CS 524 - Homework 10

Question 1a

```
In [36]: using JuMP, Gurobi
         data = [310 355 395 375 355 330 310 290;
                  100 130 160 160 160 160 160 160]
         steel bar = 45
         weights = [2.5 5 10 25 45]
         sets = 8
         num weights = 5
         m gym = Model(with optimizer(Gurobi.Optimizer, outputFlag=0))
         weights sol = [@variable(m gym, integer = true)] for i = 1:num weights, j = 1:sets, z = 1:2]
         decision sol = [@variable(m gym, binary = true)] for i = 1:num weights, j = 1:sets, z = 1:2]
         @constraint(m gym, weights sol .>= 0)
         for s in 1:sets
             for ex in 1:2
                  sum = 0
                  for w in 1:num weights
                      @constraint(m gym, (1 - decision sol[w, s, ex])weights sol[w, s, ex] == 0)
                      sum += 2decision sol[w, s, ex]weights sol[w, s, ex]weights[w]
                  end
                  sum += steel bar
                  label = data[ex, s]
                  @constraint(m gym, label == sum)
              end
         end
         \emptysetexpression(m gym, Obj, sum(weights sol[i, j, k] for i = 1:num weights, j = 1:sets, k = 1:2))
         @objective(m gym, Min, Obj)
         optimize!(m gym)
         opt weights = value.(weights sol[:,:,1])
         opt decision = value.(decision sol[:,:,1])
         value.(decision_sol) .* value.(weights_sol)*2
```

```
Set parameter Username
        Warning: your license will expire in 10 days
        Academic license - for non-commercial use only - expires 2022-05-14
        5×8×2 Array{Float64, 3}:
Out[36]:
        [:, :, 1] =
         2.0 0.0 0.0 0.0 0.0 2.0 2.0 2.0
         2.0 0.0 0.0 0.0 0.0 0.0 2.0 2.0
         2.0 4.0 2.0 0.0 4.0 0.0 2.0 0.0
         2.0 0.0 6.0 6.0 0.0 4.0 2.0 2.0
         4.0 6.0 4.0 4.0 6.0 4.0 4.0 4.0
        [:, :, 2] =
         2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0
         0.0 2.0 0.0 0.0 0.0 0.0 0.0 0.0
         0.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0
         2.0 2.0 0.0 0.0 0.0 0.0 0.0 0.0
         0.0 0.0 2.0 2.0 2.0 2.0 2.0 2.0
```

Question 1b

```
In [41]: using JuMP, Gurobi
         m homegym = Model(with optimizer(Gurobi.Optimizer, outputFlag=0))
         weights sol = [@variable(m homegym, integer = true)] for i = 1:num weights, j = 1:sets, z = 1:2]
         decision sol = [@variable(m homegym, binary = true)] for i = 1:num weights, j = 1:sets, z = 1:2]
         upper bound = [@variable(m homegym) for i = 1:num weights]
         @constraint(m homegym, weights sol .>= 0)
         for s in 1:sets
             for ex in 1:2
                 sum = 0
                 for w in 1:num weights
                     @constraint(m homegym, (1 - decision sol[w, s, ex]) * weights sol[w, s, ex] == 0)
                     sum += 2 * decision sol[w, s, ex] * weights sol[w, s, ex] * weights[w]
                  end
                  sum += steel bar
                 label = data[ex, s]
                  @constraint(m homegym, label == sum)
             end
```

```
end
# new constraint
for w in 1:num weights
    sum plates = 0
    for s in 1:sets
        for ex in 1:2
            sum plates += 2 * decision sol[w, s, ex] * weights sol[w, s, ex]
        end
    end
    @constraint(m homegym, sum plates .<= upper bound[w])</pre>
end
@expression(m homegym, Obj, sum(weights sol[i, j, k] for i = 1:num weights, j = 1:sets, k = 1:2))
@objective(m homegym, Min, Obj)
optimize!(m homegym)
println()
println("Buy 2
                 2.51b weights")
println("Buy 2
                 5.0lb weights")
println("Buy 4
                 10.0lb weights")
println("Buy 2
                 25.0lb weights")
println("Buy 6
                45.0lb weights")
Set parameter Username
Warning: your license will expire in 10 days
Academic license - for non-commercial use only - expires 2022-05-14
Buy 2
         2.5lb weights
```

