CSc 560 — Database Systems Implementation
Fall 2022 (McCann)

http://u.arizona.edu/~mccann/classes/560

Program #3: Transaction Schedule Conflict Serializability

(100 points)

Due Date: Thursday, November 17th, 2022, *at the beginning of class*

Overview: We've (very) recently covered the ideas of a transaction of sometimes–conflicting operations, serial schedules of transactions, schedules of interleaved transaction operations, conflict serializability, precedence (a.k.a. serialization) graphs, and topological sorting. This assignment combines these ideas in two related but distinct ways.

For this assignment, our transactions will be provided as sequences of read, write, and commit operations, one operation per line of a text file. Similarly, our schedules will be given as text files containing the operations of one or more potentially interleaved transactions. Operations have the format <*op*><*id*><*space*(*s*)><*item*>, where <*op*> is one of `r` (for read), `w` (write), or `c` (commit); <*id*> is an integer that identifies the transaction executing <*op*>; <*item*> is a single upper–case letter representing the DB item being read or written (and is absent if <*op*> is `c`); and <*space*(*s*)> are one or more spaces (also absent when <*op*> is `c`). The first example, below left, is a collection of transactions: Transaction 19 reads items A and B and writes item C, and transaction 8 reads B and C and writes A. The second example, below right, is a legal interleaved schedule of the same two transactions:

```
r19 A     |     r8  B
r19 B     |     r19 A
w19 C     |     r8  C
c19       |     w8  A
          |     r19 B
r8  B     |     c8
r8  C     |     w19 C
w8  A     |     c19
c8        |
```

Assignment: Using your choice of programming language that is installed on lectura for the use of all users, write a complete, well–documented program named `prog3` (using the appropriate capitalization and filename extension for your language) that accomplishes both of the following tasks, at the discretion of the user based on the command–line arguments provided.

Task 1 If the command–line argument is the path and filename of a file with the extension `.sch`, your program is to do the following with the schedule of transactions contained within the file:

1. Read the schedule from the schedule file
2. Construct the precedence graph described by the schedule
3. Check the graph for cycle(s) to determine whether or not the schedule is conflict serializable
4. If the schedule is conflict serializable, topologically sort the graph to produce a serializability order of the schedule's transactions
5. Report a list of the IDs of the transactions in the schedule file, in ascending order by ID; and either a message stating that the schedule is not conflict serializable and the IDs of the transactions participating in the detected cycle, or a message stating that the schedule is conflict serializable and a topological ordering of the conflict serializable schedule.

(Continued...)

You may use your choices of the data structure(s) to hold the schedule and the data structure(s) to represent the precedence graph. Because not everyone is using the same programming language, to level the playing field, you are required to implement all of the graph algorithms yourself — that is, you may not use any pre–written graph libraries, APIs, etc., including any that may be provided with your language and any third–party packages, to perform cycle–detection and topological sorting. You may consult references to learn about appropriate algorithms, but, again, you must implement them yourself.

Task 2 If the command–line arguments are the path/filename of a file with the extension `.set`, a non–negative integer, and an inclusive range of positive integers $a - b$ where $a \leq b$ (e.g., `3-10` means 3, 4, ..., 9, 10), your program is to do the following with the collection of transactions contained within the file (formatted as in the first example, above):

1. Read the collection of transactions from the schedule file
2. Loop for a number of times equal to the non–negative integer:
   (a) Create a legal schedule of the transactions by creating a random interleaving of the operations of the transactions (see below for the details)
   (b) Determine whether or not the schedule is conflict serializable
3. Report the command–line arguments, the quantity of schedules that were conflict serializable, and the percentage of schedules that were conflict serializable.

Here's how you are to create the randomly–interleaved schedule: Until all transaction operations have been added to the schedule, generate a random number t to select a transaction that still has operations to be included in the schedule, and a second random number r in the range $a - b$ (inclusive). Add the next r operations from t to the schedule. If t doesn't have r operations remaining, add all of its remaining operations.

For example, using transactions 19 and 8 (hereafter, T_{19} and T_8) from above, and a range of `1-3`: We randomly select the first transaction (T_{19}) and $r = 1$, and add `r19 A` (hereafter, $r_{19}[A]$) to the schedule. The next random selections are transaction 8 and $r = 3$, so we add $r_8[B]$, $r_8[C]$, and $w_8[A]$ to the schedule. Next, we again select transaction 8, with $r = 2$. There's only one operation remaining in 8 (c_8), so we add it to the schedule. At this point, there's only one transaction remaining with un–added operations, so we can add the rest of T_{19}'s operations to the schedule (or just continue generating random values until all of T_{19}'s operations have been added). The resulting schedule is:

$r_{19}[A]$ $r_8[B]$ $r_8[C]$ $w_8[A]$ c_8 $r_{19}[B]$ $w_{19}[C]$ c_{19}

Data: The expected command–line arguments are given above with the descriptions of each task. To illustrate:

Task 1's execution is enabled via a path/filename command–line argument with a `.sch` extension, such as (assuming Python3). For example:

```
python3 prog3.py /home/johndoe/cs560/program3/data/2xact.sch
```

Task 2's execution is enabled via a sequence of three arguments: A path/filename of the `.set` file, a non–negative integer, and a positive integer range $a - b$, where $a \leq b$. For example:

```
python3 prog3.py collection9.set 10000 1-1
```

The expected formatting of the content of the `.sch` and `.set` files is given in the examples in the Overview section. Code your program to accept an optional blank line between transactions in the `.set` files, as shown in the example. Those blank lines are nice for humans who have to read the content of such files.

(Continued...)

Output: Restating from the task descriptions, above:

For Task 1, output the following, using an output format of your design: (a) A list of the IDs of the transactions in the schedule file, in ascending order by ID; and (b) either a message stating that the schedule is not conflict serializable and the IDs of the transactions in the detected cycle, or a message stating that the schedule is conflict serializable and a topological ordering of the conflict serializable schedule.

For Task 2, output, again using an output format of your design, (a) the command–line arguments, (b) the quantity of schedules that were conflict serializable, and (c) the percentage of schedules that were conflict serializable.

Hand In: Using the `turnin` facility on lectura, submit your completed program file(s) and the `.set` and `.sch` files you used for testing. The submission folder is `cs560p3`. Instructions are available from the document of submission instructions linked to the class web page. In particular, because I will be grading your program on lectura, it needs to run on lectura, so be sure to test it on lectura. Feel free to split up your code over additional files if doing so is appropriate to achieve good code modularity. Submit all files as–is, *without* packaging them into `.zip`, `.jar`, `.tar`, etc., files.

Other Requirements and Hints:

- As I can't think of anything not already covered above, I'll repeat the basics from the first two programming assignments: You may use any language you wish to implement your program, so long as it is installed for general use on lectura; you can make use of resources that explain how to use your language or explain how algorithms work, but not resources that supply already–coded solutions to all or parts of the assignment; sharing test cases and output with your classmates is encouraged; test your code thoroughly on multiple inputs; document well; late days can be used if you have some remaining; and, as always, **Start early!**