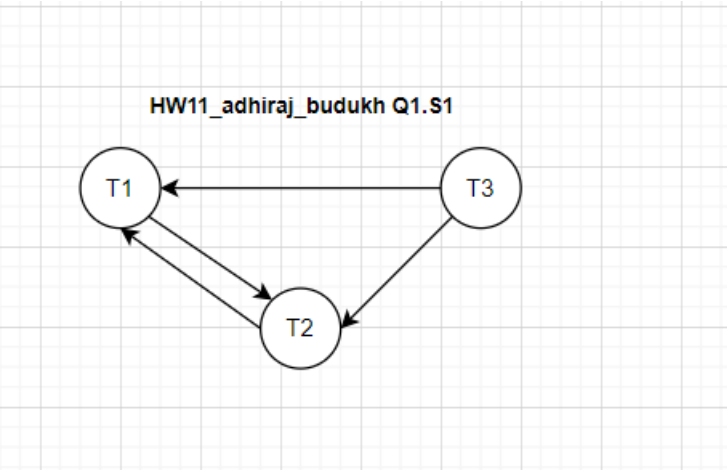
Homework11

1.S1

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

T1	T2	T3
R(A)		
R(B)		
	R(A)	
		W(C)
	W(B)	
	W(C)	
W(C)		



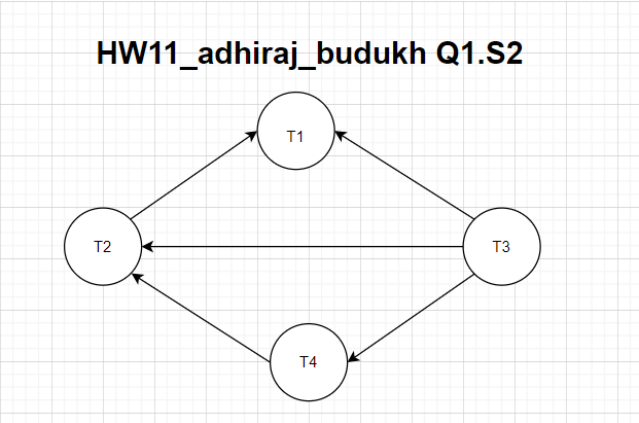
The graph has a cycle. It is **NOT CONFLICT SEIALIZABLE.**

The cycle indicates conflicting operations between transactions that cannot be serialized while maintaining the same order of operations. This can lead to different final states depending on the order of execution, violating the serializability property.

1.S2

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

T1	T2	T3	T4
		W(A)	
	W(C)		
R(A)			
W(B)			
R(C)			
	R(A)		
			R(A)
			W(D)
	R(D)		

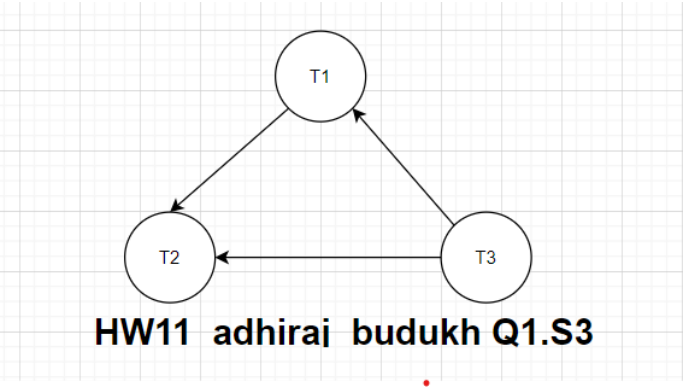


The graph has no cycle. It is **CONFLICT SERIALIZABLE.**

T3 -> T4 -> T2 -> T1

1.S3

The graph is **CONFLICT SERIALIZABLE.**



2.1

The given schedule,

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)

Is NOT LEGAL.

Reason: Transaction T2 tries to Lock item B while T1 has still lock on it.

Prior to T1 unlocking the item, T2 can't lock the same.

The given schedule,

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

Is **NOT LEGAL.**

Reason: Transaction T3 tries to put Lock on item B while T2 still has lock on it.

2.2

Transaction T1 in S1 is well formed, since there are no violations of locking or unlocking.

Transaction T1 in S2 is not well formed, because T1's tried unlocking item B before locking; the unlocking of an item should be preceded by a corresponding lock operation.

2.3

Given,

START(T1)	L1(A)	R1(A)	W1(A)	O1(A)
START(T2)	L2(C)	L1(B)	U1(A)	L2(A) W1(B)
COMMIT T1	R2(A)	W2(C)	U1(B)	
COMMIT T2	U2(A)	U2(C)		

The sequence of actions follows 2 Phase Locking protocol 2PL. Transactions T1 and T2 acquire and release locks in consistent manner.

2.4

a) <Start U>

We undo the changes made by U, since U or T have not been committed. Hence, it is necessary to reverse both U and T; later write the changes to disk accordingly.

When we'll first check the log from file's end, we discover that U hasn't changed hence we'll log <ABORT U>. Next A comes, we'll undo A since it's not done.

We'll write A to disk with value 10 set.

b) <Commit U>

Since U is committed and written to disk there is no need to undo the changes. We have to check for any active transactions from the end of the log (T in this case). If there are active transactions, undo their changes and log <ABORT T>.

Prior to writing changes to disk, we first set C to 30, A to 10 then log < ABORT T >.

c) <T,E,50>

U is committed and written to disk. But T has not committed. Hence, we have to undo the changes made by T since no commit transaction has been seen.

We set E to 50, C to 30, A to 10 starting at the end of the log file and write changes to disk.

Log<ABORT T>.

d) <Commit T>

If the log ends with this entry means both U and T are committed, written to disk. No need to undo any changes since there are no active transactions, no further changes are required.