[15pts] 6. Variations on Dining Philosophers

Recall the five Dining Philosophers (and five chopsticks) sitting around a circular table: if they all pick up their right chopstick before their left, they can end up in a deadlocked situation.

a) [4] In the box to the right, draw a Resource Allocation Graph showing the Philosophers and chopsticks and a deadlocked situation.

b) [3] Assume L philosophers are left-handed and pick up their left chopstick first, while the remaining R philosophers are right-handed and pick up their right chopstick first. Fill out the following table, indicating *yes*, *no*, or *depends* (describe scenario briefly in the last case).

L	R	Deadlock Possible?
5	0	
4	1	
3	2	

c) [3] The five philosophers instead decide to place N chopsticks in a heap in the middle of the table. Philosophers can now pick up one chopstick at a time, but still need two to eat. Say if deadlock is possible (YES, NO, DEPENDS) in each of the following cases:

N	Deadlock Possible?
<5	
5	
>5	

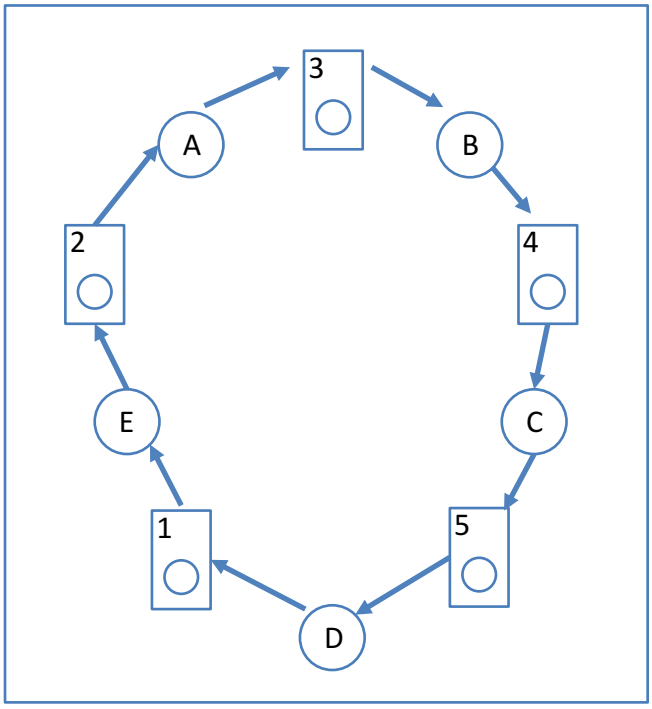
d) [5] In a recent NASA discovery, planet Mars has three-handed philosophers. Consider a table with five three-handed philosophers and a pile of N chopsticks in the middle of the table. Each philosopher needs 3 chopsticks to eat. What is the smallest N such that deadlock is impossible?

[15pts] 6. Variations on Dining Philosophers

Recall the five Dining Philosophers (and five chopsticks) sitting around a circular table: if they all pick up their right chopstick before their left, they can end up in a deadlocked situation.

a) [4] In the box to the right, draw a Resource Allocation Graph showing the Philosophers and chopsticks and a deadlocked situation.

b) [3] Assume L philosophers are left-handed and pick up their left chopstick first, while the remaining R philosophers are right-handed and pick up their right chopstick first. Fill out the following table, indicating *yes*, *no*, or *depends* (describe scenario briefly in the last case).



L	R	Deadlock Possible?
5	0	YES (each philosopher picks up left chopstick first)
4	1	NO (symmetry broken)
3	2	NO (ditto)

c) [3] The five philosophers instead decide to place N chopsticks in a heap in the middle of the table. Philosophers can now pick up one chopstick at a time, but still need two to eat. Say if deadlock is possible (YES, NO, DEPENDS) in each of the following cases:

N	Deadlock Possible?
<5	YES (each philosopher picks up 0 or 1 chopstick until nothing is left)
5	YES (each philosopher picks up 1 chopstick until nothing is left)
>5	NO (always at least one philosopher who can eat)

d) [5] In a recent NASA discovery, planet Mars has three-handed philosophers. Consider a table with five three-handed philosophers and a pile of N chopsticks in the middle of the table. Each philosopher needs 3 chopsticks to eat. What is the smallest N such that deadlock is impossible?

