lab3sol

EECS 70

September 22, 2014

1 Virtual Lab 3 Solution: The Propose & Reject Algorithm

EECS 70: Discrete Mathematics and Probability Theory, Fall 2014

Due Date: Monday, September 22nd, 2014 at 12pm

Instructions:

- Complete this lab by filling in all of the required functions, marked with `"YOUR CODE HERE"`
- If you plan to use Python, make sure to go over **Tutorial 1A: Introduction to Python and IPython** before attempting the lab
- Make sure you run every code cell one after another, i.e. don't skip any cell. A shortcut for doing this in the notebook is Shift+Enter. When you finish, choose 'Cell > Run All' to test your code one last time all at once.
- **For the sake of simplicity, assume that men and women are both represented by numerals. Whenever you see "A", translate it to 1 and so on.**
- Please do not hardcode the result or change any function without the `"YOUR CODE HERE"` mark
- Questions? Bring them to our Office Hour and/or ask on Piazza
- Good luck, and have fun!

1.1 Table of Contents

The number inside parentheses is the number of functions you are required to fill out for this lab. Always make sure to double check before you submit.

- Question 1: Create lists of men and women (2)
- Question 2: What if women make the first move? (1)
- Question 3: Same final pairings (1)
- Question 3: How long does the algorithm take? (1)

Below you will find the implementation of a Person class in Python. Since we do not expect you to use Object-oriented Programming (OOP) concepts in EECS 70, don't worry if you have trouble understand every detail of the code.

That said, you are still expected to read through the code & the docstring of each function and have a general idea of what each function does. Understanding abstraction is at the heart of Computer Science.

In [48]:
```
class Person:
    """
    A Person represents one of the participants of the propose & reject algorithm
    Instances of Person must be stored in lists called 'men' and 'women'
    """
```

```
    def __init__(self, gender, num, ranks):
        """
        Initializes the person to either a man ('M') or woman ('W'),
        and gives him/her an identifying number
        and a preference list indicating how they rank the members of the opposite gen
        """

        if gender not in ['M', 'W']:
            raise Exception("Not male or female")
        self.gender = gender
        self.num = num
        self.ranks = ranks
        self.partner = None

    def get_highest_ranked_person(self):
        """
        Returns the most desirable of this Person's possible matches
        """

        if self.gender == 'M':
            return women[self.ranks[0]-1]
        else:
            return men[self.ranks[0]-1]

    def get_rank(self, other_person):
        """
        Returns the position of "other_person" in this Person's ranking list
        """

        if self.gender == other_person.gender:
            raise Exception("Two people of the same gender don't have a" +
                            " ranking for each other")
        return self.ranks.index(other_person.num) + 1

    def get_partner(self):
        """
        Returns the current partner if there is any, or None otherwise
        """

        return self.partner

    def maybe(self, other_person):
        """
        Changes this Person and other_person's partners to be each other
        """

        self.partner = other_person
        other_person.partner = self

    def remove_from_list(self, other_person):
        """
        Removes other_person from this Person's possible matches
        """

        self.ranks.remove(other_person.num)

    def reset(self):
        """
        Resets the partner of the current Person to None
        """

        self.partner = None

    def __str__(self):
        """
        Prints a Person object
```

```
        """

        return ",".join([self.gender, str(self.num), str(self.ranks),
                         str(None if not self.partner else self.partner.num)])
```

Question 1: Create lists of men and womenIn Python, you can create an instance of a class using the syntax `ClassName(class_params)`.

For example, in a 2x2 instance, if we want to create man 1 who prefers woman 2 to woman 1, we could write:

`man1 = Person('M', 1, [2, 1])`Here's an example of how to create lists of men and women using the example from Note 3, page 1.

In [49]:
```
from IPython.display import Image
Image(url='http://inst.eecs.berkeley.edu/~cs199-xs/stable.png')
```

Out [49]:
```
<IPython.core.display.Image at 0xa4d34cc>
```

In [50]:
```
def create_example_men_list():
    """
    This function should return a list of size 3, where each
    element in the list is a man from the above example
    """

    return [Person('M', 1, [1,2,3]), Person('M', 2, [2,1,3]), Person('M', 3, [1,2,3])]
```

In [51]:
```
def create_example_women_list():
    """
    This function should return a list of size 3, where each
    element in the list is a woman from the above example
    """

    return [Person('W', 1, [2,1,3]), Person('W', 2, [1,2,3]), Person('W', 3, [1,2,3])]
```

Now it's your turn. Using the example from Note 4, page 6, which is shown below again for your convenience, create a list of all the men and another list containing all the women.

