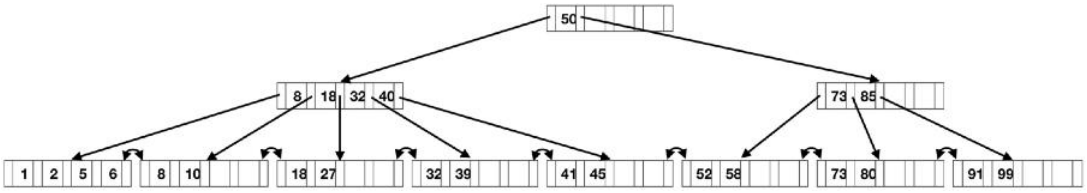
Name : Kovid Sharma
ZID : 5240067
COMP9311

Assignment 3

Question 1 (8 marks)

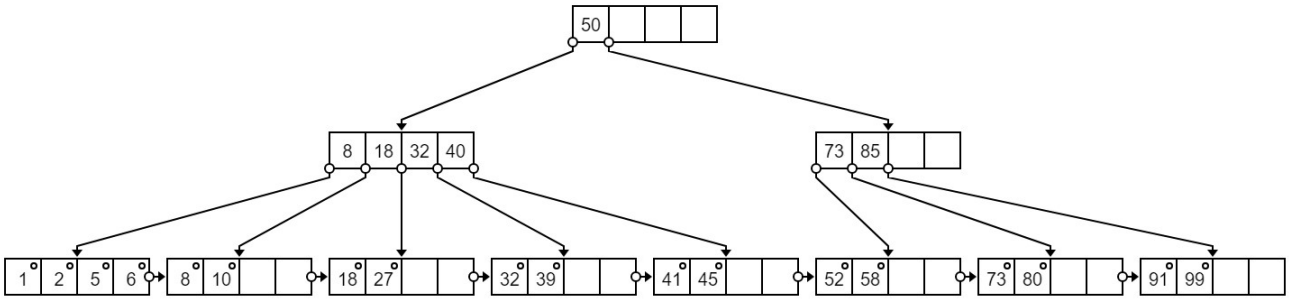
Consider the B+ tree shown in the following as an original tree.



Answer the following questions:

1) **(2 marks) There are currently 18 records in this tree. How many additional records could be added to this tree without changing its height (give the maximum possible number)?**

Ans.



For a n-order B+ tree with a height of h:

The maximum number of records stored is : $n^h - n^{h-1}$

For the given tree,

n = 5

h = 3

Total records = 125-25 = 100 number of records

Records present = 1, 2, 5, 6, 8, 10, 18, 27, 32, 39, 41, 45, 52, 58, 73, 80, 91, 99

18 records already present

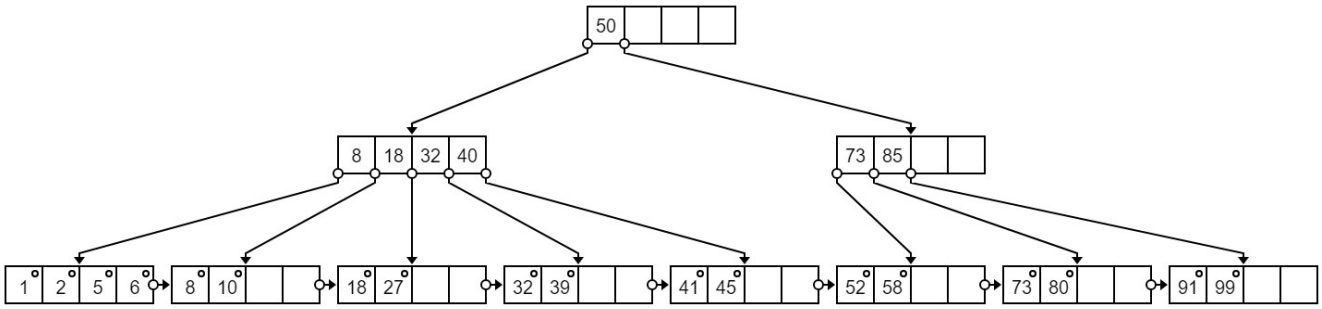
100-18 = 82

Only 82 more records can be added

2) (3 marks) Show the B+ tree after inserting a data entry with key 3 into the original tree.

