Homework #2

Due July 5 @ 11:59pm

Submission requirements

Upload a **single PDF or HTML file** of your IJulia notebook for this entire assigment. Clearly denote which question each section of your file corresponds to.

Problem 1 -- On your mark, get set, bake!

A bakery mass produces 4 different types of cake: carrot, chocolate, vanilla, and angel-food. The bakery measures cake characteristics using two different features: sweetness (S) and density (D). The following table summarizes the characteristics of each type of cake as well as the (max) number of each type that are baked each month:

Cake	S	D	Max baked
Carrot	100	15	3810
Chocolate	90	18	2700
Vanilla	86	6	4000
Angel food	110	3	1500

The cakes can either be sold directly from the bakery at $14 a cake or sold at a local market at a higher rate. There are two markets -- A and B -- that have requirements for average sweetness and density of cakes sold there. No more than 7 cakes can be sold at each market. The table below summarizes these requirements and gives the selling price per cake at each market:

Market	S	D	Max cakes	Price per cake (dollars)
A	at least 98	at most 10	7	20
B	at least 100	at most 8	7	30

The S and D values for each market are calculated as the weighted averages of the S and D of the cakes sold at the market.

(a) Formulate a linear program that will yield the largest possible revenue for the bakery. Assume it is okay to sell fractional cakes, since we are planning in aggregate.

(b) Build and solve the problem in Julia. What are the optimal numbers of each type of cake that the bakery should sell directly and at each market?

(a)

Solution

Let C be the set of cake types and P be the set of party platter types.

Decision variables

- x_i: number of each cake of type $i \in C$ that are sold as-is
- y_{ij}: number of each cake of type $i \in C$ that is used in party platter type $j \in P$

Objective

$$\max \;\; 14\sum_{i\in C}x_i + 20\sum_{i\in C}y_{iA} + 30\sum_{i\in C}y_{iB}$$

Constraints

- Do not exceed total number of each cake type produced

$$x_{car} + y_{car,A} + y_{car,B} \leq 3810$$

$$x_{choc} + y_{choc,A} + y_{choc,B} \leq 2700$$

$$x_{van} + y_{van,A} + y_{van,B} \leq 4000$$

$$x_{ang} + y_{ang,A} + y_{ang,B} \leq 1500$$

- Max number of cakes on each party platter

$$\sum_{i\in C}y_{iA} \leq 7; \; \sum_{i\in C}y_{iB} \leq 7$$

- Minimum sweetness on party platters

$$100y_{car,A} + 90y_{choc,A} + 86y_{van,A} + 110y_{ang,A} \geq 98\sum_{i\in C}y_{iA}$$

$$100y_{car,B} + 90y_{choc,B} + 86y_{van,B} + 110y_{ang,B} \geq 100\sum_{i\in C}y_{iA}$$

- Maximum density on party platters

$$15y_{car,A} + 18y_{choc,A} + 6y_{van,A} + 3y_{ang,A} \leq 10\sum_{i\in C}y_{iA}$$

$$15y_{car,B} + 18y_{choc,B} + 6y_{van,B} + 3y_{ang,B} \leq 8\sum_{i\in C}y_{iA}$$

In [1]:

```julia
# Solution 1B
using JuMP, Clp

# ~~ Create the Data ~~

# Set of possible cakes
cakes = [:carrot, :chocolate, :vanilla, :angel_food]

# Set of possible markets
markets = [:market_A, :market_B]

# Create a symbol for the direct from bakery sales price
bakery_price = 14

# Dictionary mapping the cake to its sweetness
s_c =  Dict(
  :carrot => 100,
  :chocolate => 90,
  :vanilla => 86,
  :angel_food => 110
)

# Dictionary mapping the cake to its density
d_c = Dict(
  :carrot => 15,
  :chocolate => 18,
  :vanilla => 6,
  :angel_food => 3
)

# Dictionary mapping the market to the max number of cakes that can be baked each month
max_baked = Dict(
    :carrot => 3810,
    :chocolate => 2700,
    :vanilla => 4000,
    :angel_food => 1500
)

# Dictionary mapping the market to the price of the cake
price_market_mapping = Dict(
    :market_A => 20,
    :market_B => 30
)

# Dictionary mapping the market to the max number of cakes that can be sold at that market
max_cakes = Dict(
    :market_A => 7,
    :market_B => 7
)

# Dictionary mapping the market to the weighted average sweetness requirement
sweet_req = Dict(
    :market_A => 98,
    :market_B => 100
)

# Dictionary mapping the market to the weighted average density requirement
dense_req = Dict(
    :market_A => 10,
    :market_B => 8
)
;
```