Reminder: we are using numerals to represent both genders.

In [52]:
```
from IPython.display import Image
Image(url='http://inst.eecs.berkeley.edu/~cs199-xs/stable2.png')
```

Out [52]:
```
<IPython.core.display.Image at 0xa4d368c>
```

In [53]:
```
def create_men_list():
    """
    This function should return a list of size 4, where each
    element in the list is a man from the above example

    YOUR CODE HERE
    """

    return [Person('M', 1, [1,2,3,4]), Person('M', 2, [1,4,3,2]),
            Person('M', 3, [1,3,2,4]), Person('M', 4, [1,2,3,4])]
```

In [54]:
```
def create_women_list():
    """
    This function should return a list of size 4, where each
    element in the list is a woman from the above example

    YOUR CODE HERE
    """

    return [Person('W', 1, [1,3,2,4]), Person('W', 2, [4,3,2,1]),
            Person('W', 3, [2,3,1,4]), Person('W', 4, [3,4,2,1])]
```

In [55]:
```
def check_q1():
    """
    This function checks if the functions you created above
    returns two lists of the form we expect, so that you can
    move on to the next question. It does not check if the
    preference list for each Person is correct, so be careful!
    """

    men = create_men_list()
    women = create_women_list()
    if len(men) == 4 and len(women) == 4 and\
       all(isinstance(m, Person) for m in men) and\
       all(isinstance(w, Person) for w in women):
        print "Your lists have the right syntax."
    else:
        print "Your lists are incorrect. Please doublecheck."
```

In [56]:
```
check_q1()
```

```
Your lists have the right syntax.
```

Question 2: What if women make the first move?Below you will find a simple implementation of the traditional Propose & Reject algorithm, where the men propose first. The final pairing is male-optimal, as we learn in lecture.

We'll use the example on page 1 of Note 4, in case you didn't complete Question 1. That said, you should be able to change it to any example you like, including the example on page 6 of Note 4 that you implemented above.

In [57]:
```
# Gets the lists of men and women
men = create_example_men_list()
women = create_example_women_list()
```

In [58]:
```
def men_propose():
    """
    Runs the traditional Propose & Reject algorithm where men propose
    Returns the number of days the algorithm takes to complete
    """

    num_days = 0
    has_rejection = True
    while has_rejection:
        has_rejection = False
        proposals = {}

        # initialization
        for woman in women:
            proposals[woman] = []

        # propose phase
        for man in men:
            ideal_woman = man.get_highest_ranked_person()
            proposals[ideal_woman].append(man)

        # decision phase
        for woman in women:
            if len(proposals[woman]) == 0:
                continue
            best = proposals[woman][0]
            for proposer in proposals[woman]:
                if woman.get_rank(proposer) > woman.get_rank(best):
                    proposer.remove_from_list(woman) # reject
                    has_rejection = True
                if woman.get_rank(proposer) < woman.get_rank(best):
                    best.remove_from_list(woman) # reject
                    has_rejection = True
                    best = proposer
            woman.maybe(best)
        num_days += 1
    return num_days
```

In [59]:
```
def print_men_pairing():
    """
    Prints out the final pairings where men make the proposal
    """

    print "Final pairings:"
    print "(Man, Woman)"
    for man in men:
        print "(%s, %s)" % (man.num, man.get_partner().num)
```

In [60]:
```
def get_men_pairing():
    """
    Retrieves the final pairings where men make the proposal
    """

    pairings = []
    for man in men:
        pairings.append((man.num, man.get_partner().num))
    return pairings
```

In [61]:
```
men_propose()
men_pairing = get_men_pairing()
print_men_pairing()
```

```
Final pairings:
(Man, Woman)
(1, 1)
(2, 2)
(3, 3)
```

It's your turn. Use the code above and make any necessary changes, create a new version to run for the case where women propose first. Please do not hardcode the result, as we'll run your implementation on other instances.

Hint: A good place to start would be to copy the entire body of `men_propose()` function and reverse the role of men and women. Make sure you understand each part of the algorithm instead of blindly changing the variables' names, however.

