Instructions: Try to solve all problems on your own. If you have difficulties, ask the instructor or TAs.

Submission: All lab files including the report (if not paper based) should be kept in a folder named "lab_xx_yyyyyyyy", where "xx" is the lab number (02 for this lab), and "yyyyyyyy" is your roll number. It should be zipped to a file called "lab_xx_yyyyyyyy.zip" and uploaded to Moodle.

Report: Your report should explain clearly what you have done and why. It should give detailed answers to any questions asked.

To solve the problem of the previous lab, we will solve linear programs repeatedly in Python, for which PuLP is a useful tool. PuLP is a Python-based modeling tool for writing (linear and integer-linear) optimization problems (like AMPL). Since it is based on Python, you may embed your optimization problem in any program. This feature enables us to develop more powerful algorithms. We will first obtain and set up PuLP in this lab, and then solve some problems. It will be used for solving the cutting stock problem discussed in the previous lab.

Exercise 0: Installing PuLP
PuLP is an open-source software. You are free to download, install, modify and share it. Instructions to download and install PuLP are available at https://pypi.python.org/pypi/PuLP. You may follow these instructions to install it on your laptop.

PuLP is also available in IEOR Computational Servers. Log into any one server and type

```
source /home/amahajan/IE684/setup_pulp.sh
```

To test whether PuLP is available for you, just type

```
python
>>> import pulp
>>> pulp.pulpTestAll()
```

Note: You should see the message "Solver pulp.solvers.PULP_CBC_CMD passed" in the output. You can safely ignore the message 'pulp.constants.PulpError: Tests Failed' seen on the servers.

Exercise 1: Using PuLP
Consider the linear integer program from the previous lab.

$$\begin{aligned} \min & \sum_{i=1}^{n} c_i x_i \\ \texttt{s.t.} & \sum_{i=1}^{n} w_i x_i \geq b, \\ & x_i \in \{0,1\}, \quad i = 1, \ldots, n, \end{aligned}$$

where $n \in \mathbb{Z}_+, c_i \in \mathbb{Z}_+, w_i \in \mathbb{Z}_+, i = 1, \ldots, n$ are given.

1. Solve the problem using PuLP and two input data available on Moodle ('input1.py' and 'input2.py').
2. Consider the input 'input1.py'. Plot the optimal solution value as a function of the right hand side b. Consider at least 30 different evenly spaced out values.

Exercise 2: Back to Cutting Stock
Let us now solve the LP of the cutting stock problem of the previous lab. Instead of using the nonconvex nonlinear

model from the previous lab, one can use a (somewhat big) integer linear program. Suppose we have already generated all the maximal patterns as in the last lab, and the decision variables are only how many of each pattern to use. Write an ILP formulation (only the abstract model) using those patterns as an input. How many variables and constraints are there in your model for the two data sets?

When the demands are relatively high (as in the data used in the previous lab), then one can ignore the integer restriction on the number of times a pattern is used and compensate for the approximation by rounding up the number to be used. It will ensure that all demand is met with a small relative change in the objective value. Thus we will now onwards only consider the continuous linear program.

Let us assume the LP has N variables. At the optimal basic solution of the linear program, how many variables can be nonzero? Suppose we start with a small LP and consider only a few patterns, say k, and solve the LP with these k patterns only. Will this give us a feasible solution always? Will the solution be a basic solution of the big LP? Explain.

Now suppose we have solved a small LP with k patterns, and we have a feasible solution. We know it is optimal if the reduced costs of all the N variables at this basic solution are non-negative. How can we check this condition without looking at each pattern or variable? It can be done by solving the following small optimization problem:

Find a valid pattern with the lowest reduced cost at this basic solution. Write this problem as a knapsack problem (an integer program with one constraint and one objective function). If the optimal objective value is non-negative, then we can stop. Explain why.

If the optimal objective is negative, we can add one variable to our small LP and re-solve the LP to obtain a better solution. Implement this scheme and obtain the optimal solution of the problems given in the previous lab.

Report the optimal solutions. Are they non-integers? If so, then how much relative deviation do you see when you round up the solution?