CS 245
SOLUTIONS to Final Exam – Winter 1999

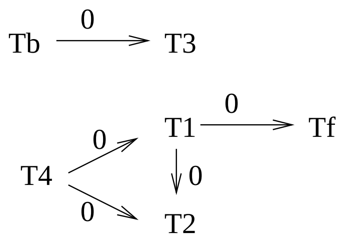
Problem 1 (10 points)

Consider the following execution schedule:

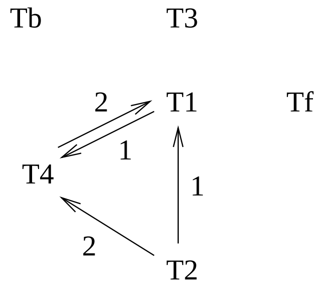
$$S = r_3(C)\ w_4(B)\ w_4(A)\ w_1(B)\ r_2(B)\ r_3(D)\ r_2(A)\ w_1(A).$$

You are trying to determine if this schedule is view serializable.

1. Show *all* the singleton arcs (those where there is no choice) of the labeled precedence graph for S. Do not show the poly-arcs. (Of course, you also need to show the nodes of the graph. Also, if two arcs are identical, please only show one.)



2. Show the poly-arcs for this labeled precedence graph. Do not include here the arcs you showed in your previous answer. (Again, show all nodes.)



3. Is schedule S view serializable? Briefly explain your answer.

   ANSWER: No. Whatever choice we make, we end up with a cycle.

Problem 2 (10 points)

Megatron Research Labs have developed a compression scheme for the tuples in their relations. All tuples in a particular relation of interest are a fixed length of 100 bytes.

It is discovered that the compressed tuples in this relation will vary in length between 20 bytes and 50 bytes. Assume there are no duplicate keys in the relation. Each disk block has 2000 bytes of available storage.

1. Each compressed block requires some overhead to describe the position of each record. Suppose that the position is determined by a single byte (a relative distance from the previous record) — this is stored in a position table at the start of the block. Assuming the position table size is fixed for all blocks, how much space do we need to allocate for this position table. Be exact!

   ANSWER: 95 bytes

2. The compressed tuples are sorted and stored in a sequential file. A primary, sparse sequential index is built on this file. Compare the size of this index with the size of an index on the uncompressed file. Consider both worst and best case scenarios for the index on the compressed file. Assume the number of tuples in the relation is large.

   WORST CASE: $\frac{20}{95}$ BEST CASE: $\frac{20}{38}$

3. A secondary index is used on the file of compressed tuples. Compare the size of this index with the size of a secondary index on the uncompressed file. Consider both worst and best case scenarios for the index on the compressed file.

   WORST CASE: 1 BEST CASE: 1

Problem 3 (10 points)

Suppose that we run the following transactions using the validation protocol. Assume that the validation protocol has been extended, so that the condition $T_i \notin$ FIN includes the condition that the write sets involved do not intersect. (The version in the textbook already includes this optimization.) The following table lists the transactions involved, together with their read and write sets:

Transaction	Read Set	Write Set
T_1	$\{a, b\}$	{b, c}
T_2	$\{a, b\}$	{d}
T_3	$\{c, d\}$	{e}
T_4	$\{a\}$	{c}
T_5	$\{a\}$	{g}
T_6	$\{a\}$	{c}

The following sequence of events takes place. (No other transaction run before or concurrently with T_1 ... T_6.)

1. T_1, T_2, T_3, T_4 start (in this order)
2. T_1 initiates validation
3. T_2 initiates validation
4. T_5 starts
5. T_3 initiates validation
6. T_4 initiates validation
7. T_5 initiates validation
8. T_1, T_2, T_3, T_4 finish (if they were not aborted earlier)
9. T_6 starts
10. T_6 initiates validation
11. T_5, T_6 finish (if they were not aborted earlier)

In the table on the next page, please write down the outcome of each transaction, either COMMITTED if it was successful at validation, or ABORTED if it was not.

Transaction	Outcome
T_1	Committed
T_2	Aborted
T_3	Aborted
T_4	Aborted
T_5	Committed
T_6	Committed

Problem 4 (10 points)

Please answer the following questions:

1. Consider a $B+$ tree of order 100. A particular non-leaf node has 95 pointers. How many keys will that node contain?

   ANSWER:______ 94

2. Consider the same $B+$ tree of order 100. A particular leaf node has 95 pointers. How many keys will that node contain?

