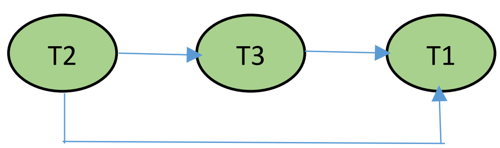
## HW6

1. For each of the following schedules, state whether it is conflict-serializable. If yes, provide all equivalent serial schedules. If no, state why it is not conflict-serializable. (ri(X) denotes a read on object X for transaction Ti. wj(Y) denotes a write on object Y for transaction Tj.)

   1) r1(X), r3(Y), r2(Y), w3(Y), r3(X), r2(Z), w1(X), w2(Z), r1(Z), w1(Z)

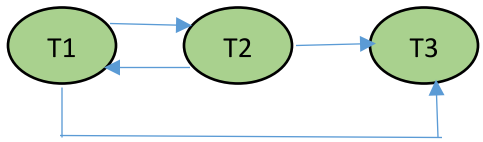
   **Solution**: We start with an empty graph with three vertices labeled $T_1$, $T_2$, $T_3$. We go through each operation in the schedule:
   r1(X): no subsequent writes to X from other transactions, so no new edges;
   r3(Y): no subsequent writes to Y from other transactions, so no new edges;
   r2(Y): Y is subsequently written by T3, so add edge T2 → T3;
   w3(Y): no subsequent operations to Y, so no new edges;
   r3(X): X is subsequently written by T1, so add edge T3 → T1;
   r2(Z): Z is subsequently written by T1, so add edge T2 → T1;
   w1(X): no subsequent operations to X, so no new edges;
   w2(Z): Z is subsequently read by T1, so add edge T2 → T1;
   r1(Z): no subsequent writes to Z from other transactions, so no new edges;
   w1(Z): no subsequent operations, so no new edges;
   We end up with the following precedency graph:




   This graph has no cycles, so the original schedule must be conflict-serializable. The equivalent serial schedule is T2-T3-T1.

   2) r1(X), r2(Y), r3(Y), w3(Y), r2(Z), w1(X), r3(X), r1(Z), w2(Z), w1(Z)

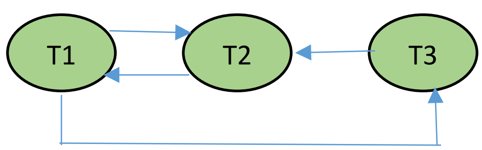
   **Solution**: We start with an empty graph with three vertices labeled $T_1$, $T_2$, $T_3$. We go through each operation in the schedule:
   r1(X): no subsequent writes to X from other transactions, so no new edges;
   r2(Y): Y is subsequently written by T3, so add edge T2 → T3;
   r3(Y): no subsequent writes to Y from other transactions, so no new edges;
   w3(Y): no subsequent operations to Y, so no new edges;
   r2(Z): Z is subsequently written by T1, so add edge T2 → T1;
   w1(X): X is subsequently read by T3, so add edge T1 → T3;
   r3(X): no subsequent operations to X, so no new edges;
   r1(Z): Z is subsequently written by T2, so add edge T1 → T2
   w2(Z): Z is subsequently written by T1, so add edge T2 → T1;
   w1(Z): no subsequent operation, so no new edges;
   We end up with the following precedency graph:




This graph has a cycle, so the original schedule is not conflict-serializable.

3) r1(X), w1(X), r3(Y), r1(Z), w3(Y), r2(Y), r2(Z), r3(X), w1(Z), w2(Z)

**Solution**: We start with an empty graph with three vertices labeled T1, T2, T3. We go through each operation in the schedule:
r1(X): no subsequent writes to X from other transactions, so no new edges;
w1(X): X is subsequently read by T3, so add edge T1 → T3;
r3(Y): no subsequent writes to Y from other transactions, so no new edges;
r1(Z): Z is subsequently written by T2, so add edge T1 → T2;
w3(Y): Y is subsequently read by T2, so add edge T3 → T2;
r2(Y): no subsequent operations to Y, so no new edges;
r2(Z): Z is subsequently written by T1, so add edge T2 → T1;
r3(X): no subsequent operations to X, so no new edges;
w1(Z): Z is subsequently written by T2, so add edge T1 → T2;
w2(Z): no subsequent operations, so no new edges;
We end up with the following precedency graph:




This graph has cycles, so the original schedule is not conflict-serializable.

