Section 9: Concurrency & Graphs

0. 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:

```
1  class UserProfile {
2      static int id_counter;
3      int id; // unique for each account
4      int[] friends = new int[9999]; // horrible style
5      int numFriends;
6      Image[] embarrassingPhotos = new Image[9999];
7
8      UserProfile() { // constructor for new profiles
9          id = id_counter++;
10         numFriends = 0;
11     }
12
13     synchronized void makeFriends(UserProfile newFriend) {
14         synchronized(newFriend) {
15             if(numFriends == friends.length
16             || newFriend.numFriends == newFriend.friends.length)
17                 throw new TooManyFriendsException();
18                 friends[numFriends++] = newFriend.id;
19                 newFriend.friends[newFriend.numFriends++] = id;
20         }
21     }
22
23     synchronized void removeFriend(UserProfile frenemy) {
24         ...
25     }
26 }
```

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.

1. 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.

```
1  public class BubbleTea {
2      private Stack<String> drink = new Stack<String>();
3      private Stack<String> toppings = new Stack<String>();
4      private final int maxDrinkAmount = 8;
5
6      // Checks if drink has capacity
7      public boolean hasCapacity() {
8          return drink.size() < maxDrinkAmount;
9      }
10
11     // Adds liquid to drink
12     public void addLiquid(String liquid) {
13         if (hasCapacity()) {
14             if (liquid.equals("Milk")) {
15                 while (hasCapacity()) {
16                     drink.push("Milk");
17                 }
18             } else {
19                 drink.push(liquid);
20             }
21         }
22     }
23
24     // Adds newTop to list of toppings to add to drink
25     public void addTopping(String newTop) {
26         if (newTop.equals("Boba") || newTop.equals("Tapioca")) {
27              toppings.push("Bubbles");
28         } else {
29             toppings.push(newTop);
30         }
31     }
32 }
```

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 addTopping 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!

2. 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.

```
1  public class PhoneMonitor {
2      private int numMinutes = 0;
3      private int numAccesses = 0;
4      private int maxMinutes = 200;
5      private int maxAccesses = 10;
6      private boolean phoneOn = true;
7      private Object accessesLock = new Object();
8      private Object minutesLock = new Object();
9
10     public void accessPhone(int minutes) {
11         if (phoneOn) {
12             synchronized (accessesLock) {
13                 synchronized (minutesLock) {
14                     numAccesses++;
15                     numMinutes += minutes;
16                     checkLimits();
17                 }
18             }
19         }
20     }
21
22     private void checkLimits() {
23         synchronized (minutesLock) {
24             synchronized (accessesLock) {
25                 if (numAccesses >= maxAccesses
26                   || numMinutes >= maxMinutes) {
27                     phoneOn = false;
28                 }
29             }
30         }
31     }
32 }
```

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!

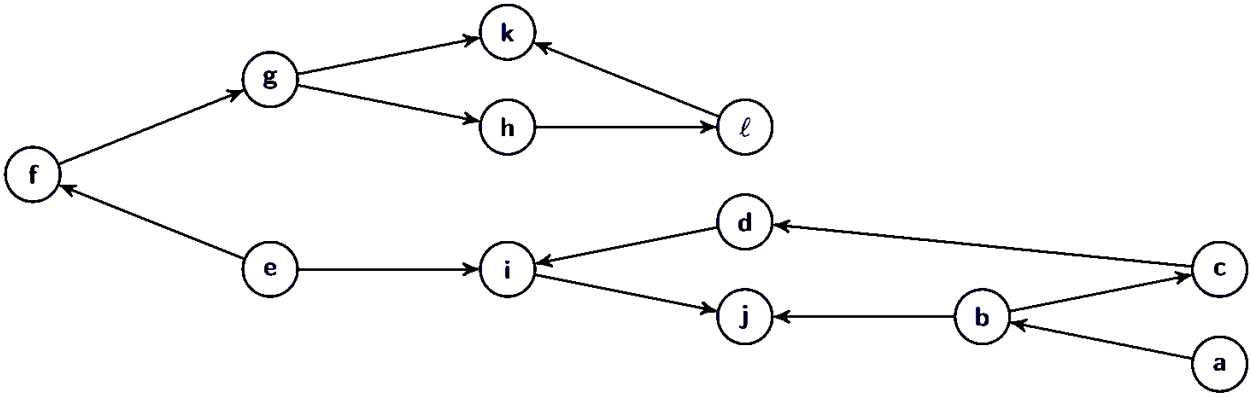
b) Suppose we made the **`checkLimits`** method public, and changed nothing else in the code. Does this modified **`PhoneMonitor`** class have (circle all that apply):

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!

3. It Rhymes with Flopological Sort

Consider the following graph:



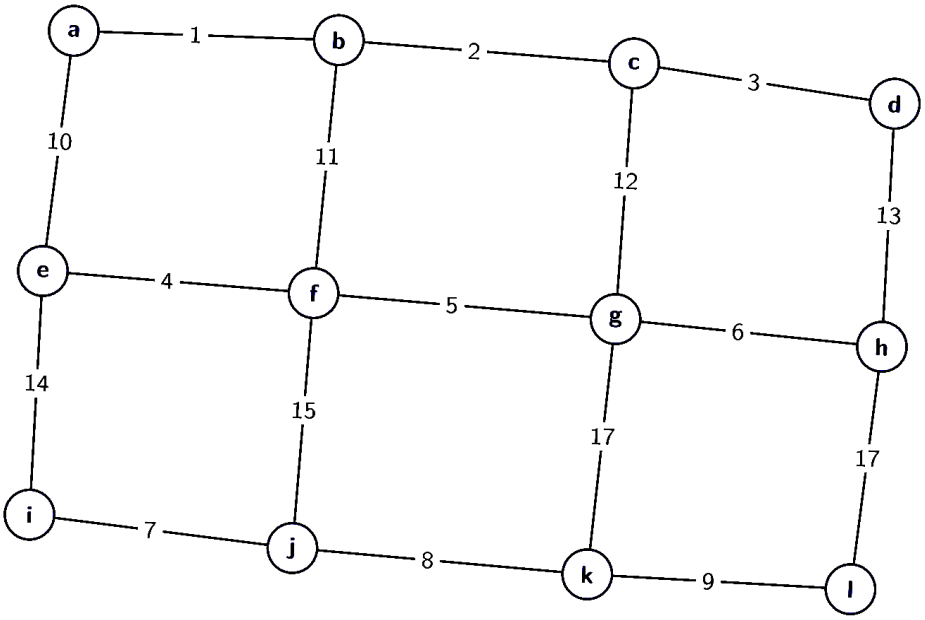
a) Does this graph have a topological sort? Explain why or why not. If you answered that it does not, remove the **MINIMUM** number of edges from the graph necessary for there to be a topological sort and carefully mark the edge(s) you are removing. Otherwise, just move on to the next part.

For the remaining parts, work with this (potentially) new version of the graph.

b) Find a topological sort of the graph. Do not bother showing intermediary work.

4. LMNST!

Consider the following graph:



a) Find an MST of this graph using both of the two algorithms we've discussed in lecture. Make sure you say which algorithm you're using and show your work.

b) Using just the graph, how can you determine if it's possible that there are multiple MSTs of the graph? Does this graph have multiple MSTs?

c) What is the asymptotic runtime of the algorithms that you used to compute the MSTs?