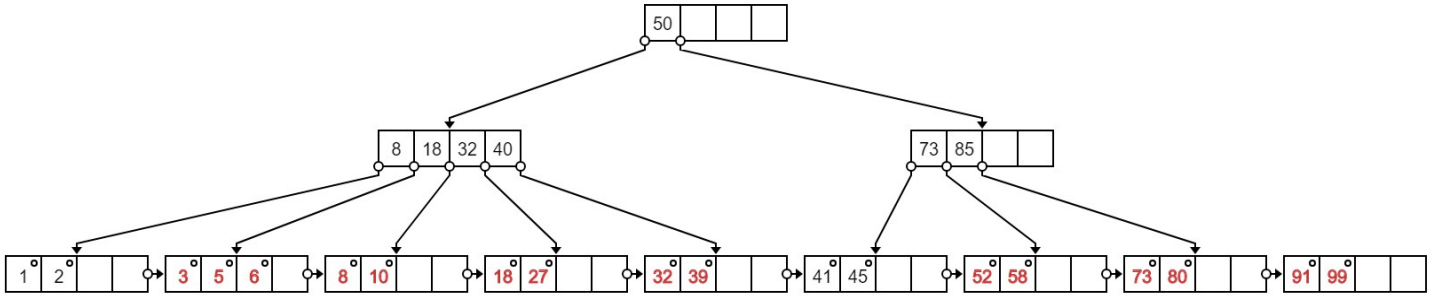
Ans.

Before insertion of 3:



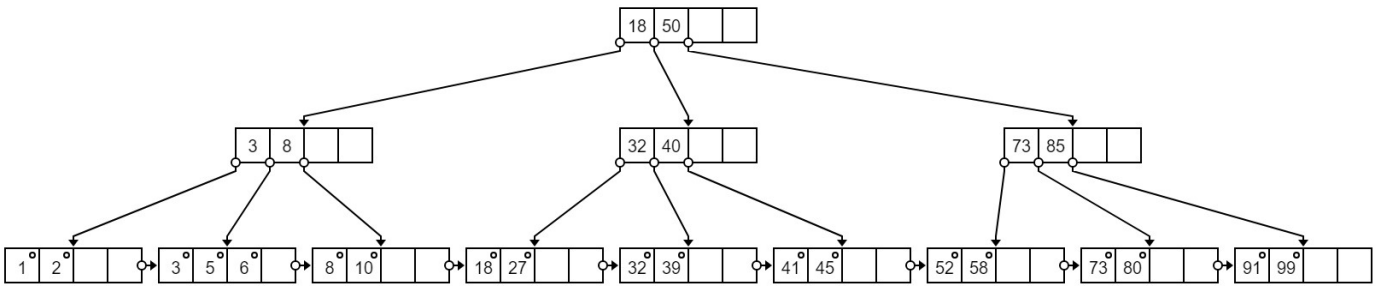
After insertion of 3:

Step 1: Find the correct leaf node to insert 3. 1,2,5,6 is the node. Adding 3 it becomes, 1,2,3,4,5. Now split it in the middle and send the middle element key up in step 2.



Step 2: The upper middle key node is 8,18,32,40. By moving 3 up, it becomes 3,8,18,32,40. Split it in the middle i.e. 18. Send 18 up to root node. Final B+ tree is formed as below

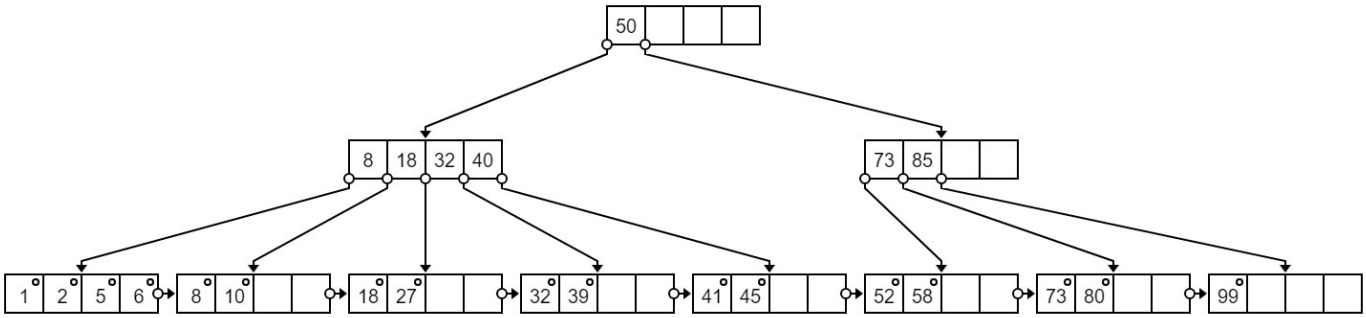
Final Ans:



