Assignment 8

Pauravi Wagh

5/1/2020

Question 6

For the given problem, we first need to find the order of the B+ tree,
using the formula,

$$n \leq \frac{blocksize - |blockaddress|}{|blockaddress| + |key|}$$

Our values are
block size = 4096 bytes
block-address size = 9 bytes
block access time = 10 ms (micro seconds)
record size = 200 bytes
record key size = 12 bytes

Substituting,

$$n \leq \frac{4096 - 9}{9 + 12}$$
$$n = 194.619$$
$$n = 194$$

6
a)
For finding the minimum tree, we need the maximum possible spread of the tree. Thus we take n = 195.
The minimum time is $ceil(log_{195}(10^8 * 10^6)) + 1 = ceil(6.1134447185) + 1 = 7 + 1$
$= 8 * blockaccesstime = 8 * 10$
$= 80msec$

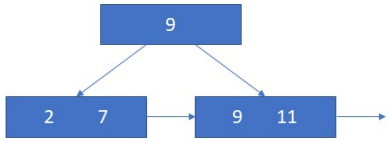
b)

Now to find the maximum time, the branching factor must be minimum. So the branching factor will be $ceil(\frac{194}{2}) + 1 = 98$
For the maximum time, at the root node let us consider the branching factor to be 2. Thus tree will be half of all the possible nodes i.e. $10^8 * 10^6/2 = 5 * 10^{13}$

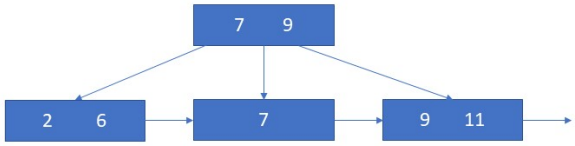
Thus the height of thus tree will be,
$ceil(log_{98}(5 * 10^{13})) = 6.8796658358 = 7 + 1 = 8$

The maximum time to retrieve a record with key k is height of the tree * block access time = 8*10 = 80 msec