2. Let T1, T2, T3 be the following transactions:

   T1: r1(D), w1(B), w1(D)

   T2: r2(C), w2(B)

   T3: r3(D), w3(D), w3(B)

   For each of the following schedules, state whether it is possible under 2PL protocol? Give your reasons.

   1) r1(D), r2(C), w2(B), w1(B), r3(D), w3(D), w1(D), w3(B)

   **Solution**: Let's try to add the lock/unlock steps to the schedule,
   l1(D), r1(D), l2(C), r2(C), l2(B), u2(C), w2(B), u2(B), l1(B), w1(B), …

The next operation is r3(D), before that operation T3 needs to get a lock on D, however, D is currently locked by T1. T3 cannot lock D until T1 has unlocked D. If T1 unlock D at this moment, according to 2PL, then T1 couldn't get lock on any other resources after that, then the subsequent operation w1(D) of T1 cannot proceeds. Thus, this schedule is not possible under 2PL protocol.

2) r3(D), r1(D), w1(B), w3(D), r2(C), w2(B), w1(D), w3(B)

**Solution**: Let's try to add the lock/unlock steps to the schedule,
l3(D), r3(D), …
The next operation is r1(D), before that operation T1 needs to get a lock on D, however, D is currently locked by T3. According to 2PL protocol, T3 won't release the lock on D until it locks B. Suppose T3 has already locked B at this moment. However, T3 won't release the lock on D until it has finished all the operations on D (after w3(D)). In other words, in this case it is impossible for r1(D) to occur before w3(D). Thus, this schedule is not possible under 2PL protocol.

3) r2(C), r1(D), w2(B), r3(D),w3(D), w3(B), w1(B), w1(D)

**Solution**: Let's try to add the lock/unlock steps to the schedule,
l2(C), r2(C), l1(D), r1(D), l2(B), u2(C), w2(B), u2(B), …
The next operation is r3(D), T3 requires to lock D at this moment before proceeds, however, T1 won't release the lock on D before it locks B. Suppose T1 has already acquired the lock on B at this moment. However, T1 won't release the lock on D until it has finished all the operations on D (after w1(D)). In other words, in this case it is impossible for r3(D) to occur before w1(D). Thus, this schedule is not possible under 2PL protocol.

4) r1(D), r2(C), w2(B), r3(D), w1(B), w3(D),w3(B), w1(D)

**Solution**: Let's try to add the lock/unlock steps to the schedule,
l1(D), r1(D), l2(C), r2(C), l2(B), u2(C), w2(B), u2(B), …
The next operation is r3(D). T3 needs to lock D before this read, however, D is currently locked by T1. According to 2PL, T1 won't unlock D until it has acquired all the locks on other resources. Suppose T1 has already acquired the lock on B at this moment. However, T1 won't release the lock on D until it has finished all the operations on D (after w1(D)). In other words, in this case it is impossible for r3(D) to occur before w1(D). Thus, this schedule is not possible under 2PL protocol.

Here are some example schedules which are consistent with 2PL protocol.
1) r3(D), r2(C), w3(D), w2(B), r1(D), w3(B), w1(B), w1(D)
Let's try to add the lock/unlock steps to the schedule:
l3(D), r3(D), l2(C), r2(C), w3(D), l2(B), u2(C), w2(B), u2(B), l3(B), u3(D), l1(D), r1(D), w3(B), u3(B), l1(B), w1(B), u1(B), w1(D), u1(D).

Pull out the lock/unlock operations of all the three transactions:
T1: l1(D), l1(B), u1(B), u1(D)
T2: l2(C), l2(B), u2(C), u2(B)
T3: l3(D), l3(B), u3(D), u3(B)
Which are consistent with the 2PL protocols.

2) r2(C), r1(D), w1(B), w2(B), w1(D), r3(D), w3(D), w3(B)
Let's try to add the lock/unlock steps to the schedule:
l2(C), r2(C), l1(D), r1(D), l1(B), w1(B), u1(B), l2(B), u2(C), w2(B), u2(B), w1(D), u1(D), l3(D), r3(D), w3(D), l3(B), u3(D), w3(B), u3(B).
Pull out the lock/unlock operations of all the three transactions:
T1: l1(D), l1(B), u1(B), u1(D)
T2: l2(C), l2(B), u2(C), u2(B)
T3: l3(D), l3(B), u3(D), u3(B)
Which are consistent with the 2PL protocols.

