CSCI 3210: Computational Game Theory
Summative Assignment 2: Linear Programming (LP)
Points: 100
Individual assignment: Collaboration Level 2 (discussion with professor only)
https://turing.bowdoin.edu/dept/collab.php

Why?

The importance of LP goes beyond this course. There's a field called Operations Research (OR) that studies a range of real-world optimization problems from supply chain to transportation to airline crew scheduling to Uber/Lyft driver matching. One of the founding pillars of Operations Research is LP. Interestingly, any LP is equivalent to a two-player zero-sum game, connecting LP to game theory.

How will it contribute to my CS education?

Unfortunately, CS is often mistaken as just coding. We forget that the second word in CS is science, meaning it's as rigorous as any other scientific discipline. We cannot be computer scientists simply by coding some detailed solution handed to us. We not only need to exercise scientific rigors in creating solutions but also need to be aware of the engineering aspects of implementing them. LP with its theoretical depth (e.g., LP duality) is a prime example of scientific rigors in CS. LP with its implementation aspects is a great example of engineering in CS. In other words, LP has both!

Background: PuLP– Linear Programming in Python

Installation
Follow the instructions on https://coin-or.github.io/pulp/main/includeme.html#installation.

Tutorials:
Video: https://youtu.be/qa4trkLfvwQ
Text: https://realpython.com/linear-programming-python/#using-pulp

Documentation:
https://www.coin-or.org/PuLP/pulp.html

Submission instruction

Submit separate .py source files (q1.py, q2.py, and q3.py) on Canvas. Use Python version 3.

Q1 (Warm-up): Bakery Problem

1. Download the code for the bakery problem from the description section of the PuLP video lecture: https://youtu.be/qa4trkLfvwQ. Run the code.
2. Program the dual of the bakery problem in the same file. Show that the primal and dual have the same optimal value by printing their optimal objective function values. Do not print anything else.

There is no input file for Q1. Name your source file **q1.py**. Make sure you include the following lines so that your code can be run from a terminal.

```
import sys
def main():
    # Entry point of your program
if __name__ == "__main__":
    main()
```

Q2: Write a program in Python version 3 to solve the following problem. Use PuLP as the LP solver.

You are helping Uber with their optimization problem of dispatching drivers to customers while minimizing the total pick-up time. Suppose we have a uniform grid of locations. At any point in time, all drivers and customers are located at grid points. The time for a driver to pick up a customer is proportional to the total of their x- and y-axis distances (which is known as the Manhattan distance between the two corresponding grid points).

(a) Given n > 0 drivers, m > 0 customers and their locations on the grid as (x,y) coordinates, formulate an LP for the problem of matching drivers to customers so that the total pick-up time is minimized for the given snapshot of driver and customer locations. That is, we are not assigning a driver to multiple customers one after another over time. We are simply matching drivers to customers at a snapshot of time. As a result, some drivers or some customers may not get matched at all.

Think why the greedy approach of assigning the drivers to their nearest customers may not work, necessitating a more sophisticated solution like LP.

Write your LP formulation as a multiline comment at the top of your source code.

(b) Implement the LP. Take a .txt file as the input that has the following format: In the first line, we have the value of n (# of drivers); in the next line, the value of m (# of customers); in each of the subsequent n lines, (x,y) coordinates for driver locations (without the parentheses and with the comma replaced by space); in each of the subsequent m lines, similarly specified (x,y) coordinates for customer locations.

Sample input file (10x10 grid, 1 driver, 2 customers)

```
10
1
2
0 2
5 3
8 7
```

Your program should only output (1) the optimal matching of drivers to customers, where driver and customer IDs start with 0 and (2) the total pick-up time. Also, as stated above, write your LP formulation as a comment at the top of your source code.

Name your source file **q2.py**. Include the following lines so that your code (1) can be run from a terminal and (2) can take the input file as a command line argument.

```
import sys
def main(input_file_name):
    # Entry point of your program
if __name__ == "__main__":
    main(sys.argv[1])
```

Your code will be evaluated by the following command in a terminal (file names are arbitrary).

```
$ python3  q2.py  anyInputFileNameWeChoose.txt
```

Q3: Write a program in Python 3 to solve any 2-player zero-sum game by implementing both the primal and dual linear programs. Use PuLP as the LP solver.

Your program should take input from a text file as a command line argument. The text file describes a 2-player zero-sum game as follows. The first line specifies the number of actions (>=2) of the row player. The arbitrary number of actions may require special attention to data structures. The second line specifies the number of actions (>= 2) of the column player. The rest of the lines specify the payoff matrix of the row player. As usual, the column player's payoff matrix is the negative of the row player's payoff matrix and is *not* specified.

Sample input file (penalty kick game)

```
2
2
-1  1
1  -1
```

What to implement

Solve both the primal and dual linear programs (see slides or [AGT] Ch 1). The primal LP will produce how much the row player gets as well as the row player's probability distribution. The dual LP will give similar information about the column player.

Required output

In the first line, print how much the row player gets at the NE, and in the second line, print how much the column player pays at the NE. In the subsequent two lines, print the row and column players' probability distributions, respectively. Do not print anything else.

Name your source file **q3.py**. Include the following lines so that your code (1) can be run from a terminal and (2) can take the input file as a command line argument.

```
import sys
def main(input_file_name):
    # Entry point of your program
if __name__ == "__main__":
    main(sys.argv[1])
```

Your code will be evaluated by the following command in a terminal (file names are arbitrary).

```
$ python3  q3.py  anyInputFileNameWeChoose.txt
```