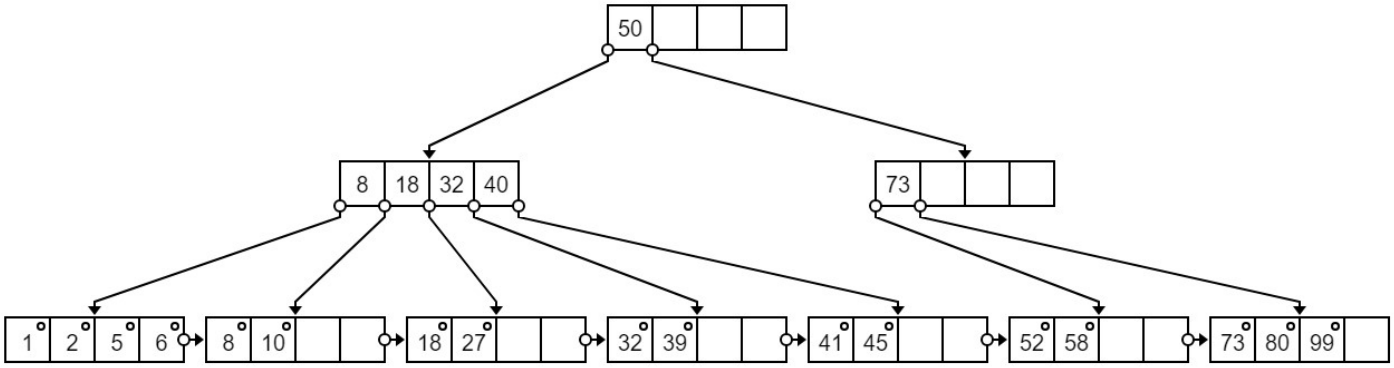
3) **(3 marks) Show the B+ tree after deleting the data entry with key 91 from the original tree.**

Deletion of 91:

Step 1 : Delete 99, but node is doesn't have 2 to 4 elements therefore redistribute and regroup in step 2