3. Let us begin with two bars: Cabana and Old Tavern. Cabana has local patrons A and remote patrons B, while Old Tavern has local patrons C and remote patrons D. Now a new bar New Tavern is opened in this area. Then Cabana and Old Tavern begin to lose clients. Suppose we have the followings two transactions:

T1: Cabana loses all patrons to Old Tavern. First locals, than the rest
T2: Old Tavern loses all patrons to New Tavern, first locals than the rest

Given the following schedule S:

T2: Insert local patrons of Old Tavern into New Tavern
T1: Insert local patrons of Cabana into Old Tavern
T2: Delete local patrons of Old Tavern
T1: Delete local patrons of Cabana
T2: Insert remote patrons of Old Tavern into New Tavern
T2: Delete remote patrons of Old Tavern
T1: Insert remote patrons of Cabana into Old Tavern
T1: Delete remote patrons of Cabana

What patrons will each bar have after the execution of this schedule (in terms of A, B, C, D or empty)? Is the schedule serializable (result equivalent to a serial schedule)?

**<u>Solution</u>**: Let's denote the local patrons and remote patrons of Cabana as C_lp, C_rp, respectively. Similarly, the local patrons and remote patrons of Old Tavern are OT_lp, OT_rp. For New Tavern, we have NT_lp, NT_rp for its local patrons and remote patrons. The initial values are:

| C_lp | C_rp | OT_lp | OT_rp | NT_lp | NT_rp |
|---|---|---|---|---|---|
| A | B | C | D | empty | empty |

We will record the values for these variables after each operation in the schedule:

After T2 inserts local patrons of Old Tavern into New Tavern:

| C_lp | C_rp | OT_lp | OT_rp | NT_lp | NT_rp |
|---|---|---|---|---|---|
| A | B | C | D | C | empty |

After T1 inserts local patrons of Cabana into Old Tavern:

| C_lp | C_rp | OT_lp | OT_rp | NT_lp | NT_rp |
|---|---|---|---|---|---|
| A | B | A, C | D | C | empty |

After T2 deletes local patrons of Old Tavern:

| C_lp | C_rp | OT_lp | OT_rp | NT_lp | NT_rp |
|---|---|---|---|---|---|
| A | B | empty | D | C | empty |

After T1 deletes local patrons of Cabana:

| C_lp | C_rp | OT_lp | OT_rp | NT_lp | NT_rp |
|---|---|---|---|---|---|
| empty | B | empty | D | C | empty |

After T2 inserts remote patrons of Old Tavern into New Tavern:

| C_lp | C_rp | OT_lp | OT_rp | NT_lp | NT_rp |
|---|---|---|---|---|---|
| empty | B | empty | D | C | D |

After T2 deletes remote patrons of Old Tavern:

| C_lp | C_rp | OT_lp | OT_rp | NT_lp | NT_rp |
|---|---|---|---|---|---|
| empty | B | empty | empty | C | D |

After T1 inserts remote patrons of Cabana into Old Tavern:

| C_lp | C_rp | OT_lp | OT_rp | NT_lp | NT_rp |
|---|---|---|---|---|---|
| empty | B | empty | B | C | D |

After T1 deletes remote patrons of Cabana:

| C_lp | C_rp | OT_lp | OT_rp | NT_lp | NT_rp |
|---|---|---|---|---|---|
| empty | empty | empty | B | C | D |

The results from the given schedule and two serial executions of T1, T2 are compared in the following table.

| | C_lp | C_rp | OT_lp | OT_rp | NT_lp | NT_rp |
|---|---|---|---|---|---|---|

| | | | | | | |
|---|---|---|---|---|---|---|
| Result | empty | empty | empty | B | C | D |
| T1 → **T2** | empty | empty | empty | empty | A,C | B,D |
| T2 → **T1** | empty | empty | A | B | C | D |

The result is not equivalent to any of the serial schedule (T1 → T2, T2 → T1), so the given schedule is not serializable.