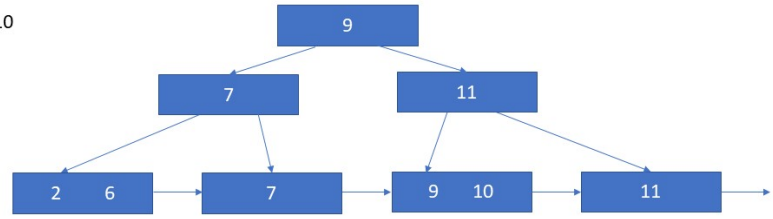
Question 7 a



insert 6

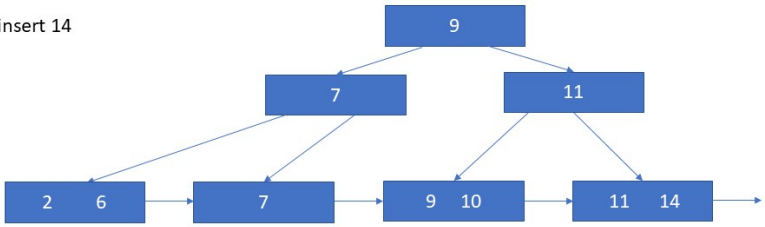


insert 10

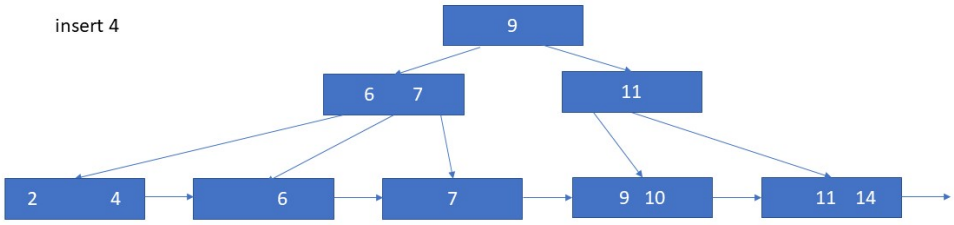


Question 7 a

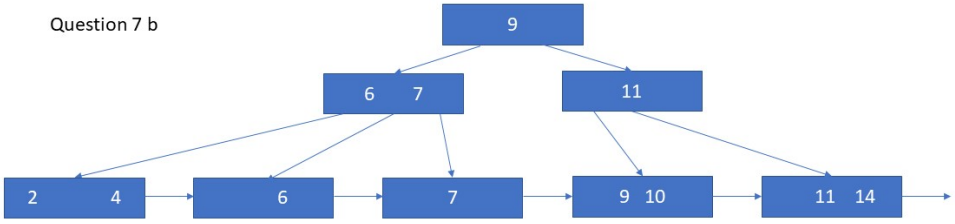
insert 14



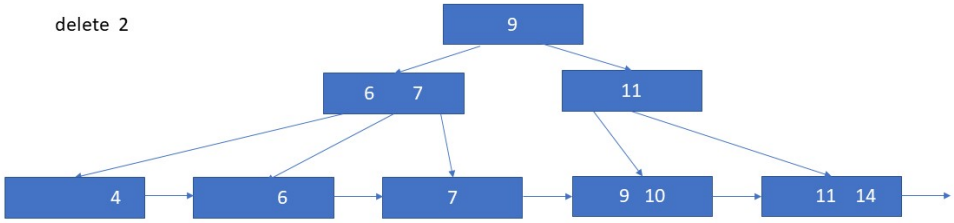
insert 4



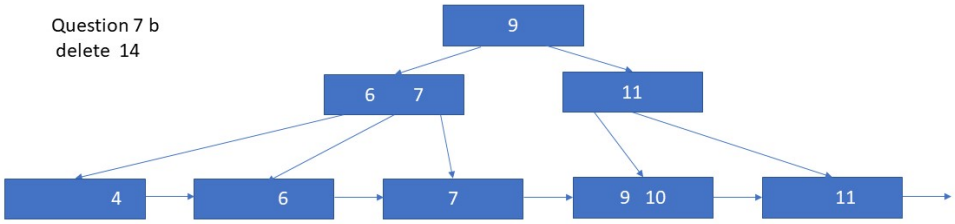
Question 7 b



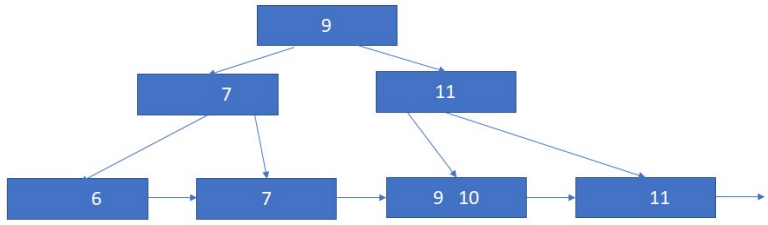
delete 2



Question 7 b
delete 14