   ANSWER:______ 94

3. Consider a B tree (no +). A particular non-leaf, non-root node has 95 pointers. How many keys will that node contain?

   ANSWER:______ 47

4. Consider the same B tree (no +). A particular leaf node (not the root) has 95 pointers. How many keys will that node contain?

   ANSWER:______ 95 (94 also got credit)

5. Consider an linear hashing structure with $i = 10$ and $m = 903$, and no overflow buckets. How many disk buckets are actually allocated at this point?

   ANSWER:______ Skipped, but answer was 904

6. Consider an extensible hashing structure with $i = 4$ and 13 actual disk blocks allocated. (Assume directory is entirely in memory.) Within each of these 13 blocks, we have a counter i_j that records the "level" of the block, in general with values between 1 and $i = 4$. *At most* how many of the 13 blocks have a counter i_j *different* from 4?

ANSWER:______ 3

7. Consider a file with 1000 fixed-size records, each 300 bytes long. This file is to be stored on 1K disk blocks, with no spanning of records. How many blocks will be needed?

ANSWER:______ 334

8. Consider a disk with a 10 inch diameter, 3600 RPM speed, 6 surfaces, 1024 cylinders, average seek time of 25ms, 4KB blocks, 1GB usable capacity, and 10% overhead for gaps. How many tracks (total over the entire disk) does this disk have?

ANSWER:______ 6144

9. Consider the same disk (10 inch diameter, 3600 RPM speed, 6 surfaces, 1024 cylinders, average seek time of 25ms, 4KB blocks, 1GB usable capacity, and 10% overhead for gaps). What is the average rotational delay to read a block, assuming we always need to find the start of the track?

ANSWER:______ 1/60 seconds or 17ms.

10. We are designing the record layout for records with a variable number of fields. The data content of each field is fixed length, 5 bytes. With a variable format record, we need to add one byte to the record to give the number of fields, plus we need to add 1 byte to each field to describe it. For fixed format records, we do not need any additional bytes; however, we have to allocate space for a maximum of 100 fields. Say records have on the average k fields out of the possible 100. What is the minimum value of k at which the fixed format is better?

ANSWER:______ 84 (k should be greater than 83.16)

Problem 5 (10 points)

Your company sells a database system that uses pure redo logging for crash recovery. You have been asked to design a recovery mechanism for "disk crashes." In a disk crash, not only does the computer stop and the contents of memory are lost, but the contents of the database on disk are also lost. To recover, you rely on a backup database copy made to *tape* earlier.

The process for making a backup database tape is as follows:

1. You write a "start backup" record on the log. This record contains a list L_A of all active transactions, i.e., of all transactions that started before the "start backup" record was written, but that had not written their commit record to the log. You save a pointer P_S to the "start backup" log record in a very safe place (that will survive disk crashes), so that you can later on figure out where this record is.

2. You wait one hour, and abort any transactions in L_A that have not committed or aborted on their own. Once all transactions in L_A have written their commit (or abort) records to the log, then you write a "ready" record to the log. You also record the position of this record, P_R in a safe place. Note that new transactions (not in L_A) could have started during the one hour wait. These are *not* aborted.)

3. You start writing the value of every database object to the backup tape. For each object, you ensure that the latest committed value (found on disk or in memory) is writen to tape. During this process you continue processing transactions as usual.

4. When all objects have been written to tape, you write an "end backup" record to the log, and you record its position, P_E, in a safe place.

5. This completes the backup process. Later on you may wish to do another backup, but for this problem, assume that only one backup was performed successfully. When a disk crash occurs, you have available the P_S, P_R, P_E pointers, as well as a pointer to the very begining of the log, P_B, and a pointer to final record written before the crash, P_F.

(Note: before you try answering one of two questions on next page, read *both* questions.)

1. Write pseudo code to restore the database after a disk crash. You want to create a new database (on a good disk) that reflects *all* committed transactions. *Your code should not redo actions unnecessarily.* You start by scanning the log once to find the following sets:

   - C_B: the set of transactions whose commit record is between P_B and P_S.
   - C_S: the set of transactions whose commit record is between P_S and P_R.
   - C_R: the set of transactions whose commit record is between P_R and P_E.
   - C_E: the set of transactions whose commit record is between P_E and P_F (inclusive).