In [3]:

```julia
# ~~ Create the model ~~
m = Model(Clp.Optimizer)

# ~~ Create decision variables ~~

# 1. cakes_sold_bakery[cakes] is a variable for the number of cakes sold from the bakery each month for each ca
# 2. cakes_sold_market[markets,cakes] is a variable for the number of cakes sold at each market each month for

@variable(m, cakes_sold_bakery[cakes] >= 0)
@variable(m, cakes_sold_market[markets,cakes] >= 0)

# ~~ Create objective function ~~
# Objective Function: maximize the profit from selling the cakes

@objective(m, Max, sum(bakery_price*cakes_sold_bakery[cake] for cake in cakes) + sum(cakes_sold_market[market,c

# ~~ Create Constraints ~~
@constraint(m, constr_cake_max[cake in cakes], cakes_sold_bakery[cake] + sum(cakes_sold_market[market,cake] fo
@constraint(m, constr_market_max[market in markets], sum(cakes_sold_market[market,cake] for cake in cakes) <= m
@constraint(m, constr_sweet_req[market in markets], (sum(cakes_sold_market[market,cake] * s_c[cake] for cake i
@constraint(m, constr_density_req[market in markets], (sum(cakes_sold_market[market,cake] * d_c[cake] for cake

# ~~ Optimize the Model ~~
optimize!(m)

# ~~ Output Model Information ~~

# Print the objective value
println("Objective value: ",objective_value(m))
println("-------------------------------------------")

# Print the number of cakes sold from the bakery each month
println("# of Cakes sold directly from Bakery")
for cake in cakes
    println("->", cake,  ":",value(cakes_sold_bakery[cake]))
end
println("------------------------------------------")

println("# of Cakes sold in Markets (A or B)")
for market in markets
    println(market, ":")
    for cake in cakes
        println("->",cake, ":", value(cakes_sold_market[market,cake]))
    end
    print("\n")
end

print("\n")


# Print the number of cakes sold from the bakery each month next to their Name
# print(m)
```

```
Objective value: 168294.00000000003
-------------------------------------------
# of Cakes sold directly from Bakery
->carrot:3810.0000000000005
->chocolate:2693.4200000000005
->vanilla:3999.0666666666666
->angel_food:1493.5133333333333
------------------------------------------
# of Cakes sold in Markets (A or B)
market_A:
->carrot:0.0
->chocolate:3.080000000000001
->vanilla:0.9333333333333322
->angel_food:2.9866666666666672

market_B:
->carrot:0.0
->chocolate:3.500000000000001
->vanilla:0.0
->angel_food:3.499999999999999
```

```
Coin0506I Presolve 10 (0) rows, 12 (0) columns and 36 (0) elements
Clp0006I 0  Obj 0 Primal inf 31.431467 (4) Dual inf 193.3846 (12)
Clp0006I 12  Obj 168294
Clp0000I Optimal - objective value 168294
Clp0032I Optimal objective 168294 - 12 iterations time 0.002
```

Problem 2 -- Start your engines!