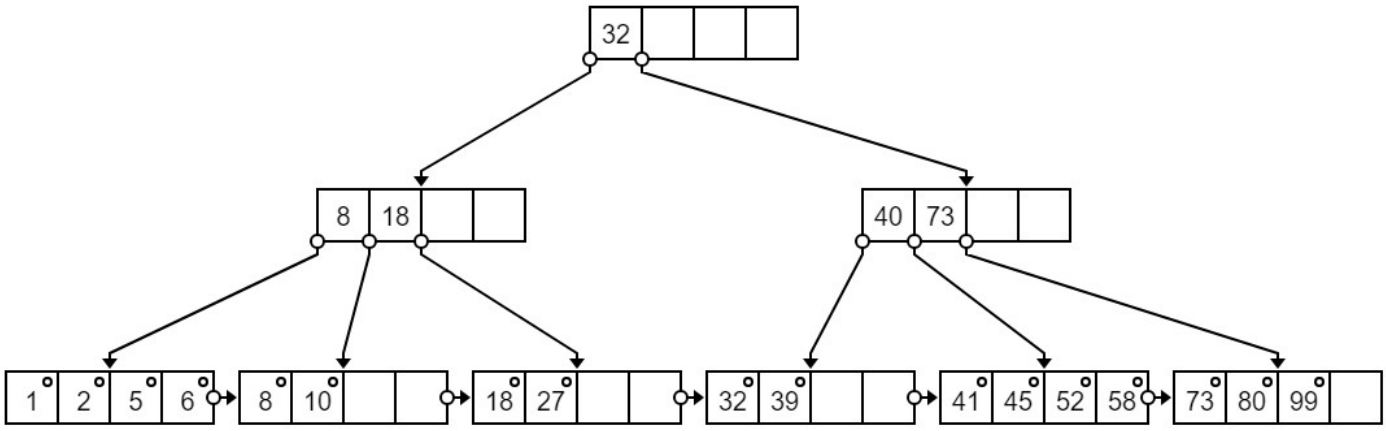


Step 2 : Regrouping 99 with it's sibling node, we have the following graph. Now the key indexing has shifted and the node 73 needs to be regrouped



Step 3: Nodes need to have minimum 2 to maximum 4 entries therefore 73 cannot be left alone. Merge 73 with 8,12,32,40. By splitting in the middle we take 32 up.

Final Ans:



Question 2 (4 marks)

Consider a relation R(a,b,c,d,e,f,g,h) containing 10,000,000 records, where each data page of the relation holds 10 records. R is organised as a sorted file with the search key R.a. Assume that R.a is a candidate key of R, with values lying in the range 0 to 9,999,999. For the relational algebra $\pi_{\{a,b\}}(\sigma_{(a>2{,}000{,}000 \text{ and } a<8{,}000{,}000)}(R))$, state which of the following approaches (or combination thereof) is the most likely to be the cheapest:

We assume that the database considers index-only plans. Index-only plans allow an index to contain all columns required to answer the query. It means that by using index-only plans, you will not have to access the data records in the file that contain the queried relations.

1. Access the sorted file for R directly.
2. Use a clustered B+ tree index on attribute R.a.
3. Use a clustered B+ tree index on attribute R.b.
4. Use a linear hashed index on attribute R.a.
5. Use a clustered B+ tree index on attributes (R.a, R.b).
6. Use a linear hashed index on attribute s (R.a, R.b).

Answer:
The problem at hand is t0 access of records within a range in a sorted file. The records are kept in a sorted order and we need to access the ones between 2 million and 8 million in the fastest way possible.

Accessing a sorted file directly is a very costly. We have better techniques to access a record in a sorted file so option 1 is out.

Hash indexes are best for equality selections. They cannot support range searches hence option 4 and 6 are out.

Tree structured indexes are best for sorted access and range queries.
We can ignore option 3 since it a B+ indexed on a non-key value.

Option 2, though being a candidate key and using B+ tree will fetch a faster result, but since our relational algebra says we need to access attributes a and b of the relation R, the best way to do it would be option 5. It is indexed on candidate key R.a and R.b helps the search become even faster.

Final Ans:
Use a clustered B+ tree index on attributes (R.a, R.b).

Question 3 (8 marks)

Consider the schedule below. Here, R(*) and W(*) stand for 'Read' and 'Write', respectively.
!T_1, !T_2, !T_3 and !T_4 represent four transactions and !$_i$ represents a time slot.

Time	!1	!2	!3	!4	!5	!6	!7	!8	!9	!10	!11	!12
T_1		R(B)				R(A)	W(B)		W(A)			
T_2			R(A)	W(A)								
T_3								R(B)		W(B)		
T_4	R(A)				W(A)						R(B)	W(B)

Each transaction begins at the time slot of its first Read, and commits right after its last Write (same time slot).

Regarding the following questions, give and justify your answers.

1) Assume a checkpoint is made between !$_4$ and !$_5$, what should be done to the four transactions when the crash happens between !$_7$ and !$_8$. (2 marks)

Ans:

Suppose A has originally value 100. T4 reads this value of A and then T2 also reads this value of A = 5. Suppose, T2 adds 20 to A and T4 adds 40 to A. Then at time slot 4, T2 rewrites A as 100 + 20 = 120. At time slot 5, T4 rewrites A as original value of A i.e. 100 + 40 = 140. The updated value of A from T2 has been lost. This is known as a lost-update problem.

Since there is a checkpoint after time slot 4, transaction T2 is committed and hence is unaffected by the crash between timeslot 7 and 8. T3 has no read writes before time slot 4 and it is also unaffected.

Transactions T1 and T4 need to be redone.
For T1 and T4, since the transaction was not committed, read of B has been lost. Therefore, both these operations need to be done after timeslot 4.

