14.6 **Answer:** There is a serializable schedule corresponding to the precedence graph below, since the graph is acyclic. A possible schedule is obtained by doing a topological sort, that is, T_1, T_2, T_3, T_4, T_5.

14.7 **Answer:** A cascadeless schedule is one where, for each pair of transactions T_i and T_j such that T_j reads data items previously written by T_i, the commit operation of T_i appears before the read operation of T_j. Cascadeless schedules are desirable because the failure of a transaction does not lead to the aborting of any other transaction. Of course this comes at the cost of less concurrency. If failures occur rarely, so that we can pay the price of cascading aborts for the increased concurrency, noncascadeless schedules might be desirable.

15.10 **Answer:**

a. Serializability can be shown by observing that if two transactions have an I mode lock on the same item, the increment operations can be swapped, just like read operations. However, any pair of conflicting operations must be serialized in the order of the lock points of the corresponding transactions, as shown in Exercise 15.1.

b. The **increment** lock mode being compatible with itself allows multiple incrementing transactions to take the lock simultaneously thereby improving the concurrency of the protocol. In the absence of this mode, an **exclusive** mode will have to be taken on a data item by each transaction that wants to increment the value of this data item. An exclusive lock being incompatible with itself adds to the lock waiting time and obstructs the overall progress of the concurrent schedule.
In general, increasing the **true** entries in the compatibility matrix increases the concurrency and improves the throughput.

The proof is in Korth, "Locking Primitives in a Database System," JACM Vol. 30, 1983.

15.21 Most implementations of database systems use strict two-phase locking. Suggest three reasons for the popularity of this protocol.
Answer: It is relatively simple to implement, imposes low rollback overhead because of cascadeless schedules, and usually allows an acceptable level of concurrency.

16.2 Explain the purpose of the checkpoint mechanism. How often should checkpoints be performed? How does the frequency of checkpoints affect:

- System performance when no failure occurs?
- The time it takes to recover from a system crash?
- The time it takes to recover from a media (disk) failure?

Answer: Checkpointing is done with log-based recovery schemes to reduce the time required for recovery after a crash. If there is no checkpointing, then the entire log must be searched after a crash, and all transactions undone/redone from the log. If checkpointing had been performed, then most of the log-records prior to the checkpoint can be ignored at the time of recovery.

Another reason to perform checkpoints is to clear log-records from stable storage as it gets full.

Since checkpoints cause some loss in performance while they are being taken, their frequency should be reduced if fast recovery is not critical. If we need fast recovery checkpointing frequency should be increased. If the amount of stable storage available is less, frequent checkpointing is unavoidable.

Checkpoints have no effect on recovery from a disk crash; archival dumps are the equivalent of checkpoints for recovery from disk crashes.

16.10 Explain the reasons why recovery of interactive transactions is more difficult to deal with than is recovery of batch transactions. Is there a simple way to deal with this difficulty? (Hint: Consider an automatic teller machine transaction in which cash is withdrawn.)

Answer: Interactive transactions are more difficult to recover from than batch transactions because some actions may be irrevocable. For example, an output (write) statement may have fired a missile, or caused a bank machine to give money to a customer. The best way to deal with this is to try to do all output statements at the end of the transaction. That way if the transaction aborts in the middle, no harm will be have been done.

Output operations should ideally be done atomically; for example, ATM machines often count out notes, and deliver all the notes together instead of delivering notes one-at-a-time. If output operations cannot be done atomically, a physical log of output operations, such as a disk log of events, or even a video log of what happened in the physical world can be maintained, to allow perform recovery to be performed manually later, for example by crediting cash back to a customers account.

16.18 Consider the log in Figure 16.5. Suppose there is a crash just before the $< T_0$ abort$>$ log record is written out. Explain what would happen during recovery.

Answer: Recovery would happen as follows:

Redo phase:

a. Undo-List = T_0, T_1

b. Start from the checkpoint entry and perform the redo operation.

c. C = 600

d. T_1 is removed from the Undo-list as there is a commit record.

e. T_2 is added to the Undo list on encountering the $< T_2$ start $>$ record.

f. A = 400

g. B = 2000

Undo phase:

a. Undo-List = T_0, T_2

b. Scan the log backwards from the end.

c. A = 500; output the redo-only record $< T_2$, A, 500 $>$

d. output $< T_2$ abort $>$

e. B = 2000; output the redo-only record $< T_0$, B, 2000 $>$

f. output $< T_0$ abort $>$

At the end of the recovery process, the state of the system is as follows:

A = 500
B = 2000
C = 600

The log records added during recovery are:

$< T_2$, A, 500 $>$
$< T_2$ abort $>$
$< T_0$, B, 2000 $>$
$< T_0$ abort $>$

Observe that B is set to 2000 by two log records, one created during normal rollback of T_0, and the other created during recovery, when the abort of T_0 is completed. Clearly the second one is redundant, although not incorrect. Optimizations described in the ARIES algorithm (and equivalent optimizations described in Section 16.7 for the case of logical operations) can help avoid carrying out redundant operations, which create such redundant log records.

16.22 In the ARIES recovery algorithm:

a. If at the beginning of the analysis pass, a page is not in the checkpoint dirty page table, will we need to apply any redo records to it? Why?

b. What is RecLSN, and how is it used to minimize unnecessary redos?

Answer:

a. If a page is not in the checkpoint dirty page table at the beginning of the analysis pass, redo records prior to the checkpoint record need not be applied to it as it means that the page has been flushed to disk and been removed from the DirtyPageTable before the checkpoint. However, the page may have been updated after the checkpoint, which means it will appear in the dirty page table at the end of the analysis pass.
For pages that appear in the checkpoint dirty page table, redo records prior to the checkpoint may also need to be applied.

b. The RecLSN is an entry in the DirtyPageTable, which reflects the LSN at the end of the log when the page was added to DirtyPageTable. During the redo pass of the ARIES algorithm, if the LSN of the update log record encountered, is less than the RecLSN of the page in DirtyPageTable, then that record is not redone but skipped. Further, the redo pass starts at RedoLSN, which is the earliest of the RecLSNs among the entries in the checkpoint DirtyPageTable, since earlier log records would certainly not need to be redone. (If there are no dirty pages in the checkpoint, the RedoLSN is set to the LSN of the checkpoint log record.)