A company who works with car manufacturers produces and sells engines of various types and sizes. There is one particular style of engine that comes in three different sizes (call them "size 1," "size 2," and "size 3"). The contractor processes raw materials to produce the engines. Processing a pallet of raw materials costs $700 per pallet and produces 2 size 1 engines, 1 size 2 engine, and 1 size 3 engine. Size 1 engines sell for \$160 each, size 2 engines sell for $210 each, and size 3 engines sell for \$300 each. Engines (of any size) can be stored from month to month. Keeping engines in good working condition costs $5$$\times(*enginesize*)*perenginepermonth*(*i*. *e*. , $5*forsize*1,

10 for size 2, $15 for size 3).

The table below shows the maximum amount of each engine the contractor can sell in the next three months as well as the maximum number of pallets of raw materials it can process in each month. Demand that is not met in the month it occurs cannot be carried over to a later month (i.e., no backlogging is allowed -- so the demand amounts in the table represent the maximum that can be sell -- it may not be possible to sell this much). The contractor currently has 10 size 1 engines and 2 size 3 engines (no size 2 engines) in stock, and wishes to have at least that much -- plus at least one size 2 engine -- in stock at the end of month 3.

Month (t)	**Max raw materials (RM_t)**	**Max size 1 ($M1_t$)**	**Max size 2 ($M2_t$)**	**Max size 3 ($M3_t$)**
1	15	30	12	15
2	20	15	18	17
3	12	20	13	21

(a) Formulate a linear programming model to help the contractor maximize its profits over the next three months. Solve the model in Julia/JuMP and display the production plan (how many of each engine size to produce, store, and sell in each month).

(b) Now suppose the contractor would like to allow backlogging of demand. Demand not met in a month may be met in a future month (backlogged) or may be lost. The contractor pays a fee of $19 per engine per month backlogged, regardless of engine size. Modify your code from part (a) to include the ability to backlog. The same ending stock requirements should be enforced in part (b) and no engines should be backlogged in the final month. How, if at all, does this change the optimal objective value and solution? Speculate as to the reason for the change (or lack of change).

