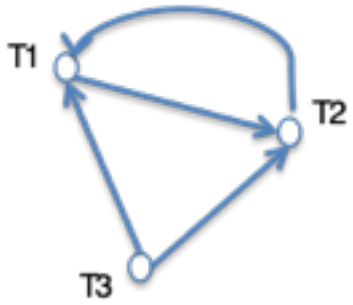
Problem 1 (Precedence Graph) [30 Points]

For each of the following schedules, answer the following:

Q1: Draw the precedence graph for the schedule.
Q2: Is the schedule conflict-serializable? If yes, write down a possible equivalent serial schedule. If not, explain why not.

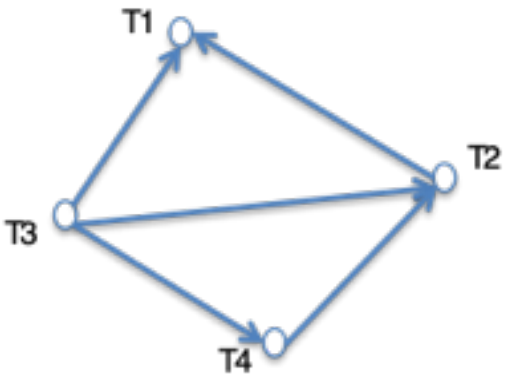
S1: r1(A), r1(B), r2(A), w3(C), w2(B), w2(C), w1(C)

Not conflict-serializable



S2: w3(A), w2(C), r1(A), w1(B), r1(C), r2(A), r4(A), w4(D), r2(D)

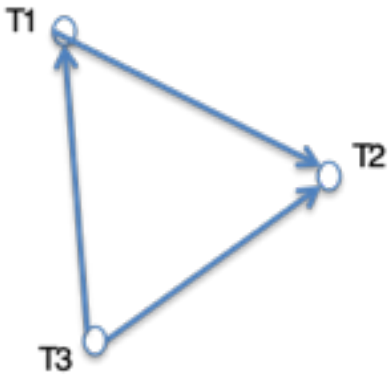
It is Conflict-Serializable.
One possible schedule: T3, T4, T2, T1



S3:

Transaction T_1	Transaction T_2	Transaction T_3
		read_item(Y);
		read_item(Z);
read_item(X);		
		write_item(Y);
write_item(X);		
		write_item(Z);
	read_item(Z);	
read_item(Y);		
write_item(Y);		
	read_item(Y);	
	write_item(Y);	
	read_item(X);	
	write_item(X);	

Conflict Serializable



Problem 2 (Locking Protocol) [30 Points]

Q1 [10 Points]: For each of the following schedules, state whether or not the schedule is legal, and discuss why.

Remember: A schedule is legal iff:
- *No two transactions have a lock on the same object, and*
- *No transaction unlocks an object that it does not own a lock for*

S1 = l1(A) l1(B) r1(A) w1(B) l2(B) u1(A) u1(B) r2(B) w2(B) u2(B) l3(B) r3(B) u3(B)

>>> Not legal because T2 gets a lock on B (action 5) while T1 still holds its lock (action 2)

S2 = l1(A) r1(A) w1(B) u1(A) u1(B) l2(B) r2(B) w2(B) l3(B) r3(B) u3(B)

>>> Not legal because T1 unlocks B (action 5) while it does not have a lock on this object

Q2 [20 Points]: State whether or not Transaction 1 in each of the above schedules is ***Well-Formed***. Discuss why?

By checking the actions of T1 only, we get:
S1 = l1(A) l1(B) r1(A) w1(B) u1(A) u1(B) ➔ T1 is well formed

S2 = l1(A) r1(A) w1(B) u1(A) u1(B) ➔ T1 is not well formed because it unlocks B without locking it

Problem 3 (2 Phase Locking Protocol) [20 Points]

1. Does the following sequence of actions follow 2PL? (L denotes Lock, U denotes Unlock, R denotes Read, W denotes Write, O denotes Output. So L1(A) denotes that Transaction 1 gets lock for A. We do not show Input here, but instead assume that that will happen along with R or W).

```
0. Start T1.
1. L1(A)
2. R1(A)
3. W1(A)
4. O1(A)
5. Start T2.
6. L2(C)
7. L1(B)
8. U1(A)
9. L2(A)
10. W1(B)
11. Commit T1
12. R2(A)
13. W2(C)
14. U1(B)
15. Commit T2
16. U2(A)
17. U2(C)
```

>> It follows the 2PL rules since no transaction creates new locks after the first unlock

Problem 4 (Recovery Control) [20 Points]

The following is a sequence of undo-log records written by two transactions *T* and *U*:

<Start *T*>; <*T*, A, 10>; <Start *U*>; <*U*, B, 20>; <*T*, C, 30>; <*U*, D, 40>; <Commit *U*>; <*T*, E, 50>; <Commit *T*>

Describe the action of the recovery manager, including the changes to both disk and the log if there is a crash and the last log record to appear on disk is:

(a) < Start U>
(b) <Commit U>
(c) <T, E, 50>
(d) <Commit T>

(a) Both U and T will be undone. U has no actions to undo, while T will have one action to undo (set A = 10). Then two records will be added to the log <Abort T> <Abort U>

(b) Since <Commit U> is on disk, then all its updates are on disk. So, U will be skipped. Transaction T will be undone in the following order (Backward):
- C =30
- A =10

Then, an <Abort T> record will be added to the log

(c) Since <Commit U> is on disk, then all its updates are on disk. So, U will be skipped. Transaction T will be undone in the following order (Backward):
- E = 50
- C =30
- A =10

Then, an <Abort T> record will be added to the log

(d) Both transactions will be skipped because both of them are committed.