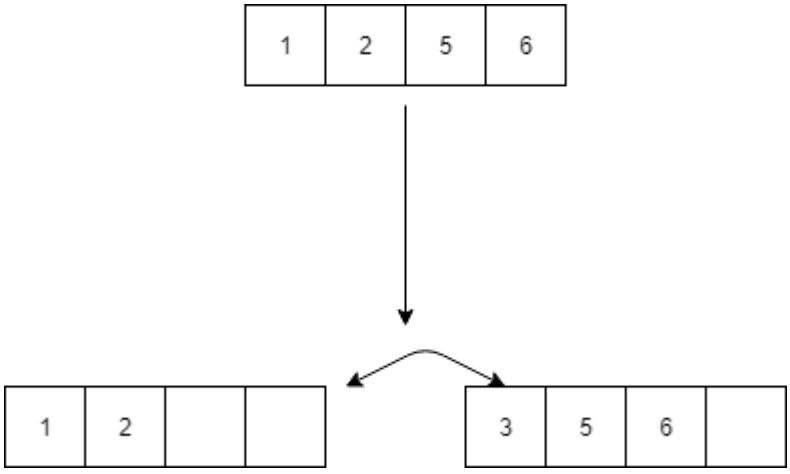
Q1. Consider the B+ tree shown in the following as an original tree.

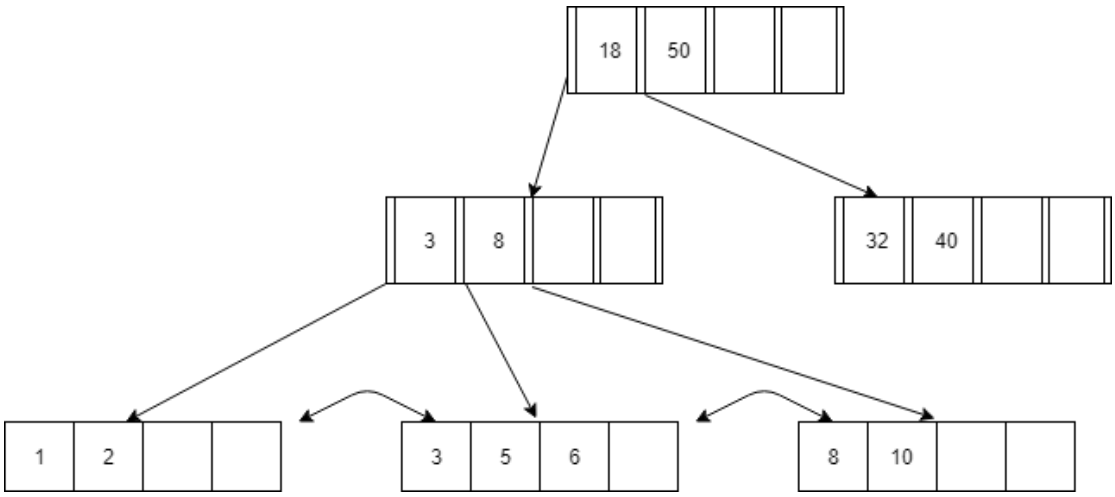
1. We can add 14 records into the position are empty, if we add more, then its parent pop up, for example, add to the right leaf node of the right node, then it split. As a result, we can add 8 more, then the right parent node is full(level two). Add 4 more again, the parent node split(becomes two and two), and the root has two records. As a result, we can add another 16 more records to make right level two node become full For the right node, add 4 more, it split again, and the root has three records, add 16 more, then add 4 more, root become full, and we can still add 16 more to make the level two node full.

The maximum total additional records could be added to this tree without changing its height is 14+8+4+16+4+16+4+16=82

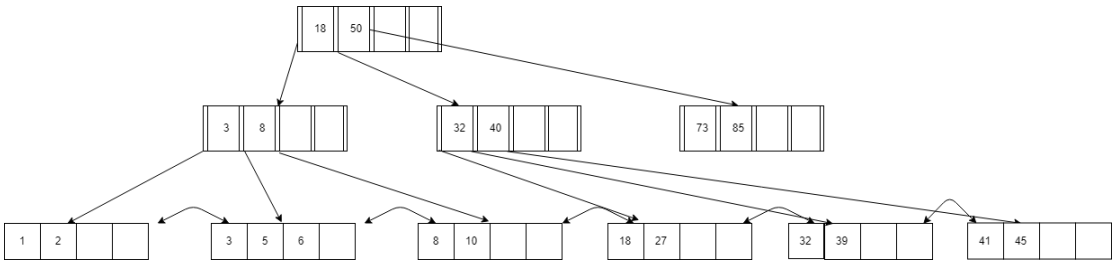
2.When insert 3, its nodes become like below