In [8]:

```julia
# Solution 2(a)
using JuMP, Clp, NamedArrays

engine_size = 1:3 # engine size options
month = 1:3 # months in planning horizon

rm_cost = 700 # cost of processing a pallet of raw materials
max_rm = Dict(zip(month,[15 20 12])) # max pallets of rm that can be processed each month

# create matrix of max demand for each month of each engine size
demand_matrix = [30 12 15
                 15 18 17
                 20 13 21]

# named array to store the demand data for each month
demand_NA = NamedArray(demand_matrix,(month,engine_size),("month","engine_size"))

rev = Dict(zip(engine_size, [160 210 300])) # revenue per engine

hold_cost = Dict(zip(engine_size,[5 10 15])) # cost to hold an engine in inventory

current_inv = Dict(zip(engine_size,[10 0 2])) # initial stock
end_inv = Dict(zip(engine_size,[10 1 2])); # desired stock at end

m = Model(Clp.Optimizer)

@variable(m, r[month] >= 0) # number of pallets of rm to process each month
@variable(m, x[month, engine_size] >= 0) # how many engines to produce each month
@variable(m, y[1:4, engine_size] >= 0) # how many engines to store each month
@variable(m, z[month, engine_size] >= 0) # hom many engines to sell each month

# objective is to maximize profit (revenue - production costs - holding costs)
@objective(m, Max, sum(rev[j]*z[i,j] for i in month for j in engine_size) -
    rm_cost*sum(r[i] for i in month) -
    sum(hold_cost[j]*y[i,j] for i in month for j in engine_size))

# inventory balance
@constraint(m, inv_bal1[i in engine_size], current_inv[i] + x[1, i] == z[1, i] + y[2, i])
@constraint(m, inv_bal2[i in engine_size], y[2,i] + x[2, i] == z[2, i] + y[3, i])
@constraint(m, inv_bal3[i in engine_size], y[3,i] + x[3, i] == z[3, i] + y[4, i])

# meet ending requriements
@constraint(m, end_stock[i in engine_size], y[4,i] >= end_inv[i])

# balance rm used
@constraint(m, bal_rm_size1[i in month], 2r[i] == x[i,1]) # get two size 1 engines for every pallet processed
@constraint(m, bal_rm_size2[i in month], r[i] == x[i,2]) # get one size 2 engine2 for every pallet processed
@constraint(m, bal_rm_size3[i in month], r[i] == x[i,3]) # get one size 3 engine for every pallet processed

# don't exceed demand
@constraint(m, dem[i in month, j in engine_size], z[i,j] <= demand_NA[i,j])

# don't exceed max rm
@constraint(m, max_rm_used[i in month], r[i] <= max_rm[i])

optimize!(m)

println("Production schedule: ", NamedArray(value.(x),
        (month, engine_size),("month", "engine_size")))
println("Storage plan: ", NamedArray(value.(y), (1:4, engine_size),("month","engine_size")))
println("Selling plan: ", NamedArray(value.(z), (month, engine_size),("month","engine_size")))
println("Total profit: ", objective_value(m))
```

```
Production schedule: 3×3 Named JuMP.Containers.DenseAxisArray{Float64, 2, Tuple{UnitRange{Int64}, UnitRange{Int
64}}, Tuple{JuMP.Containers._AxisLookup{Tuple{Int64, Int64}}, JuMP.Containers._AxisLookup{Tuple{Int64, Int6
4}}}}
month ╲ engine_size │    1     2     3
────────────────────┼─────────────────
1                   │ 20.0  10.0  10.0
2                   │ 21.0  10.5  10.5
3                   │ 24.0  12.0  12.0
Storage plan: 4×3 Named JuMP.Containers.DenseAxisArray{Float64, 2, Tuple{Base.OneTo{Int64}, UnitRange{Int64}},
Tuple{JuMP.Containers._AxisLookup{Base.OneTo{Int64}}, JuMP.Containers._AxisLookup{Tuple{Int64, Int64}}}}
month ╲ engine_size │    1     2     3
────────────────────┼─────────────────
1                   │  0.0   0.0   0.0
2                   │  0.0   0.0   0.0
3                   │  6.0   0.0   0.0
4                   │ 10.0   1.0   2.0
Selling plan: 3×3 Named JuMP.Containers.DenseAxisArray{Float64, 2, Tuple{UnitRange{Int64}, UnitRange{Int64}}, T
uple{JuMP.Containers._AxisLookup{Tuple{Int64, Int64}}, JuMP.Containers._AxisLookup{Tuple{Int64, Int64}}}}
month ╲ engine_size │    1     2     3
────────────────────┼─────────────────
1                   │ 30.0  10.0  12.0
2                   │ 15.0  10.5  10.5
3                   │ 20.0  11.0  10.0
Total profit: 3985.0
Coin0506I Presolve 9 (-24) rows, 18 (-15) columns and 30 (-36) elements
Clp0006I 0  Obj -0 Primal inf 24.999995 (5) Dual inf 2010 (9)
Clp0006I 15  Obj 3985
Clp0000I Optimal - objective value 3985
Coin0511I After Postsolve, objective 3985, infeasibilities - dual 610 (3), primal 0 (0)
Coin0512I Presolved model was optimal, full model needs cleaning up
Clp0006I 0  Obj 3985
Clp0000I Optimal - objective value 3985
Clp0032I Optimal objective 3985 - 15 iterations time 0.002, Presolve 0.00
```