In [62]:
```
# Gets the lists of men and women
men = create_example_men_list()
women = create_example_women_list()
```

In [63]:
```
def women_propose():
    """
    Runs the flipped Propose & Reject algorithm where women propose
    Returns the number of days the algorithm takes to complete

    YOUR CODE HERE
    """

    num_days = 0
    has_rejection = True
    while has_rejection:
        has_rejection = False
        proposals = {}

        # initialization
        for man in men:
            proposals[man] = []

        # propose phase
        for woman in women:
            ideal_man = woman.get_highest_ranked_person()
            proposals[ideal_man].append(woman)

        # decision phase
        for man in men:
            if len(proposals[man]) == 0:
```

```
                continue
            best = proposals[man][0]
            for proposer in proposals[man]:
                if man.get_rank(proposer) > man.get_rank(best):
                    proposer.remove_from_list(man) # reject
                    has_rejection = True
                if man.get_rank(proposer) < man.get_rank(best):
                    best.remove_from_list(man) # reject
                    has_rejection = True
                    best = proposer
            man.maybe(best)
        num_days += 1
    return num_days
```

In [64]:
```
def print_women_pairing():
    """
    Prints out the final pairing where women make the proposal
    """

    print "Final pairings:"
    print "(Man, Woman)"
    for woman in women:
        print "(%s, %s)" % (woman.get_partner().num, woman.num)
```

In [65]:
```
def get_women_pairing():
    """
    Retrieves the final pairing where women make the proposal
    """

    pairings = []
    for woman in women:
        pairings.append((woman.get_partner().num, woman.num))
    return sorted(pairings)  # sort by man's ID
```

Test your implementation below.

In [66]:
```
women_propose()
women_pairing = get_women_pairing()
print_women_pairing()
if women_pairing == [(1, 2), (2, 1), (3, 3)]:
    print "\nCorrect implementation, nice job!"
else:
    print "\nSomething is wrong with your implementation."
```

```
Final pairings:
(Man, Woman)
(2, 1)
(1, 2)
(3, 3)

Correct implementation, nice job!
```

Question 3: Same final pairingsThis question will focus on exploring what happens when we randomize preferences. Below, you will find an implementation of the function `create_random_lists`, which allows you to create a random set of preferences. This function will be used to explore some questions about a typical case of the propose-and-reject algorithm.

In [67]:
```
def create_random_lists(n=3):
    """
    Create random lists of men and women with random preferences

    n -- the number of men/women, default to 3
    """

    import random
```

```
    people = list(range(1, n+1))
    men = [Person('M', i, random.sample(people, n)) for i in range(1, n+1)]
    women = [Person('W', i, random.sample(people, n)) for i in range(1, n+1)]
    return men, women
```

If you run the cell below many times (repeated Ctrl+Enter), you will see that it prints different preference lists each time!

In [68]:
```
men, women = create_random_lists(4)
print ' | '.join(str(m) for m in men)
print ' | '.join(str(w) for w in women)
```

```
M,1,[1, 4, 2, 3],None | M,2,[1, 2, 3, 4],None | M,3,[1, 3, 4, 2],None
| M,4,[2, 3, 4, 1],None
W,1,[2, 1, 3, 4],None | W,2,[3, 4, 2, 1],None | W,3,[4, 2, 3, 1],None
| W,4,[4, 2, 1, 3],None
```

We're interested in how often the men-propose and women-propose algorithms return the same pairing. Complete the following function that generates a random list of preferences for men and women, then runs each variant of the propose-and-reject algorithm on that set of preferences. Make sure to use the same set of preferences for each iteration of the algorithm.

Follow the comments to carry out what you need to do for this question.

In [69]:
```
def same_final_pairing(n=100, num_people=4):
    """
    Runs the propose-and-reject on random preference lists n times,
    and report the number of times men-propose and women-propose
    give the same final pairing

    n -- number of iterations
    num_people -- how large our instance is

    Please note that there are four (4) locations where you have to fill in
    to make the function complete. At each location, you need at most two
    lines of code.
    """

    # Using global variables is highly discouraged in Python
    # and programming in general, but for syntax simplicity,
    # we'll be using it in this lab
    global men, women
    count = 0

    # Iterate n times
    for _ in range(n):
        # Generates random list of preferences
        # YOUR CODE HERE
        men, women = create_random_lists(num_people)

        # Runs the men-propose algorithm
        # YOUR CODE HERE
        men_propose()
        men_pairing = get_men_pairing()

        # Resets the algorithm by removing the current pairings
        # Do NOT edit the four lines below, or your implementation will be buggy
        for man in men:
            man.reset()
        for woman in women:
            woman.reset()

        # Runs the women-propose algorithm
        # YOUR CODE HERE
        women_propose()
```

```
        women_pairing = get_women_pairing()

        # Compares the final pairings, increments the counter
        # if the pairings are the same
        # YOUR CODE HERE
        if men_pairing == women_pairing:
            count += 1
    return count
```

Test your implementation below. How often is the male-optimal stable pairing the same as the female-optimal stable pairing? Does that frequency change as we change the value of `n`?

