CSE 332: Data Structures and Parallelism

Section 8: Concurrency + Complexity

Part One: Concurrency

1. User Profile

You are designing a new social-networking site to take over the world. To handle all the volume you expect, you want to support multiple threads with a fine-grained locking strategy in which each user's profile is protected with a different lock. At the core of your system is this simple class definition:

```
class UserProfile {
    static int id_counter;
    int id; // unique for each account
    int[] friends = new int[9999]; // horrible style
    int numFriends;
    Image[] embarrassingPhotos = new Image[9999];

    UserProfile() { // constructor for new profiles
        id = id_counter++;
        numFriends = 0;
    }

    synchronized void makeFriends(UserProfile newFriend) {
        synchronized(newFriend) {
            if(numFriends == friends.length
            || newFriend.numFriends == newFriend.friends.length)
                throw new TooManyFriendsException();
                friends[numFriends++] = newFriend.id;
                newFriend.friends[newFriend.numFriends++] = id;
        }
    }

    synchronized void removeFriend(UserProfile frenemy) {
        ...
    }
}
```

a) The constructor has a concurrency error. What is it and how would you fix it? A short English answer is enough - no code or details required.

b) The `makeFriends` method has a concurrency error. What is it and how would you fix it? A short English answer is enough no code or details required.

2. Bubble Tea

The `BubbleTea` class manages a bubble tea order assembled by multiple workers. Multiple threads could be accessing the same `BubbleTea` object. Assume the Stack objects are thread-safe, have enough space, and operations on them will not throw an exception.

```java
public class BubbleTea {
    private Stack<String> drink = new Stack<String>();
    private Stack<String> toppings = new Stack<String>();
    private final int maxDrinkAmount = 8;

    // Checks if drink has capacity
    public boolean hasCapacity() {
        return drink.size() < maxDrinkAmount;
    }

    // Adds liquid to drink
    public void addLiquid(String liquid) {
        if (hasCapacity()) {
            if (liquid.equals("Milk")) {
                while (hasCapacity()) {
                    drink.push("Milk");
                }
            } else {
                drink.push(liquid);
            }
        }
    }

    // Adds newTop to list of toppings to add to drink
    public void addTopping(String newTop) {
        if (newTop.equals("Boba") || newTop.equals("Tapioca")) {
             toppings.push("Bubbles");
        } else {
            toppings.push(newTop);
        }
    }
}
```

a) Does the `BubbleTea` class above have (circle all that apply):

a race condition | potential for deadlock | a data race | none of these

If there are any problems, give an example of when those problems could occur. Be specific!

b) Suppose we made the `hasCapacity` method synchronized, and changed nothing else in the code. Does this modified BubbleTea class above have (circle all that apply):

a race condition | potential for deadlock | a data race | none of these

If there are any FIXED problems, describe why they are FIXED. If there are any NEW problems, give an example of when those problems could occur. Be specific!

3. Phone Monitor

The `PhoneMonitor` class tries to help manage how much you use your cell phone each day. Multiple threads can access the same `PhoneMonitor` object. Remember that synchronized gives you reentrancy.

```
public class PhoneMonitor {
    private int numMinutes = 0;
    private int numAccesses = 0;
    private int maxMinutes = 200;
    private int maxAccesses = 10;
    private boolean phoneOn = true;
    private Object accessesLock = new Object();
    private Object minutesLock = new Object();

    public void accessPhone(int minutes) {
        if (phoneOn) {
            synchronized (accessesLock) {
                synchronized (minutesLock) {
                    numAccesses++;
                    numMinutes += minutes;
                    checkLimits();
                }
            }
        }
    }

    private void checkLimits() {
        synchronized (minutesLock) {
            synchronized (accessesLock) {
                if (numAccesses >= maxAccesses
                  || numMinutes >= maxMinutes) {
                    phoneOn = false;
                }
            }
        }
    }
}
```

a) Does the `PhoneMonitor` class as shown above have (circle all that apply):

a race condition	potential for deadlock	a data race	none of these

If there are any problems, give an example of when those problems could occur. Be specific!

b) Suppose we made the `checkLimits` method public, and changed nothing else in the code. Does this modified `PhoneMonitor` class have (circle all that apply):

a race condition	potential for deadlock	a data race	none of these

If there are any FIXED problems, describe why they are FIXED. If there are any NEW problems, give an example of when those problems could occur. Be specific!