In [10]:

```julia
using JuMP, Clp, NamedArrays

engine_size = 1:3 # engine size options
month = 1:3 # months in planning horizon

rm_cost = 700 # cost of processing a pallet of raw materials
max_rm = Dict(zip(month,[15 20 12])) # max pallets of rm that can be processed each month

# create matrix of max demand for each month of each engine size
demand_matrix = [30 12 15
                 15 18 17
                 20 13 21]

# named array to store the demand data for each month
demand_NA = NamedArray(demand_matrix,(month,engine_size),("month","engine_size"))

rev = Dict(zip(engine_size, [160 210 300])) # revenue per engine

hold_cost = Dict(zip(engine_size,[5 10 15])) # cost to hold an engine in inventory
short_cost = 19

current_inv = Dict(zip(engine_size,[10 0 2])) # initial stock
end_inv = Dict(zip(engine_size,[10 1 2])); # desired stock at end

m = Model(Clp.Optimizer)

@variable(m, r[month] >= 0) # number of pallets of rm to process each month
@variable(m, x[month, engine_size] >= 0) # how many engines to produce each month
@variable(m, y[1:4, engine_size]) # how many engines in inventory
@variable(m, b[1:4, engine_size] >= 0) # number of engines backlogged each month
@variable(m, s[1:4, engine_size] >= 0) # number of engines left over each month
@variable(m, z[month, engine_size] >= 0) # how many engines to sell each month

# objective is to maximize profit (revenue - production costs - shortage costs - holding costs)
@objective(m, Max, sum(rev[j]*z[i,j] for i in month for j in engine_size) -
    rm_cost*sum(r[i] for i in month) -
    short_cost*sum(b[i,j] for i in month for j in engine_size) -
    sum(hold_cost[j]*s[i,j] for i in month for j in engine_size))

# inventory balance
@constraint(m, inv_bal1[i in engine_size], current_inv[i] + x[1, i] == z[1, i] + y[2, i])
@constraint(m, inv_bal2[i in engine_size], y[2,i] + x[2, i] == z[2, i] + y[3, i])
@constraint(m, inv_bal3[i in engine_size], y[3,i] + x[3, i] == z[3, i] + y[4, i])
@constraint(m, inv_id[i in month, j in engine_size], s[i,j] - b[i,j] == y[i,j])

# meet ending requriements
@constraint(m, end_stock[i in engine_size], y[4,i] >= end_inv[i])

# balance rm used
@constraint(m, bal_rm_size1[i in month], 2r[i] == x[i,1]) # get two size 1 engines for every pallet processed
@constraint(m, bal_rm_size2[i in month], r[i] == x[i,2]) # get one size 2 engine2 for every pallet processed
@constraint(m, bal_rm_size3[i in month], r[i] == x[i,3]) # get one size 3 engine for every pallet processed

# don't exceed demand
@constraint(m, dem[i in month, j in engine_size], z[i,j] <= demand_NA[i,j])

# don't exceed max rm
@constraint(m, max_rm_used[i in month], r[i] <= max_rm[i])

optimize!(m)

println("Production schedule: ", NamedArray(value.(x),
        (month, engine_size),("month", "engine_size")))
println("Engines leftover: ", NamedArray(value.(s), (1:4, engine_size),("month","engine_size")))
println("Engines backlogged: ", NamedArray(value.(b), (1:4, engine_size),("month","engine_size")))
println("Selling plan: ", NamedArray(value.(z), (month, engine_size),("month","engine_size")))
println("Total profit: ", objective_value(m))
```

```
Production schedule: 3×3 Named JuMP.Containers.DenseAxisArray{Float64, 2, Tuple{UnitRange{Int64}, UnitRange{Int
64}}, Tuple{JuMP.Containers._AxisLookup{Tuple{Int64, Int64}}, JuMP.Containers._AxisLookup{Tuple{Int64, Int6
4}}}}
```

```
month ╲ engine_size │   1     2     3
────────────────────┼─────────────────
1                   │ 20.0  10.0  10.0
2                   │ 21.0  10.5  10.5
3                   │ 24.0  12.0  12.0
Engines leftover: 4×3 Named JuMP.Containers.DenseAxisArray{Float64, 2, Tuple{Base.OneTo{Int64}, UnitRange{Int6
4}}, Tuple{JuMP.Containers._AxisLookup{Base.OneTo{Int64}}, JuMP.Containers._AxisLookup{Tuple{Int64, Int64}}}}
month ╲ engine_size │  1    2    3
────────────────────┼──────────────
1                   │ 0.0  0.0  0.0
2                   │ 0.0  0.0  0.0
3                   │ 6.0  0.0  0.0
4                   │ 0.0  0.0  0.0
Engines backlogged: 4×3 Named JuMP.Containers.DenseAxisArray{Float64, 2, Tuple{Base.OneTo{Int64}, UnitRange{Int
64}}, Tuple{JuMP.Containers._AxisLookup{Base.OneTo{Int64}}, JuMP.Containers._AxisLookup{Tuple{Int64, Int64}}}}
month ╲ engine_size │  1    2    3
────────────────────┼──────────────
1                   │ 0.0  0.0  0.0
2                   │ 0.0  0.0  0.0
3                   │ 0.0  0.0  0.0
4                   │ 0.0  0.0  0.0
Selling plan: 3×3 Named JuMP.Containers.DenseAxisArray{Float64, 2, Tuple{UnitRange{Int64}, UnitRange{Int64}}, T
uple{JuMP.Containers._AxisLookup{Tuple{Int64, Int64}}, JuMP.Containers._AxisLookup{Tuple{Int64, Int64}}}}
month ╲ engine_size │   1     2     3
────────────────────┼─────────────────
1                   │ 30.0  10.0  12.0
2                   │ 15.0  10.5  10.5
3                   │ 20.0  11.0  10.0
Total profit: 3985.0
Coin0506I Presolve 9 (-33) rows, 24 (-33) columns and 42 (-51) elements
Clp0006I 0  Obj -0 Primal inf 36.999993 (7) Dual inf 2010 (9)
Clp0006I 14  Obj 3985
Clp0000I Optimal - objective value 3985
Coin0511I After Postsolve, objective 3985, infeasibilities - dual 610 (3), primal 0 (0)
Coin0512I Presolved model was optimal, full model needs cleaning up
Clp0006I 0  Obj 3985
Clp0000I Optimal - objective value 3985
Clp0032I Optimal objective 3985 - 14 iterations time 0.002, Presolve 0.00
```