Final Ans:
Redo transactions T1(read B) and T4(read A). Transactions T2 and T3 remain unaffected.

2) Is the transaction schedule conflict serialisable? Give the precedence graph to justify your answer. (2 marks)

To construct a Schedule Graph $GS = (V, A)$ for a schedule S

1. A vertex in V represents a transaction.
2. For two vertices Ti and Tj, an arc $Ti \rightarrow Tj$ is added to A if

• there are two *conflicting* operations $O1 \in Ti$ and $O2 \in Tj$,

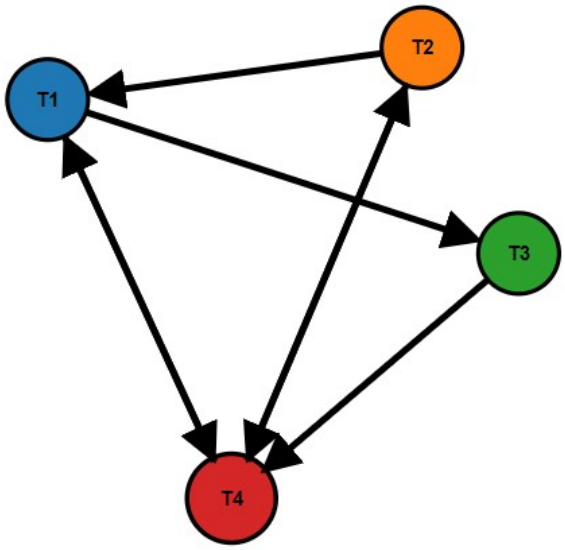
• in S, $O1$ is before $O2$.

Two operations $O1$ and $O2$ are *conflicting* if

• they are in different transactions but on the same data item,

• one of them must be a write.

For the Precedence Graph : r4(a) r1(b) r2(a) w2(a) w4(a) r1(a) w1(b) r3(b) w1(a) w3(b) r4(b) w4(b)



To check Conflict Serializability

Algorithm

Step 1: Construct a *schedule* (or *precedence*) graph – a *directed graph*.

Step 2: Check if the graph is *cyclic*:

• Cyclic: non-serializable.

• Acyclic: serializable.

Ans:

Schedule is not conflict serializable because its Conflict Graph is cyclic.

3) Construct a schedule (which is different from above) of these four transactions which causes deadlock when using two-phase locking protocol. If no such schedule exists, explain why. (2 marks)

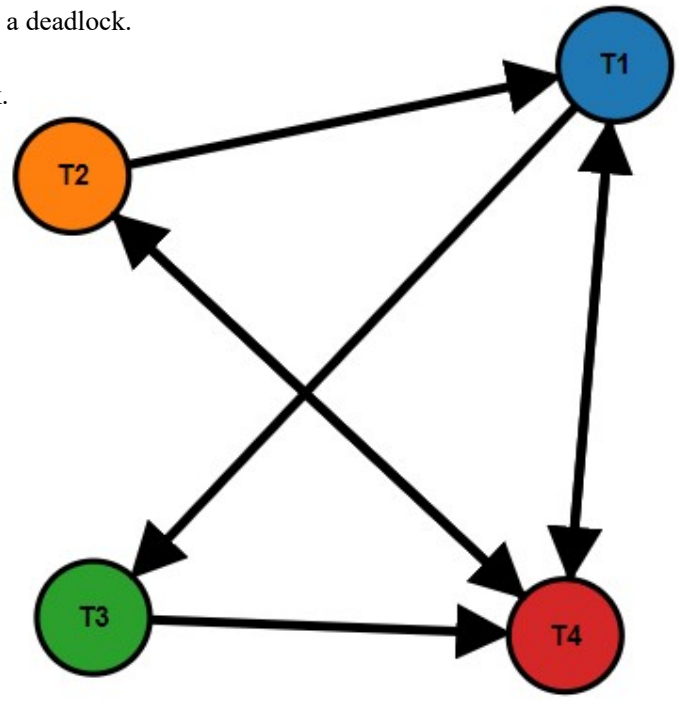
Pg 43 week 7

Time	!1	!2	!3	!4	!5	!6	!7	!8	!9	!10	!11	!12
T_1	R(B) Lock S(B)				R(A) Lock S(A)	W(B) Unlock S(B)		W(A) Unlock S(A)				
T_2		R(A) Lock S(A)	W(A) Unlock S(A)									
T_3							R(B) Lock S(B)		W(B) Unlock S(B)			
T_4	R(A) Lock S(A)									R(B) Lock S(B)	W(A) Unlock S(A)	W(B) Unlock S(B)

