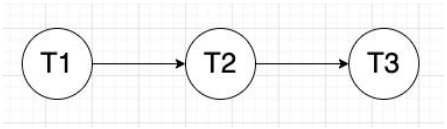
Homework 1

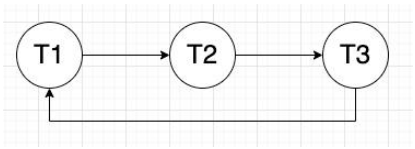
NetID: xm2074, jh7948
Name: Xiao Ma, Junru He

1. Determine if it is serializable of each executions.

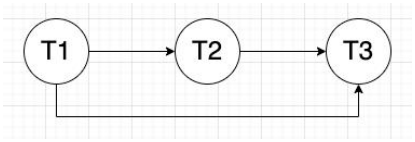
a) . Serializable. The graph is asyclic.



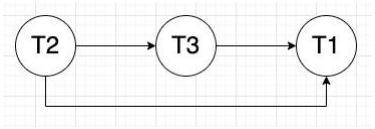
b) . Not serializable. The graph has a cycle.



c) . Serializable. The graph is asyclic.



d) . Serializable. The graph is asyclic.

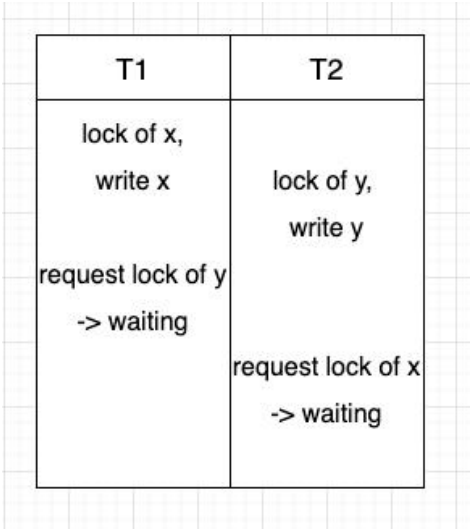


2. Proof of the serialization graph theorem.

Suppose the actual execution and the serial order have different reads-from. Suppose Rm(x) of Tm reads x from Wp(x) of Tp in the serial order, but Rm(x) of Tm reads x from Wq(x) of Tq in actual execution. Since both Tp and Tq contain writes to x, they must be connected in the serialization graph, so these three transactions must be ordered in the graph. Two cases:

1) Tp → Tq. Then Tq must fall between Tp and Tm in serial schedule. So Tm reads-from Tq in serial schedule.
So Wq(x) is the final write. This contradicts with the prerequisites that Wp(x) is the final write in the serial order.

2) Tq → Tp. Then Tm must read x from Tp in actual execution.
So Wp(x) is the final write, which is contradicts the prerequisites that Wq(x) is the final write in the actual order.



T1	T2
lock of x, write x request lock of y -> waiting	lock of y, write y request lock of x -> waiting

3. Is two phase locking deadlockfree?

No. It may produce a deadlock.
Considering the following scenario:
T1 has the lock of x and writes x, and T2 has the lock of y and writes y. Both of transaction T1 and T2 do not release the lock until the transaction ends. But at this time, T1 wants to read y and T2 wants to read x, so both of them request the lock of each other, and both of them owns the lock that each other needs. Then a deadlock produces.

4. “Pure” two pharse locking -> order-preserving serializable executions.

To prove that “pure” two phrase locking produces order-preseving serializable excutions, we need two steps.

- First we need to prove that “pure” two phrase locking produces serializable excutions. That means there is no cycle in the serialization graph.

Suppose there is a cycle, take T1 -> T2 -> T1 as an example. For the definition, we know that locks can be released earlier than the end of the transaction but **no new locks are acquired** aftewards. That means the transaction finish all possible conflict operations before releasing the lock. We show the time point that the transaction finish its conflict operations and release lock with lockpoint(transaction). For the example, lockpoint(T1) < lockpoint(T2), that means T1 finishes all conflict operations and releases locks before T2 starts. Then from the graph, lockpoint(T2) < lockpoint(T1), it contradicts with the statement before, so a cycle is impossible.

- Second we need to prove that “pure” two phrase locking produces order-preseving serializable excutions.

Take T1 -> T2 -> T1 as an example again. Suppose that T1 and T2 are not serializable. There must be some conflicts between T2 and T1 (conflict of T1 that happens after T2 begins). Accoring to the definition of “pure” two phrase locking, T2 begins when all conflicts in T1 has been finished. Then it is impossible that there is conflict of T1 that happens after T2 begins. So all operations of T1 must precede all operations of T2, and T1 and T2 is an equivalent serial execution.

From all above, “pure” two phrase locking produces order-preseving serializable excutions.

5. Would intention locking protocal be serializable?

No. The intention locking protocol is not serialiazable.
Considering the following scenario as the counterexample:
T1 requests intention write lock on Database and intention write lock on File 1. If intention lock can be released at any time, then T1 releases intention write lock on Database when T1 gets intention write lock on File 1.
At the same time, T2 requests read lock on Database and it succeeds.
Under this circumstance, T1 and T2 are not serializable.

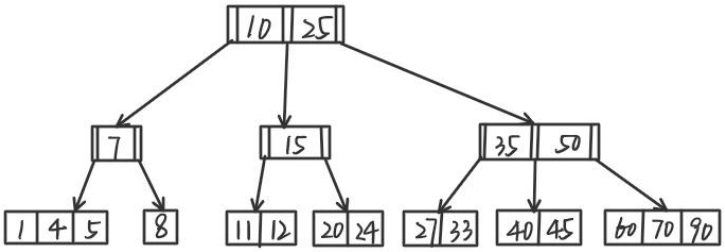
6. Modify optimistic concurrency control algorithm.