Problem 3 -- Timetabling

The College of Engineering has 5 different levels of classes (e.g., intro level, PhD level, ...) it plans to offer next year in the Summer 2023 session. For simplicity, the class levels are labeled $A, B, C, D,$ and E. There are 5 different types of professors (based on area of expertise and type of professorship) who can teach classes next summer: $I1, I2, I3, I4,$ and $I5$. Each individual instructor can teach one course. Professors of type $I1$ can teach any of the 5 levels. Professors of type $I2$ can teach levels $A, C,$ and D. Professors of type $I3$ can teach levels $B, C,$ and E. Professors of type $I4$ can teach levels A and D. Finally, Professors of type $I5$ are highly specialized and can only teach level E.

The number of classes at each level that are scheduled to be taught next year are given in the table below:

Level	A	B	C	D	E
# needed	5	8	12	11	4

The cost of hiring a professor to teach a class and the available number of each type of professor are listed in the table below (the cost does not depend on the level of class taught - only on the type of professor). It is not required to use all professors -- these number available quantities represent the maximum of each type that can be used.

Prof. type (i)	$I1$	$I2$	$I3$	$I4$	$I5$
Cost of hiring	\$385	\$340	\$300	\$270	\$250
# available (M_i)	20	7	10	11	3

Formulate a linear program that will help determine how many professors of each type should be hired to teach the required courses next year at a minimum cost. Solve your model in Julia and report the solution (make sure you interpret the meaning of your solution). What type of MCNF problem is this?