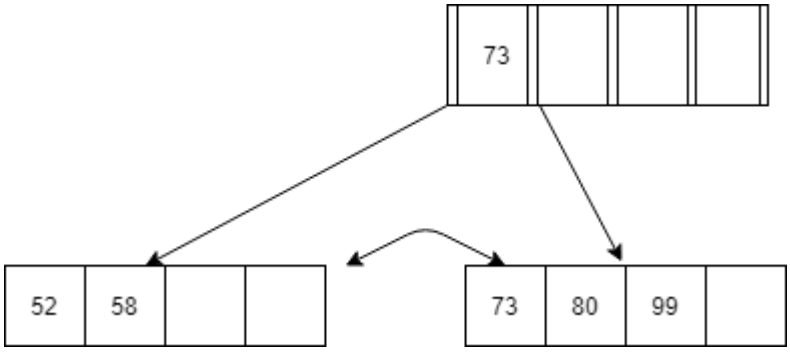
Finally, it become like this, only draw the changes (for example, the right subtree remains unchanged, so I ignore it and do not draw the same thing).



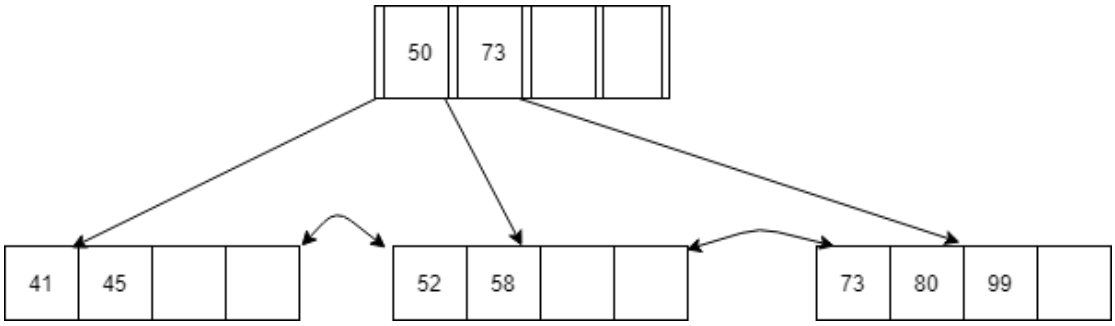
Finally, it will become like this(the right part is omitted, I only draw the left part, because the right part is the same as before)



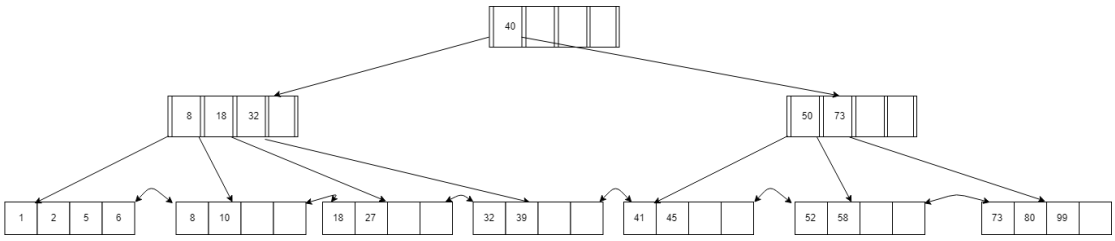
3. After delete 91, the far right node will merge with the left node that besides it, and the parent borrow one node from left sibling. Now it satisify all the need. I will just show the result and the process. Other part will remain the same as before.



After borrow the left sibiing. The root become 40.



Finally, it become like below.



Q2.

In this case, the cheapest approach is clustered B+ tree on (R.a, R.b), this gives the cheapest cost because it is an index-only plan, and therefore will not have to access the data records in the file that contain the queried relations. If we want to find a record equal to something, linear hashed index on attribute will be better. If want to find something not equal to something, accessing the sorted file will be better, since the selection will require a scan of the available entries, and starting at the beginning of the sorted index.

