Northeastern University
CS5200 – Database Management Systems
Fall 2017, Derbinsky

HW6

This assignment has eight (8) problems worth 100 points total. Notes:

- To receive credit, submit to Blackboard **a single ZIP file** that contains a **a single PDF** document, containing your responses to each of the problems.
- You must typeset all responses.

Transactions

Problem 1 (20 points). Consider the following schedule...

$$r_1(A), r_2(B), w_1(A), w_1(B), r_2(C), w_1(A), c_1, r_3(B), w_3(B), c_3, w_2(C), c_2$$

Assume that all writes to A, B, and C prior to the operations above have already committed. Provide an answer, with justification, for each of the following questions.

a. Is this schedule serial?
b. Is this schedule conflict serializable?
c. Is this schedule conflict-equivalent to a serial schedule? (if so, which one?)
d. Is this schedule recoverable?
e. Is this schedule cascadeless?

Problem 2 (15 points). Consider the following transactions ...

$$T_1 : r(B), r(A), w(A)$$
$$T_2 : r(B), w(B), r(A)$$

List all schedules that are conflict equivalent to serial schedule (T_2, T_1).

Problem 3 (15 points). Consider the following constraint: $X < Y$

For each of the following transactions, state whether or not they preserve the consistency of the database. If not, supply an example of initial values and what they result in after the transaction completes.

a. $X = 5X; Y = 5Y$
b. $X = 3X; Y = Y + 25$
c. $X = X - 10; Y = Y + X$

Problem 4 (10 points). At the time of a system failure, let the following reflect the pertinent UNDO/REDO log on disk ...

```
<T1, START>
<T1, X, 110, 8>
<T1, COMMIT>
<T2, START>
<T2, X, 8, 2>
<START CKPT(T2)>
<T2, Z, 120, 9>
<T3, START>
<T2, COMMIT>
<T3, X, 2, 6>
<END CKPT>
<T3, Z, 9, 30>
<T4, START>
<START CKPT(T3, T4)>
<T3, COMMIT>
<T4, Z, 30, 7>
```

What are the values of X and Z in the database at the end of recovery?

Security

In `HW6.jar`, you have been supplied a very simple MD5 password cracking utility for use in this assignment. The goal is to understand that MD5 hashing is NOT a secure method of password storage. There are two modes – MD5 hashing and MD5 cracking.

To generate an MD5 hash for a password, simply run the following command (assumes Java 8), supplying your password in quotes:

```
$ java -cp HW6.jar edu.neu.ccs.course.cs5200.MD5 "pass"
1A1DC91C907325C69271DDF0C944BC72
```

To crack a password, you have some options. First, you can check against a pre-computed table of 2000 common passwords[1]:

```
$ java -cp HW6.jar edu.neu.ccs.course.cs5200.MD5Cracker 1A1DC91C907325C69271DDF0C944BC72
Start: 2017/11/24 11:49:40
Password: pass
Searched: 39
Stop: 2017/11/24 11:49:40
```

2000 is rather a small number for such tools, but does illustrate the speed with which this simple password was found (under 1 second). Typical tools also have dictionary attacks, common misspellings, 1337 speak, etc.

To perform a brute-force attack:

```
$ java -cp HW6.jar edu.neu.ccs.course.cs5200.MD5Cracker a-4-4 1A1DC91C907325C69271DDF0C944BC72
Start: 2017/11/24 11:50:37
Generating strings of length [4, 4] using [a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p,
q, r, s, t, u, v, w, x, y, z]
Searching length 4...
Password found: pass
Searched: 328,552
Stop: 2017/11/24 11:50:37
```

This tells the program to use lower-case letters (`a`) of length 4. As you can see, while it has to generate a few hundred thousand MD5 hashes, this likely will take a fraction of a second. But in general, an attacker won't be sure about characters/lengths, so let's be a bit more unconstrained ...

```
$ java -cp HW6.jar edu.neu.ccs.course.cs5200.MD5Cracker aA1-1-6 1A1DC91C907325C69271DDF0C944BC72
Start: 2017/11/24 11:56:17
Generating strings of length [1, 6] using [a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p,
q, r, s, t, u, v, w, x, y, z, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T,
U, V, W, X, Y, Z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Searching length 1...
Searching length 2...
Searching length 3...
Searching length 4...
Password found: pass
Searched: 4,601,346
Stop: 2017/11/24 11:56:18
```

[1] `http://www.passwordrandom.com`

This command allowed for lower- and upper-case, as well as numbers, ranging from length 1 to 6. The password was successfully found in under 2 seconds after generating 4.6 million hashes.

Problem 5 (5 points). Before iOS 9, Apple's default passcode configuration included 4 numbers from 0 – 9. Show the math to compute how many distinct passcodes this yields. Show your command and output for how long it takes to search all these passcodes. Show your command and output for how long it takes to crack `4321` using both the brute-force and common-password approaches.

Problem 6 (5 points). Apple's default passcode configuration now is 6 numbers from 0 – 9. Show the math to compute how many distinct passcodes this yields. Show your command and output for how long it takes to search all these passcodes. Show your command and output for how long it takes to crack `654321` using both the brute-force and common-password approaches.

Problem 7 (5 points). Show your command and output for how long it takes to search all passwords with lower-case and upper-case letters of length 5 on your machine. Given this output, estimate how many MD5 hashes can be generated per second. (Note: this program only uses 1 CPU core.) Now estimate the max length of password (using lower-case, upper-case, numbers) that could be cracked via brute force in a day on your machine using this program – show the math required to compute this estimate.

Problem 8 (25 points). You have been provided the following list of MD5 password hashes from an authentication database breach. You have been told that the source was an organization that limited passwords to 4–8 lower-case letters. For each entry, identify the corresponding password with as little time/computational resource usage possible. If you used the supplied tool, show the command used and output; if not, indicate how you determined the password.

a. `5F4DCC3B5AA765D61D8327DEB882CF99`

b. `0782EFD61B7A6B02E602CC6A11673EC9`

c. `5D41402ABC4B2A76B9719D911017C592`

d. `20EE80E63596799A1543BC9FD88D8878`

e. `5F4DCC3B5AA765D61D8327DEB882CF99`