We can record a priority accoding to the timestamp of every transaction. Then time stamp is the first time that the transaction begins and will never change even if it restarts. The earlier the transaction starts, the priority is higher.
When there is a conflict, keep the transaction with higher priority and let it commit, then restart the other one. With this method, we can make sure that every transaction can eventually completes.

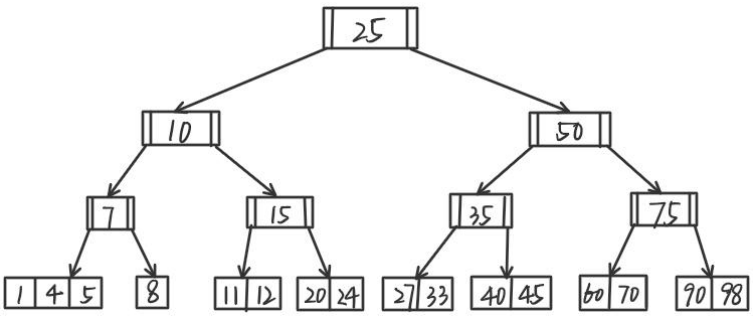
7. B-Tree

a)

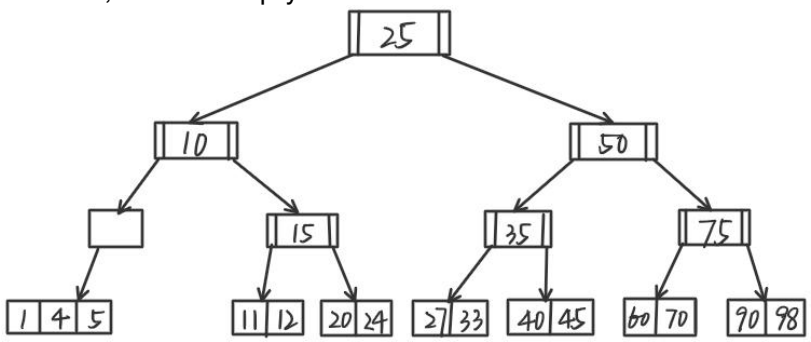
Original data structure:



Insert 98, Split:



Delete 8, free-on-empty:



b)

```
begin at some non-leaf node n
while x is not in n:
        if x in keyset(n):
                return None   # x is not found
        else if x is in inset(n) and edgeset(n, n'):
                n = n'
        else:
                set n to some ancestor node
if n is leaf:
        return key x and its value
else:
        return None
```

c)

For any search operation on any data item x beginning at any node m, the following invariant holds:
After any operation of any process, if the search/insert/delete for item x is at node n1, then either
item x is in keyset(n1) or there is a path from n1 to node n2 such that item x is in keyset(n2) and
every edge E along that path has x in its edgeset.
The invariant above works for ordinary B tree that data is stored not only in leaf nodes but also in
inner nodes. However, in our case, data is only stored in leave nodes and inner nodes only store indexes.
Therefore, in question (b), we make sure that the search algorithm gets valid result only on leaf nodes. So, the invariant also holds in the B-tree in our problem.

d)

When the node becomes a leaf node or all its children become empty.

8. Multi-version read consistency ensures serializability.

Considering the following scenario:
Suppose that the multi-version read consistency can not ensure the serializability, that means there is a cycle in the serializability graph. Then we suppose that Ti -> Tj
(lockpoint(Ti) < lockpoint(Tj)) has a conflict, it has three cases:

- Both Ti and Tj are read-only transaction:
  It is serializable, because read can not conflicts.

- Ti is read-only and Tj is read-write:
  In this case, Ti may read an item x before Tj write on x, and Tj writes on y before Ti reads item y. That means until Tj release it lock, Ti can reads item y, so lockpoint(Tj) < lockpoint(Ti), which contradicts with the assumption.

- Ti is read-write and Tj is read-only:
  In this case, Ti holds a lock until Ti ends. After Ti releases the lock, Tj reads the item. Therefore, lockpoint(Ti) precedes lockpoint(Tj). It is serializable.

Above all, multi-version read consistency ensures serializability.

9. Oracle database system.

This can lead to a non-serializable execution.

Considering the following senerio:
T1: select * from db where id < 20
T2: insert into db values (10, 'Tom')
T1: select * from db where id < 25

For multi-version read consistency, T1 do not require locks, The first part of T1 executes before T2, and the second part executes after T2 releasing locks. So the first part and the second part of T1 get different result from the same transaction. And the serializability graph is T1 -> T2 -> T1, having a cycle.

So it leads to a non-serializable execution.

10. AQuery

i) The code and the result are as following:

Code:

```
DROP TABLE IF EXISTS ticks

CREATE TABLE ticks(id INT, date INT, endofdayprice INT)

LOAD DATA INFILE "data/ticks.csv" INTO TABLE TICKS FIELDS
TERMINATED BY ","

SELECT id, max(ratios(endofdayprice)) as max_return,
min(ratios(endofdayprice)) as min_return FROM ticks
ASSUMING ASC id, ASC date
GROUP BY id
```

Result:

```
id | max_return | min_return
============================
3000 124 0.00909091
3001 117 0.00571429
3002 190 0.00869565
3003 83.5 0.037037
done.
```

ii)
Since id has been sorted, so group by id is an easy operation that cost little. Beside, another thing we need to do is to sort date for each id and calculate the ratio. Then we write the following code:

```
SELECT id, max(ratios(endofdayprice)) as max_return,
min(ratios(endofdayprice)) as min_return FROM ticks
ASSUMING ASC date
GROUP BY id
```