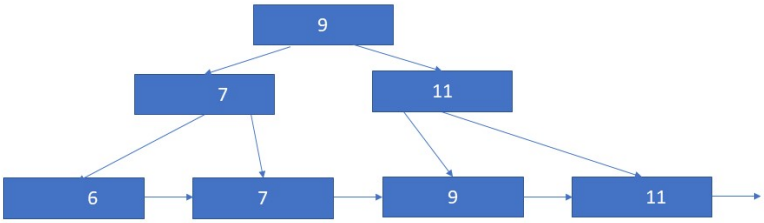


delete 4



Question 7 b

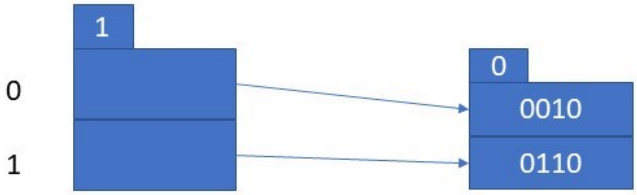
Delete 10



Question 8 a

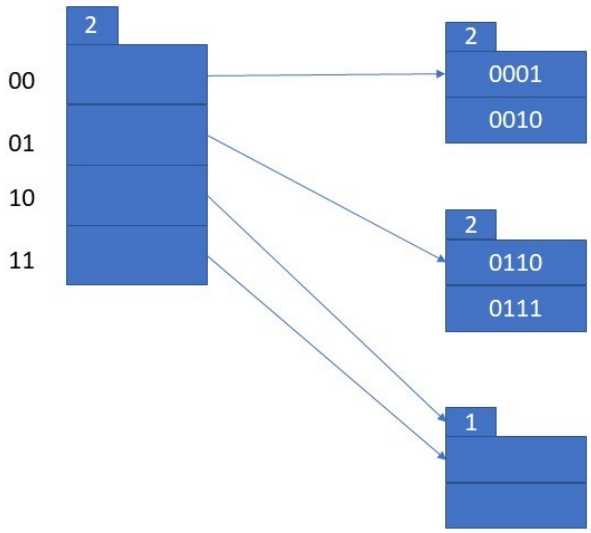
i) After adding 2 and 6

0010 and 0110

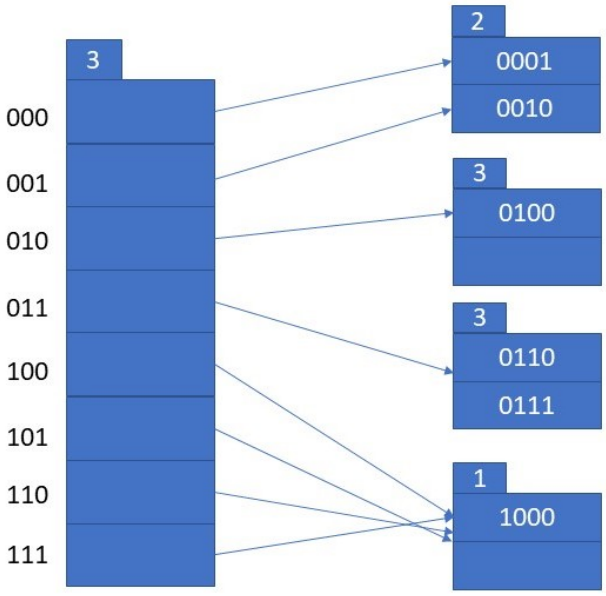


Question 8 a

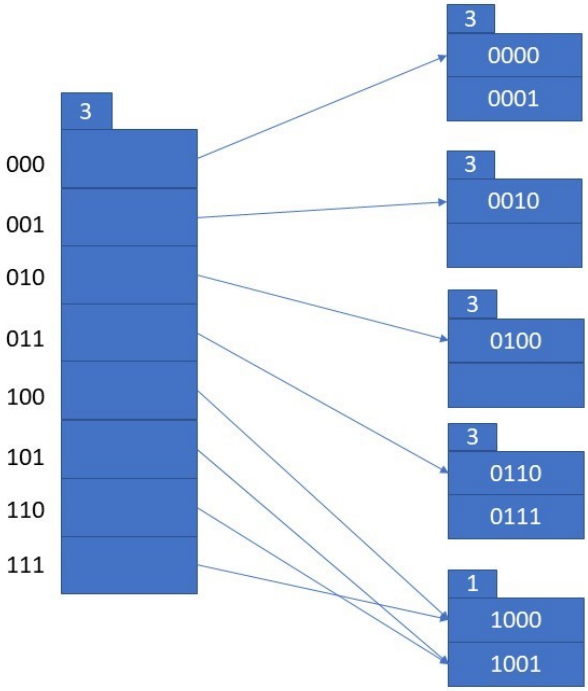
ii) After adding 1 and 7
0001 and 0111



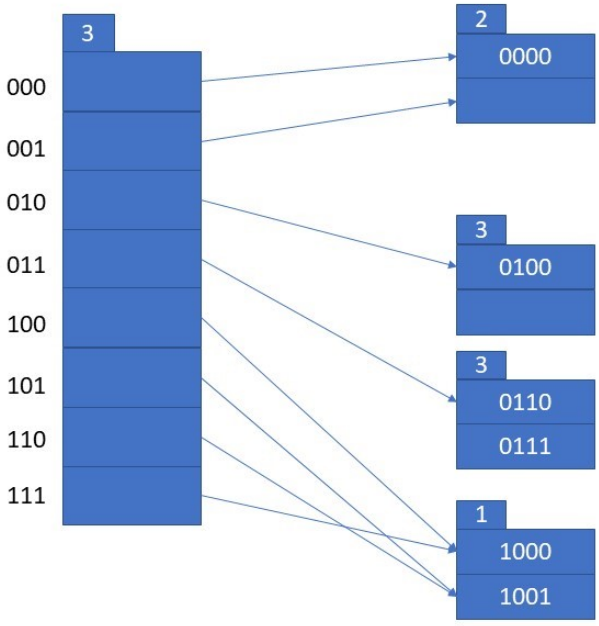
Question 8 a
iii) After adding 4 and 8
0100 and 1000