Note that there is no one correct answer here - however, you should be sure to back up your statements with numbers (for instance, how many times you've seen each at some value of `n`).

In [70]:
```
same_final_pairing()
```

Out [70]:
```
56
```

Question 4: How many days does the algorithm take?Finally, we're going to explore how long the men-propose algorithm takes as a function of `n`. Note that the function returns how many days it takes, so we can get the number of days taken with `num_days = men_propose()`.Write a function that returns how many days it takes for the Propose and Reject algorithm, where women make the proposal, to arrive at its stable solution with a random preference list.

In [72]:
```
def how_long(num_people=10):
    """
    Returns how many days the female-optimal algorithm takes
    to produce the final pairings for randomly-generated preference lists
    """

    global men, women

    # YOUR CODE HERE
    men, women = create_random_lists(num_people)
    return women_propose()
```

In [73]:
```
how_long(200)
```

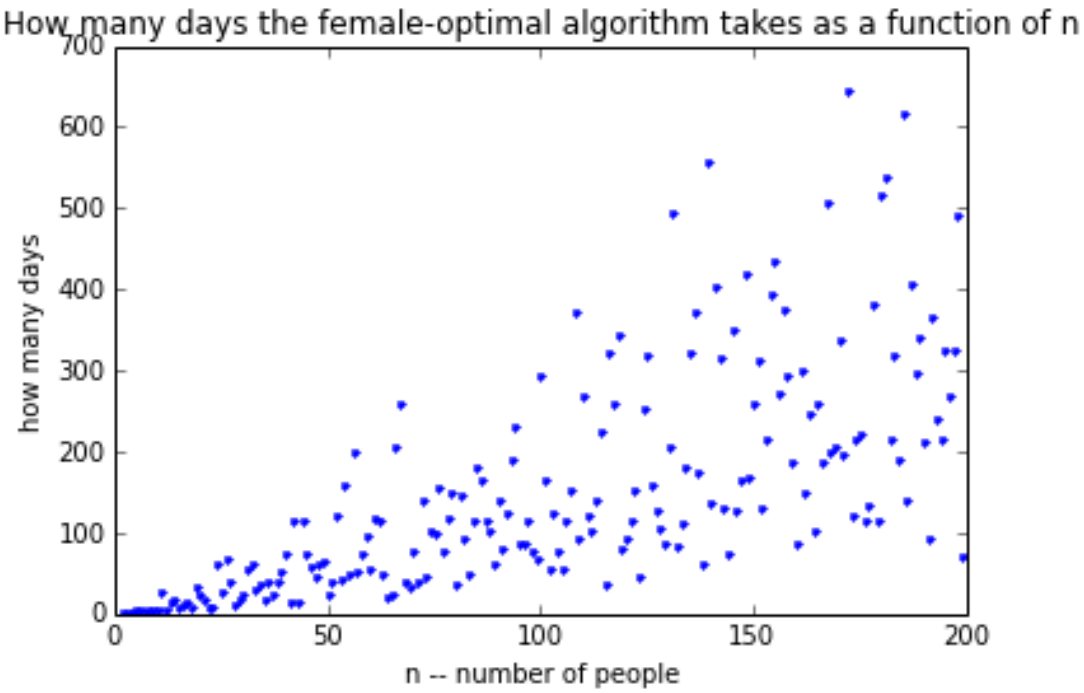
Out [73]:
```
246
```

In lecture, you learned that the algorithm must terminate after at most n^2 days (you will soon prove a stricter bound in another problem in this homework).

Try it on some inputs of different size. How quickly does the number of days grow with the input? Does it grow linearly (at the same rate as n - maybe twice as fast or half as fast), quadratically, logarithmically? If you can't tell, try graphing some outputs by hand. You do not have to submit any graph, but it certainly would help defend your claim.

In [27]:
```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

plt.plot(xrange(2, 200), map(how_long, xrange(2, 200)), '.')
plt.xlabel('n -- number of people')
plt.ylabel('how many days')
plt.title("How many days the female-optimal algorithm takes as a function of n")
plt.show()
```



Congratulations! You are done with Virtual Lab 3.

Don't forget to convert this notebook to a pdf document, merge it with your written homework, and submit both the pdf and the code (as a zip file) on glookup.

Reminder: late submissions are NOT accepted. If you have any technical difficulty, resolve it early on or use the provided VM.**Acknowledgment**: The Person class was originally written by Mike Ambrose, one of our readers this semester.