4. Time Machine

The `TimeMachine` class tries to help manage how much you use your cell phone each day. Multiple threads can access the same `TimeMachine` object.

```
public class TimeMachine {
    private int now = 1985;
    private int future = 2015;
    private int energy = 100;

    public boolean hasEnergy() {
        return energy >= 100;
    }

    public void adjustEnergy(int charge) {
        if (energy + charge < 0 ) { // energy should never be negative
            return;
        }
        energy = energy + charge;
    }

    public void setFuture(int newFuture) {
        future = newFuture;
    }
}
```

a) Does the `TimeMachine` class as shown above have (circle all that apply):

a race condition	potential for deadlock	a data race	none of these

If there are any problems, give an example of when those problems could occur. Be specific!

b) Suppose we add a new method called `backToTheFuture`, and changed nothing else in the code.

```java
public class TimeMachine {
    private int now = 1985;
    private int future = 2015;
    private int energy = 100;
    public boolean hasEnergy() {
        return energy >= 100;
    }
    public void adjustEnergy(int charge) {
        if (energy + charge < 0 ) { // energy should never be negative
            return;
        }
        energy = energy + charge;
    }
    public void setFuture(int newFuture) {
        future = newFuture;
    }
    public boolean backToTheFuture() {
        if (!hasEnergy() && now != future) {
            return false;
        }
        now = future;
        energy = energy – 100;
        System.out.println("Heading to:" + future + " Energy remaining:" +energy);
        return true;
    }
}
```

Does this modified `TimeMachine` class have (circle all that apply):

a race condition   potential for deadlock   a data race   none of these

If there are any FIXED problems, describe why they are FIXED. If there are any NEW problems, give an example of when those problems could occur. Be specific!

c) Modify the code to use locks to allow the most concurrent access and to avoid all of the potential problems listed above.

Part Two: Complexity

1. P & NP Membership

For problems A and B below, show that they belong to both P and NP. Show that problem C belongs to NP.

a) Problem A: Given a list of 2-dimensional points, return true or false to indicate whether some pair of points have a distance of less than 5.

Belongs to P

Belongs to NP

b) Problem B: Given a list of integers, return true or false to indicate whether any items are duplicated (so return true if some value appears at least twice, false if all are distinct).

Belongs to P

Belongs to NP

c) Problem C: Given a weighted graph, a pair of nodes X and Y, and a number k, return true or false to indicate whether there is a path from X to Y with a cost of at least k

Belongs to NP

2. Practice

If A polynomial-time reduces to B and B is NP-Hard then A is NP-Hard.

True False

If B is NP-Hard and there exists a polynomial time algorithm for B, then P=NP.

True False

If B is NP-Hard and there does not exist a polynomial time algorithm for B, then P does not equal NP.

True False

If A reduces to B in polynomial time, and B reduces to C in polynomial time, and A is NP-Hard, then C is NP-Hard.

True False

If A reduces to B in polynomial time, and B reduces to C in polynomial time, and A is NP-Complete, then C is NP-Complete.

True False

If A reduces to B in polynomial time, and B reduces to C in polynomial time, and A is in EXP, then C is EXP.

True False

If A reduces to B in polynomial time, and B reduces to C in polynomial time, and A is in P, then C is P.

True False

If A reduces to B in polynomial time, and B reduces to C in polynomial time, and A is in NP, and C is in P, then P=NP.

True False

If A reduces to B in polynomial time, and B reduces to C in polynomial time, and A is in NP-Hard, and C is in P, then P=NP.

True False