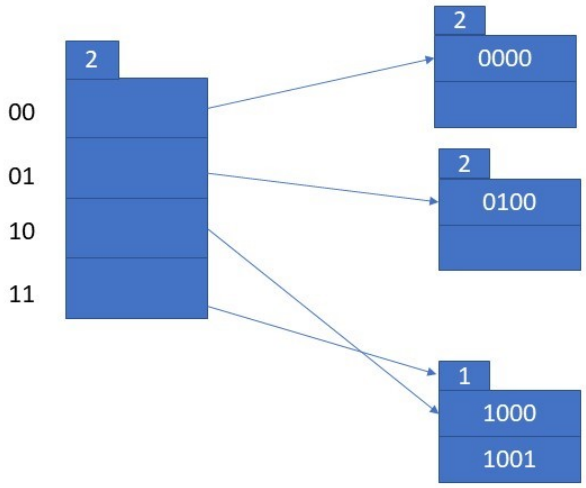
Question 8 a
iv) After adding 0 and 9
0000 and 1001



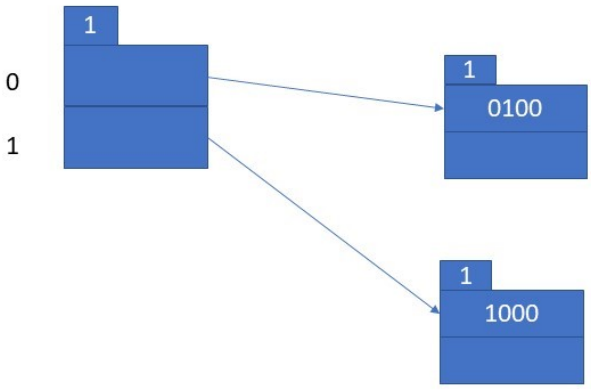
Question 8 b
i) After deleting 1 and 2
0001 and 0010



Question 8 b
ii) After deleting 6 and 7
0110 and 0111



Question 8 b
iii) After deleting 0 and 9
0000 and 1001



Question 9

R has 1,500,000 records
S has 5000 records

Records of R can fit in a block : 30 Records of S can fit in a block : 10

Number of blocks of R :

$$b(R) = \frac{number\ of\ records}{records\ in\ one\ block}$$

$$b(R) = \frac{1500000}{30}$$

$$b(R) = 50000$$

Number of blocks of S :

$$b(S) = \frac{number\ of\ records}{records\ in\ one\ block}$$

$$b(S) = \frac{5000}{10}$$

$$b(S) = 500$$

Main-memory buffer contains 101 blocks. We can consider this as $M + \Delta M$. So if we use 1 block as ΔM. Then our main memory consists of 100 blocks.

a) Block nested-loops join algorithm:
With R as the outer loop,

$$IOs\ necessary = b(R) + \frac{b(R) * b(S)}{M}$$

$$IOs\ necessary = 50000 + \frac{500 * 50000}{100}$$

$$IOs\ necessary = 300000$$

Using S as the outer loop will reduce the IO operations.
With S as the outer loop,

$$IOs\ necessary = b(S) + \frac{b(S) * b(R)}{M}$$

$$IOs\ necessary = 500 + \frac{500 * 50000}{100}$$

$$IOs\ necessary = 250500$$

b) Sort-merge join algorithm

$$IOs\ necessary = b(R) + b(S) + 2 * b(R) * ceil(log_M(b(R))) + 2 * b(S) * ceil(log_M(b(S)))$$

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

$$IOs\ necessary = 50000 + 500 + 300000 + 2000$$

$$IOs\ necessary = 352500$$

c) In this case, we can see that the p-values can be varied from 1...n. We will first sort all the values and then perform block nested loop on them.
So for p = 1,
We see that there is only one value of B in both the tables and this is the worst case scenario. So, the number of IO operations apart from sorting will be as per the block nested-algorithm. Since, b(S) is a smaller set, I will use it as the outer loop

$$IOs\ necessary = 2 * b(R) * ceil(log_M(b(R))) + 2 * b(S) * ceil(log_M(b(S))) + b(s) + \frac{b(S) * b(R)}{M}$$

$$IOs\ necessary = 300000 + 2000 + 500 + \frac{500 * 50000}{100}$$

$$IOs\ necessary = 552500$$

For p = 2,
We see that now we will have to perform the block-nested loop twice. since, the values are uniformly divided, $b(R) = 50000/2 = 25000$ and $b(S) = 500/2 = 250$.
Thus the operation performed will be, (keeping S as the outer loop),

$$IOs\ necessary = 2 * b(R) * ceil(log_M(b(R))) + 2 * b(S) * ceil(log_M(b(S))) + (b(s) + \frac{b(S) * b(R)}{M}) * p$$