   In your code, you can assume that these sets have been computed and are available for use. Make sure your code is very clear and well structured. details like the order in which you scan the log (if you do scan the log) are important. You may lose credit for lack of clarity, even if the code is correct.

   (Step 1) Copy all objects on backup tape to a new database.

   (Step 2) Starting at P_S, scan the log forward until P_F. For each redo record found, if it corresponds to a transaction in C_R or C_E, redo the action on the new database.

2. The log can be divided into the four regions given below. For each region, write "YES" if you algorithm redoes actions in that region, "NO" otherwise. Explain briefly why it is OK to do not not do these actions.

   - Records between P_B and P_S.

     NO. We only need to redo $C_R \cup C_E$ transactions. These trasnaction started after P_S so they have no undo records in this region.

   - Records between P_S and P_R.

     YES. Transactions in C_R may have started in this region, so we need to undo their redo records from this region.

   - Records between P_R and P_E.

     YES. The trasnactions we are redoing may have records in this region.

   - Records between P_E and P_F (inclusive).

     YES. The trasnactions we are redoing may have records in this region.

Problem 6 (10 points)

A data warehouse contains a fact table $sales(P, S, A)$, representing the sales amount A for product P at store S. The amount A is the measure here. The warehouse also contains a dimension table $store(S, C, R)$ giving the city C and region R for each store S.

At the warehouse we join *sales* and *store* (natural join) to obtain view $V(P, S, C, R, A)$. This view is materialized.

At the warehouse we also materialize the sub-cube W of CUBE(V) that contains all tuples for the form $(*, *, C, R, A)$ (using the textbook's notation). The aggregation is the sum of the A fields.

1. For each of the following desired values, indicate if it is more efficient (or even feasible) to compute them from V or from W.

   (a) Find the total sales in the south, i.e., find $(*, *, *, \text{"south"}, A)$.
   This is computed more efficiently from W.

   (b) For all products, find their total sales in the southern region, i.e., find all tuples of the form $(P, *, *, \text{"south"}, A)$ for any values of P, A.
   This must be computed from V. It cannot be computed from W.

   (c) Find the tuple of the form $(*, *, \text{"atlanta"}, \text{"south"}, A)$ for any value of A.
   This is computed more efficiently from W.

   (d) Find all tuples of the form $(P, *, *, *, A)$ for any values of P, A.
   This must be computed from V. It cannot be computed from W.

2. Explain very briefly, in English, what the difference between the following pairs of expressions.

   (a) ("chair", $*, *, *, A$) and ("chair", $*, *, *, *$).
   ("chair", $*, *, *, A$) represents the total amount sold for chiars, across all stores and cities. ("chair", $*, *, *, *$) is not a very useful tuple. It simply reports if the product "chair" exisits in the database.

   (b) The table containing all ("chair", $*, *, R, A$) tuples and the table containing all ("chair", $*, C, R, A$) tuples.
   The first table give the total sales of "chairs" by region. The second table gives the total sales of "chairs" by city, also listing the region where each city is located.

   (c) The table containing all ("chair", $*, C, *, A$) tuples and the table containing all ("chair", $*, C, R, A$) tuples.
   Both of these tables give the total sales of "chairs" by city. The only difference is that the second table also lists the region that each city is in. Notice that a city can only be in one region, because *store* is a dimension table.

Problem 7 (10 points)

Consider three relations $R(a, b)$, $S(b, c, d)$, and $U(d, e)$ with the following statistics:

- R has 1000 tuples. $V(R, a) = 30$, $V(R, b) = 20$. (This uses the textbook's notation. In class, we would have said $V(a, R) = 30$, $V(b, R) = 20$.)
- S has 500 tuples. $V(S, b) = 30$, $V(S, c) = 100$, $V(S, d) = 50$.
- U has 2000 tuples. $V(U, d) = 30$, $V(U, e) = 200$.

1. Estimate the number of tuples in $T_1 = R \bowtie S \bowtie U$, when the joins are done in this order. Briefly explain how you get your answer. (You do not need to perform multiplications and divisions. Just write down the right expression.)

   The answer is $(1000 \times 500 \times 2000)/(30 \times 50)$. See Example 7.25 in textbook for an explanation.

2. What is the expected number of tuples for $T_1 = U \bowtie S \bowtie R$, when done in this order? Explain.

   The number of tuples is the same as before. See Example 7.25 in textbook for an explanation.