In [13]:

```julia
# Solution 3 - This is an "Assignment Matching Problem"
using JuMP, Clp

nodes = [:A :B :C :D :E :I1 :I2 :I3 :I4 :I5] # levels of classes and types of profs are nodes
arcs = [(:A,:I1) (:A,:I2) (:A,:I4) (:B,:I1) (:B,:I3) (:C,:I1) (:C,:I2) (:C,:I3) (:D,:I1) (:D,:I2) (:D,:I4) (:E,

arc_cost = Dict(zip(arcs,[385 340 270 385 300 385 340 300 385 340 270 385 300 250])) # cost of hiring a profess
net_supply = Dict(zip(nodes,[5 8 12 11 4 -20 -7 -10 -11 -3])) # net supply at each node



m = Model(Clp.Optimizer)

@variable(m, x[arcs] >= 0)

@objective(m, Min, sum(arc_cost[a]*x[a] for a in arcs))

@constraint(m, class_n[i in nodes],
        sum(x[a] for a in arcs if a[1] == i) - sum(x[a] for a in arcs if a[2] == i) >= net_supply[i])

optimize!(m)

for j in nodes[6:10]
    for i in nodes[1:5]
        if (i,j) in arcs
            if value(x[(i,j)]) > 0
                println("Hire ", value(x[(i,j)]), " professors of type ", j, " to teach class level ", i)
            end
        end
    end
end
```

```
Hire 5.0 professors of type I1 to teach class level A
Hire 3.0 professors of type I1 to teach class level C
Hire 1.0 professors of type I1 to teach class level E
Hire 7.0 professors of type I2 to teach class level C
Hire 8.0 professors of type I3 to teach class level B
Hire 2.0 professors of type I3 to teach class level C
Hire 11.0 professors of type I4 to teach class level D
Hire 3.0 professors of type I5 to teach class level E
Coin0506I Presolve 9 (-1) rows, 14 (0) columns and 27 (-1) elements
Clp0006I 0  Obj 0 Primal inf 39.999995 (5)
Clp0006I 11  Obj 12565
Clp0000I Optimal - objective value 12565
Coin0511I After Postsolve, objective 12565, infeasibilities - dual 0 (0), primal 0 (0)
Clp0032I Optimal objective 12565 - 11 iterations time 0.002, Presolve 0.00
```

Problem 4 -- A tricky one

We're going to model a math puzzle as a shortest path problem. Start with the number 1. At each step, you can either triple the current number or add one to the current number. What is the minimal number of operations (steps) needed to get from 1 to 2020? There are a few different ways to solve this problem. For full credit, build an optimization model structured as a shortest path problem. Hint: you may find the following code snippet useful:

In [14]:

```julia
# Solution 4
using JuMP, Clp

# nodes will just be all numbers from 1 to 2020
ln = 2020
N = 1:ln
arcs = []
for i in N
    append!(arcs,[(i,i+1)])
    append!(arcs,[(i,i*3)])
end

cost = Dict(zip(arcs,ones(length(arcs))))
b = Dict(zip(N,zeros(length(N))))
b[1] = 1
b[ln] = -1

m = Model(Clp.Optimizer)

@variable(m, 0 <= x[arcs] <= 1)

@objective(m, Min, sum(cost[a]*x[a] for a in arcs))

@constraint(m, supp[i in N], sum(x[a] for a in arcs if a[1] == i) - sum(x[a] for a in arcs if a[2] == i) == b[i

optimize!(m)

println("minimum operations: ", objective_value(m))

for a in arcs
    if value(x[a]) > 0
        println("use arc ", a)
    end
end
```

```
minimum operations: 15.0
use arc (1, 2)
use arc (2, 6)
use arc (6, 7)
use arc (7, 8)
use arc (8, 24)
use arc (24, 72)
use arc (72, 73)
use arc (73, 74)
use arc (74, 222)
use arc (222, 223)
use arc (223, 224)
use arc (224, 672)
use arc (672, 673)
use arc (673, 2019)
use arc (2019, 2020)
Coin0506I Presolve 1342 (-678) rows, 2462 (-1578) columns and 4475 (-2257) elements
Clp0006I 0  Obj 3 Primal inf 1.999998 (2) Dual inf 1.999998 (2)
Clp0006I 33  Obj 15
Clp0000I Optimal - objective value 15
Coin0511I After Postsolve, objective 15, infeasibilities - dual 0 (0), primal 0 (0)
Clp0032I Optimal objective 15 - 33 iterations time 0.022, Presolve 0.02
```