$$IOs\ necessary = 300000 + 2000 + (250 + \frac{250 * 25000}{100}) * 2$$

$$IOs\ necessary = 427500$$

For p = 3,
We see that now we will have to perform the block-nested loop thrice since, the values are uniformly divided, $b(R) = 50000/3 = 16667$ and $b(S) = 500/3 = 167$.
Thus the operation performed will be, (keeping S as the outer loop),

$$IOs\ necessary = 2 * b(R) * ceil(log_M(b(R))) + 2 * b(S) * ceil(log_M(b(S))) + (b(s) + \frac{b(S) * b(R)}{M}) * p$$

$$IOs\ necessary = 300000 + 2000 + (167 + \frac{167 * 16667}{100}) * 3$$

$$IOs\ necessary = 300000 + 2000 + 84003$$

$$IOs\ necessary = 386003$$

For p-values above this, it is analogous.

d) Hash Join:

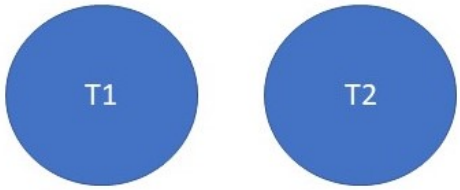
$$IOs\ necessary = 3 * (b(R) + b(S))$$

$$IOs\ necessary = 3 * (50000 + 500)$$

$$IOs\ necessary = 151500$$

Question 10)

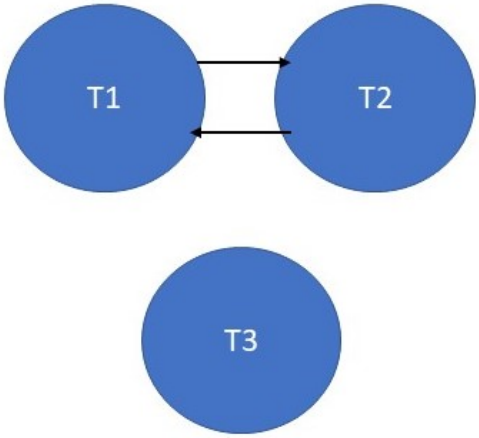
a) $S_1 = R_1(x)R_2(y)R_1(z)R_2(x)R_1(y)$



This schedule consists of read operations only. Read operations are non-conflicting operations. Since there is no conflict or a cycle, this schedule is conflict serializable.

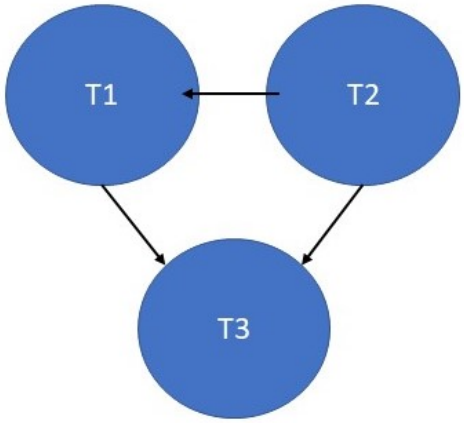
Conflict-equivalent Schedule:
$S_1' = R_1(x)R_1(z)R_1(y)R_2(y)R_2(x)$

b) $S_2 = R_1(x)W_2(y)R_1(z)R_3(z)W_2(x)R_1(y)$



We can see from the precedence graph that there is a cycle betwen transactions T_1 and T_2. Hence, we say that they are not conflict-serializable.

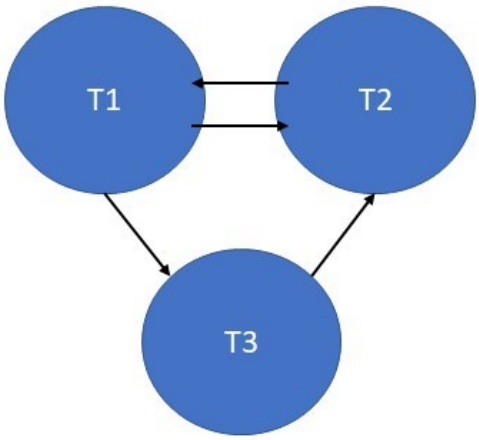
c) $S_3 = R_1(z)W_2(x)R_2(z)R_2(y)W_1(x)W_3(z)W_1(y)R_3(x)$



There is no cycle present. Hence this is conflict serializable schedule.