In conclusion, for this case, select a>2,000,000 and a<8,000,000, A clustered B+ tree index on attributes (R.a, R.b) is the cheapest.

Q3.

1.For T1, T3, T4, they do not finish their work before t4 and t5 (the time that check point is made), so these transactions need to be recovered. T2 has been done before that time, so this transaction should not be recovered (REDO or UNDO). Therefore, the operations for the four transactions shows below.

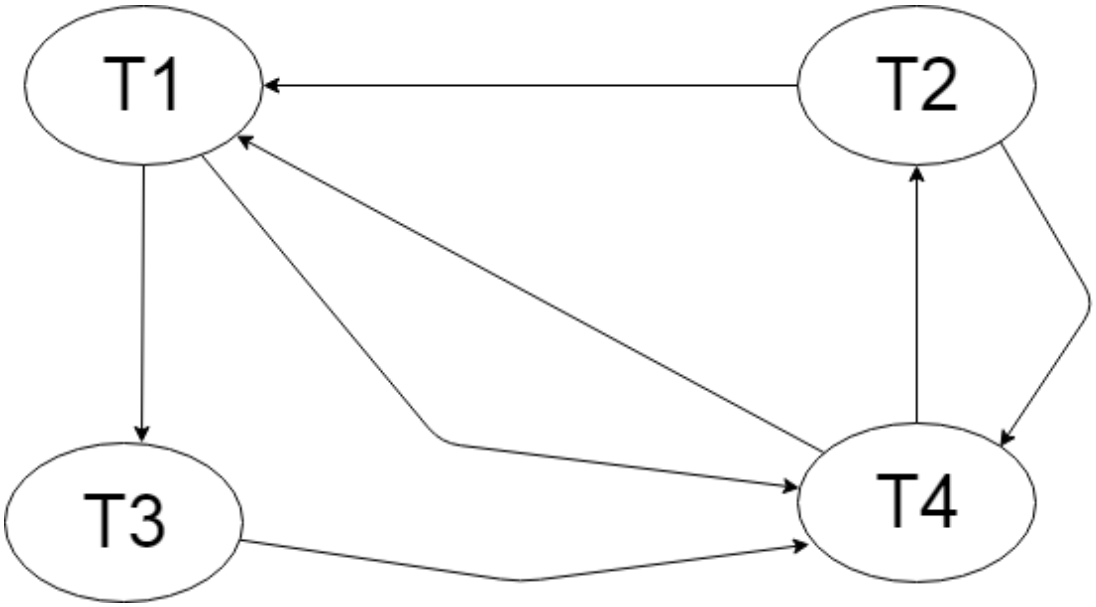
T1: UNDO

T2: No operation

T3: UNDO

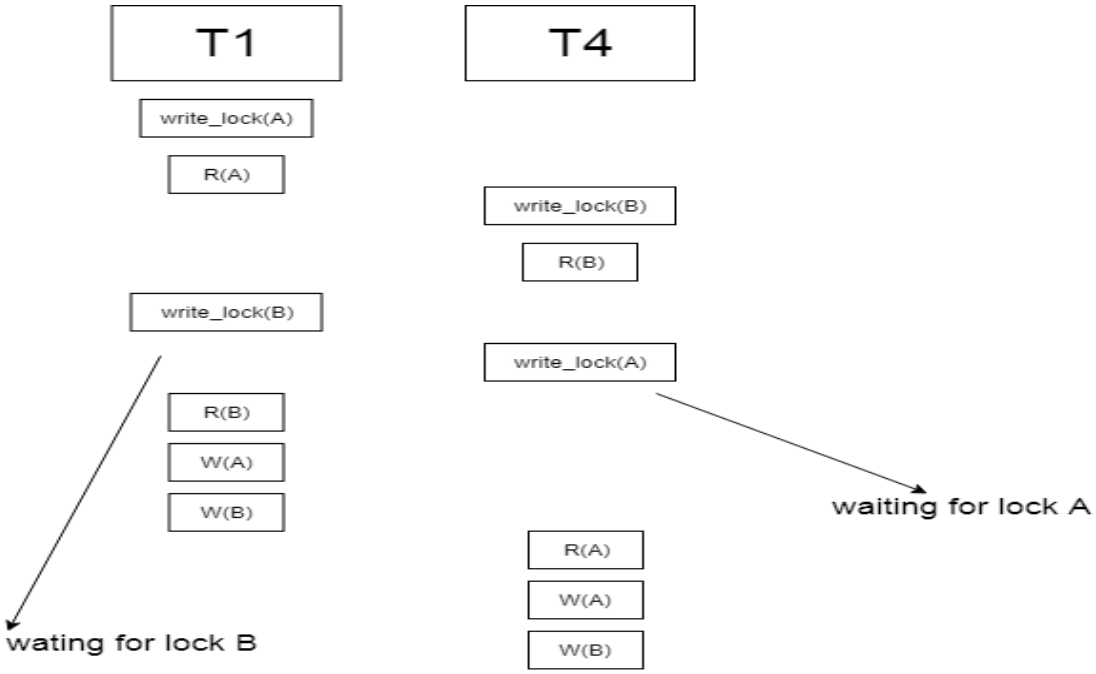
T4: UNDO

2. The precedence graph is as follows, so the transaction schedule is not conflict serializable, because there are circles in this graph.

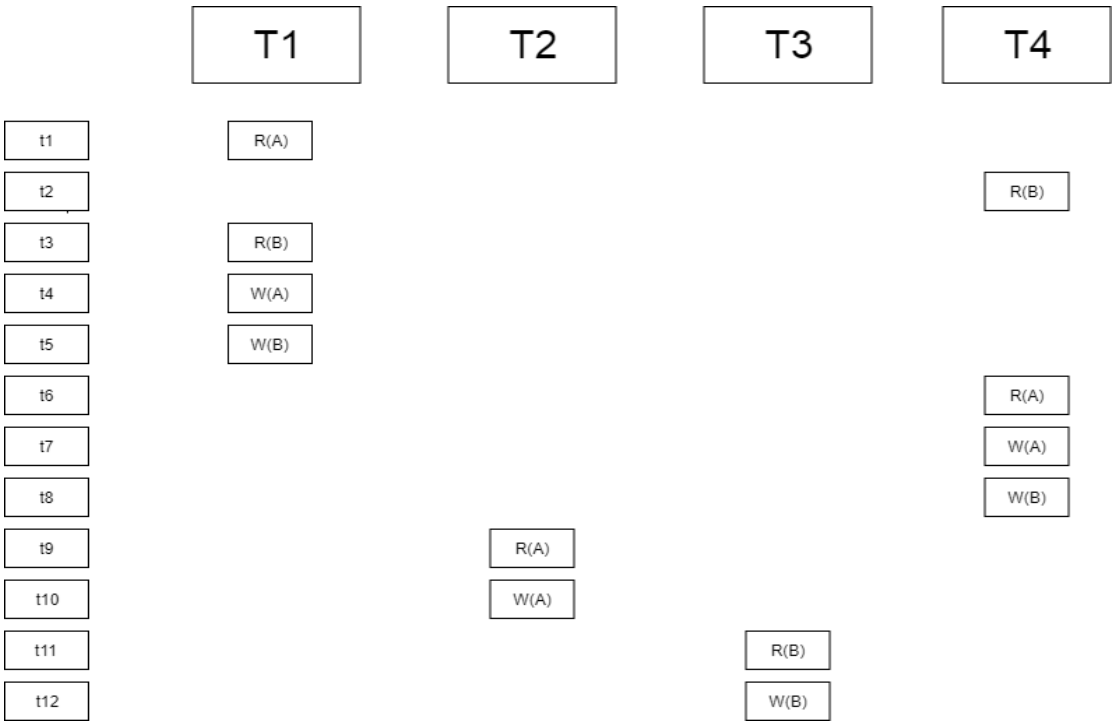


3.T1, T4 read and write on A and B, so we have potential to make a

deadlock. Just for T1 and T4, there is a deadlock. T1 obtain a write lock on A and then T4 obtain a lock on B, before T1 read B, it need to wait for T4 leasing the lock on B, but T4 is waiting for a lock on A that T1 release. So there is a deadlock.



Then show the schedual of these four transctions.



4. a serial schedule will not cause deadlock, like follows, every lock will be released when one transaction is done, so it will not affect other transaction. Although the efficiency of this schedule is low, it will not cause deadlock. Actually we even no do need the lock, and it is still conflict serializable.

I will just give the schedual like below.