r4(a) r1(b) r2(a) w2(a) r1(a) w1(b) r3(b) w1(a) w3(b) r4(b) w4(a) w4(b)

Two Phase Locking –
A transaction is said to follow Two Phase Locking protocol if Locking and Unlocking can be done in two phases.
Growing Phase: New locks on data items may be acquired but none can be released.
Shrinking Phase: Existing locks may be released but no new locks can be acquired.
Shared Lock (S): also known as Read-only lock. As the name suggests it can be shared between transactions because while holding this lock the transaction does not have the permission to update data on the data item. S-lock is requested using lock-S instruction.
Exclusive Lock (X): Data item can be both read as well as written. This is Exclusive and cannot be held simultaneously on the same data item. X-lock is requested using lock-X instruction. Since this graph is cyclic, it can cause a deadlock.
For example, T4 at timeslot 1 locks A for read,
Then T2 also wants to write A causing a deadlock.

For T1 – Lock B Lock A Unlock B Unlock A
For T2 – Lock A Unlock A
For T3 – Lock B Unlock B
For T1 – Lock A Lock B Unlock A Unlock B



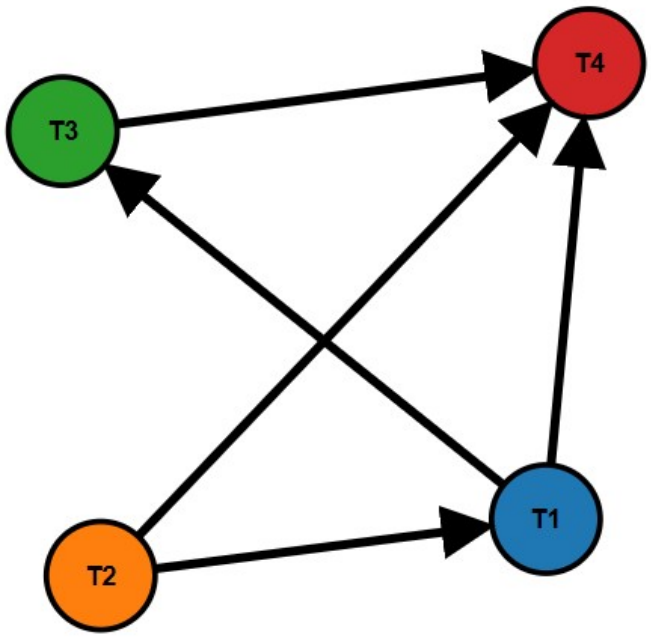
Since this graph is cyclic, it can cause a deadlock.
.

4) **Construct a schedule (which is different from above) of these four transactions which does not cause deadlock when using two-phase locking protocol. If no such schedule exists, explain why. (2 marks)**

Time	!1	!2	!3	!4	!5	!6	!7	!8	!9	!10	!11	!12
T_1	R(B) Lock S(B)				R(A) Lock S(A)	W(B) Unlock S(B)		W(A) Unlock S(A)				
T_2		R(A) Lock S(A)	W(A) Unlock S(A)									
T_3							R(B) Lock S(B)		W(B) Unlock S(B)			
T_4									R(A) Lock S(A)	R(B) Lock S(B)	W(A) Unlock S(A)	W(B) Unlock S(B)

r1(b) r2(a) w2(a) r1(a) w1(b) r3(b) w1(a) w3(b) r4(a) r4(b) w4(a) w4(b)

For T1 – Lock B Lock A Unlock B Unlock A
For T2 – Lock A Unlock A
For T3 – Lock B Unlock B
For T1 – Lock A Lock B Unlock A Unlock B
Therefore 2 Phase Locking.



The graph is acyclic hence conflict serialisable and won't cause any deadlocks