Conflict-equivalent Schedule:
$S_3' = W_2(x)R_2(z)R_2(y)W_1(x)R_1(z)W_1(y)W_3(z)R_3(x)$

Question 11)



The three transactions are:

$T_1 = R_1(x)W_1(y)$
$T_2 = W_2(x)R_2(y)$
$T_3 = W_3(y)$

The Schedule is :
$S = R_1(x)W_3(x)W_2(x)R_2(y)W_1(y)$

Question 12)

For, a schedule to be conflict serializable with all serial schedules, it needs to not have a conflict in the first place. Thus, all the transactions must work on different objects.

$T_1 = W_1(x)R_1(x)$
$T_2 = R_2(y)W_2(y)$
$T_3 = W_3(z)R_3(z)$

The conflict-equivalent schedule is
$S = W_1(x)R_1(x)R_2(y)W_2(y)W_3(z)R_3(z)$
The conflict equivalent schedules are:

$S = W_1(x)R_1(x)R_2(y)W_2(y)W_3(z)R_3(z)$

$S = W_1(x)R_1(x)W_3(z)R_3(z)R_2(y)W_2(y)$

$S = W_3(z)R_3(z)R_2(y)W_2(y)W_1(x)R_1(x)$

$S = W_3(z)R_3(z)W_1(x)R_1(x)R_2(y)W_2(y)$

$S = R_2(y)W_2(y)W_1(x)R_1(x)W_3(z)R_3(z)$

$S = R_2(y)W_2(y)W_3(z)R_3(z)W_1(x)R_1(x)$

Question 13 a)
For this question we have 2 serial ways in which we can execute these tasks.

1) T1T2

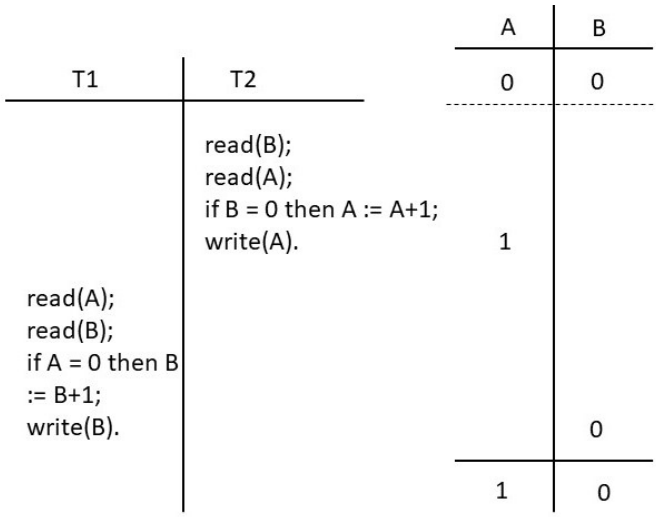
T1	T2	A	B
		0	0

read(A);
read(B);
if A = 0 then B
:= B+1;
write(B).
			1

	read(B);
	read(A);
	if B = 0 then A := A+1;
	write(A).
		0	

		0	1

2) T2T1

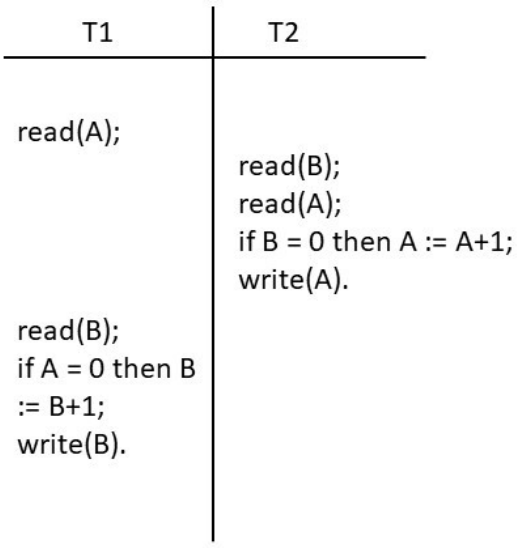


As we can see here, that the contrained is maintained. Thus, each serial schedule involving transaction T1 and T2 preserves the consistency requirement of the database.

Question 13 b)

For the schedule to be non-serializable, a presence of a cycle is enough to prove that.

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



Question 13 c)

In this question, without changing the order of the transactions, it is not possible to make a serializable schedule out of a non-serializable schedule. Using the transactions T_1, T_2 and T_3, we see that the read operations always happen before the writes and in no way can we resolve the conflict along with maintaining the contraint consistency. Since, changing the order of the operations, can cause semantic changes, we cannot change the order, hence, we cannot make a serializable schedule.