11

With 10 chopsticks, each philosopher could pick up two chopsticks and end up deadlocked. With 11, there is always at least one philosopher who can eat.

[10pts] 6. A New Job at Knab Bank

You have just started working for the Knab Bank to maintain their core code. A now retired programmer has written some highly concurrent code that allows many operations on bank accounts to go on concurrently. The programmer took great care to make sure that it would never be possible to ``see inconsistent state'', for example, halfway through a transfer from one account to another when money has been withdrawn from the one account but not yet deposited into the other. All that seems to work great. Unfortunately, some operations like deposit and transfer sometimes seem to hang for ever. Your job is to find the bug and fix it. Below is an excerpt of the code:

```
class Account:                      # account object
    def __init__(self):
        self.lock = Lock()          # lock on the account
        self.balance = 0            # amount of money in the account

class Bank:
    def __init__(self):             # initialize instance variables
        self.lock = Lock()          # lock on the list of accounts
        self.accounts = []          # append-only list of accounts

    def newAccount(self):           # create an account and return new account number
        with self.lock:
            acct_number = len(self.accounts)
            self.accounts.append(Account())
            return acct_number

    def deposit(self, acct_number, amount):      # add money to account
        acct = self.accounts[acct_number]
        with acct.lock:
            acct.balance += amount

    # transfer money from one account to another. Return whether successful or not
    def transfer(self, acct_number_from, acct_number_to, amount):
        acct_from = self.accounts[acct_number_from]
        acct_to = self.accounts[acct_number_to]
        with acct_from.lock:
            with acct_to.lock:
                if acct_from.balance < amount:          # insufficient funds
                    sufficient_balance = False
                else:                                   # update both accounts
                    sufficient_balance = True
                    acct_from.balance -= amount
                    acct_to.balance += amount
        return sufficient_balance                       # return success status
```

Briefly describe the bug(s) and how to fix it (them) in the box below. Use plain English, not code.

[10pts] 6. A New Job at Knab Bank

You have just started working for the Knab Bank to maintain their core code. A now retired programmer has written some highly concurrent code that allows many operations on bank accounts to go on concurrently. The programmer took great care to make sure that it would never be possible to ``see inconsistent state'', for example, halfway through a transfer from one account to another when money has been withdrawn from the one account but not yet deposited into the other. All that seems to work great. Unfortunately, some operations like deposit and transfer sometimes seem to hang for ever. Your job is to find the bug and fix it. Below is an excerpt of the code:

```
class Account:                      # account object
    def __init__(self):
        self.lock = Lock()          # lock on the account
        self.balance = 0            # amount of money in the account

class Bank:
    def __init__(self):             # initialize instance variables
        self.lock = Lock()          # lock on the list of accounts
        self.accounts = []          # append-only list of accounts

    def newAccount(self):           # create an account and return new account number
        with self.lock:
            acct_number = len(self.accounts)
            self.accounts.append(Account())
            return acct_number

    def deposit(self, acct_number, amount):      # add money to account
        acct = self.accounts[acct_number]
        with acct.lock:
            acct.balance += amount

    # transfer money from one account to another. Return whether successful or not
    def transfer(self, acct_number_from, acct_number_to, amount):
        acct_from = self.accounts[acct_number_from]
        acct_to = self.accounts[acct_number_to]
        with acct_from.lock:
            with acct_to.lock:
                if acct_from.balance < amount:          # insufficient funds
                    sufficient_balance = False
                else:                                   # update both accounts
                    sufficient_balance = True
                    acct_from.balance -= amount
                    acct_to.balance += amount
        return sufficient_balance                       # return success status
```

Briefly describe the bug(s) and how to fix it (them) in the box below. Use plain English, not code.

The two locks in the transfer() operation should be acquired in, say, order of account number, or deadlock occurs when cycles of waiting for locks form.