used to accommodate the algorithm. 1 can be used for output block.

Hence M = 100

9.a) Block Nested Loop Join

Considering R to be an outer relation, we can calculate using formula

B(R) + (B(R) * B(S) /M) = 50000 + (50000 * 500 /100) = 50000 + 250000 = 300000

Considering S to be an outer relation, we can calculate using formula

B(S) + (B(R) * B(S) /M) = 500 + (50000 * 500 /100) = 500+ 250000 = 250500

9.b) Sort-Merge Join

It can be calculate using,

$B(R) + B(S) + 2\ B(R)\ ceil(\log_M(B(R))) + 2B(S)\ ceil(\log_M(B(S)))$

$= 50000 + 500 + 2* 50000 * \log_{100}(50000) + 2* 500 * \log_{100}(500)$

$= 50500 + (100000 * 3) + (1000 * 2)$

$= 352500$

First Part of calculation B(R) + B(S) is for merge phase.

Second part $2\ B(R)\ ceil(\log_M(B(R))) + 2B(S)\ ceil(\log_M(B(S)))$ is for external sorting phase.

9.c) Using p different B-Values

The external sorting phase would remain the same for any value of p.

= 300000 + 2000 = 302000.

Consider p=1, here we need to do block nested loop join because all the records with same B value can not fit into a memory. Since, S is a smaller relation, we will consider S to be an outer relation.

Number of block IO's = B(R) + (B(R) * B(S) /M) = 500 + (50000*500/100) = 250500

Overall Number of block IO's = 302000 + 250500 = 552500

Consider p=2, since we have 2 B-values, we need 2 block nested loop join for B(R) and B(S)

Number of block IO's = 500/2 + (50000*500)/ (2*2*100) = 250 + 62500 = 62750.

Hence, together we will need p* 62750 = 2 * 62750 = 125500

Overall Number of block IO's = 302000 + 125500 = 427500

Consider p=3, we have 3 B-values, so 3 block nested loop join.

Number of block IO's = 500/3 + (50000*500)/ (3*3*100) = 166.6 + 27777.7 = 27944.3 ≈ 27945

Hence, together we will need p* 27945 = 3 * 27945 = 83835

Similarly, we can calculate for further p different B-Values.

9.d) Hash-Join

Hash phase costs 2*B(R)+2*B(S)

Merge phase costs B(R) + B(S)

Total: 3*(B(R)+B(S))

(Given in lectures, assuming for given any B value, buffer is large enough to store all records from R and S.

= 3 * (50000 + 500) = 3*50500

=151500

Question 10

1) This is conflict-serializable.

Since all these are read operations, we can move them around. Also, there are no conflicting pairs. Hence, we can swap all T1 to front, so that we will get serial schedule. Like the one given below which follows: T1 - > T2

Serial Schedule for S1 is: **$R_1(x)R_1(z)\ R_1(y)R_2(y)R_2(x)$**

2) This is not conflict-serializable. If we look at the precedence graph, it forms a cycle for T1 and T2. Because, $R_1(x)$ is followed by $W_2(x)$ and $W_2(y)$ is followed by $R_1(y)$.

3) This is conflict-serializable.

P(S3) = { (T1, T3), (T2, T1), (T2, T3) }

If we look at the precedence graph, it is acyclic which means S3 is serializable. We can swap all the T2 transactions to the front so that we will get a serial schedule like T2 -> T1 -> T3

Serial Schedule for S1 is: **$W_2(x)$ $R_2(z)$ $R_2(y)$ $R_1(z)$ $W_1(x)$ $W_1(y)$ $W_3(z)$ $R_3(x)$**

Question 11

Transactions can be:

T1: $R_1(x)$ $W_1(y)$ $R_1(z)$

T2: $W_2(x)$ $R_2(y)$

T3: $W_3(z)$

Schedule S based on graph is:

$R_1(x)$ $W_2(x)$ $R_2(y)$ $W_1(y)$ $R_1(z)$ $W_3(z)$ $R_2(z)$

Question 12

T1: $R_1(x)$ $W_1(x)$

T2: $R_2(y)$ $W_2(y)$

T3: $R_3(z)$ $W_3(z)$

This is simplest one, each involve the read and write operations and there are no conflicting pairs. Hence any schedule over these transactions can be converted into any serial schedule using proper swap operations.

Question 13

13.a)

For T1 and T2, there are two ways in which it will be a serial schedule.

1) For T1 -> T2: We get values A=0 and B=1 after the execution

2) For T2-> T1: We get values A=1 and B=0 after the execution

Hence, for both at least one of the values is zero. Hence, consistency requirement is preserved.

13.b)

For this, we can put T2 inside T1 transactions or vice versa to make a non-serializable schedule.

T1	T2
	R(B)
R(A)	
R(B)	
if A = 0 then B := B+1;	
W(B)	
	R(A)
	if B = 0 then A := A+1;
	W(A)

S: $R_2(B)$ $R_1(A)$ $R_1(B)$ $W_1(B)$ $R_2(A)$ $W_2(A)$

13.c)

No. There is no non-serial schedule on T1 and T2 that produces a serializable schedule.

Because, consider schedule starts with $R_1(A)$, this will conflict with $W_2(A)$. Consider $R_1(A)$ $R_2(B)$, here B will conflict with $W_1(B)$, i.e. we will get a cyclic precedence graph. Again, if we consider $R_1(A)$ $R_1(B)$, then take $R_2(B)$, we will get a cyclic precedence graph. Hence, we will have to first execute T1 then T2 if we want to start our schedule with $R_1(A)$.

Similarly, the same scenario applies if we schedule $R_1(B)$. Hence, the answer is no.