Question 2

Buy 2

5.0lb weights

Buy 4 10.0lb weights Buy 2 25.0lb weights Buy 6 45.0lb weights

```
In [43]: using JuMP, Cbc, NamedArrays

voter_data = [
    80 34
    60 44
```

```
40 44
    20 24
    40 114
    40 64
    70 14
    50 44
   70 54
    70 64
party = [:r, :d]
districts = [:A, :B, :C, :D, :E]
cities = collect(1:10)
voter array = NamedArray(voter data, (cities, party))
m vote = Model(with optimizer(Cbc.Optimizer, logLevel=0))
@variable(m vote, present[districts, cities], Bin) # to see which city is in which district
@variable(m vote, win[districts], Bin) # to see if district is won
# ensure all voters in a city must be in the same district
for city in cities
    @constraint(m vote, sum(present[d,city] for d in districts) == 1)
end
# each district must contain between 150,000 and 250,000 voters
for d in districts
    @constraint(m vote, 150 <= sum((voter array[c,:r] + voter array[c,:d])*present[d,c] for c in cities))</pre>
    @constraint(m vote, sum((voter array[c,:r] + voter array[c,:d])*present[d,c] for c in cities) <= 250)</pre>
end
for d in districts
    @constraint(m vote, sum((voter array[c,:r] - voter array[c,:d])*present[d,c] for c in cities) <= 100*(1-win[d]))</pre>
end
# maximize republican wins in each district (rep must win)
@objective(m vote, Max, sum(win))
optimize!(m vote)
res = value.(present)
println("The republicans win in ", objective value(m vote), " out of 5 districts.")
println()
println("The assignment of cities should be: ")
println("
                        1
                              2
                                    3
                                          4
                                              5
                                                      6
                                                                         9
                                                                               10")
for d in districts
```

```
print("District ", d, ":")
  for c in cities
     print(" ", res[d, c])
  end
  println()
end
```

The republicans win in 3.0 out of 5 districts.

```
The assignment of cities should be:
              1
                    2
                          3
                                      5
                                           6
                                                 7
                                                       8
                                                             9
                                                                   10
District A:
             0.0
                   0.0
                         0.0
                               0.0
                                     0.0
                                          0.0
                                                1.0
                                                      0.0
                                                            0.0
                                                                  1.0
District B:
             0.0
                   0.0
                         0.0
                               0.0
                                          0.0
                                                0.0
                                                      0.0
                                    1.0
                                                            0.0
                                                                  0.0
District C:
             1.0
                  0.0
                        0.0
                               0.0
                                     0.0
                                          0.0
                                                0.0
                                                      0.0
                                                            1.0
                                                                  0.0
District D:
             0.0
                   1.0
                        0.0
                               0.0
                                     0.0
                                          1.0
                                                0.0
                                                      0.0
                                                           0.0
                                                                  0.0
District E:
             0.0
                   0.0 1.0 1.0
                                     0.0
                                          0.0
                                                0.0
                                                      1.0
                                                                  0.0
                                                            0.0
```

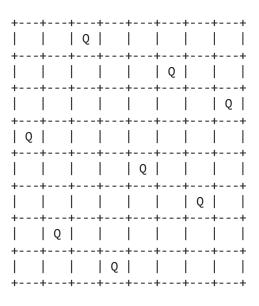
Question 3 Helper

```
In [52]: # function to print chess board with queens
          function printBoard(Q,n)
              print("+")
              for i in 1:n
                  print("---+")
              end
              print("\n")
              for i in 1:n
                  print(" ")
                  for j in 1:n
                      Q[i,j] == 1 ? print("Q") : print(" ")
                      print(" | ")
                  end
                  print("\n")
                  print("+")
                  for j in 1:n
                      print("---+")
                  end
                  print("\n")
              end
          end
```

Out[52]: printBoard (generic function with 1 method)

Question 3a

```
In [66]: using JuMP, Cbc
          n = 8
          m queen a = Model(with optimizer(Cbc.Optimizer, logLevel=0))
          @variable(m queen a, q[1:n queens,1:n queens], Bin) # position of queen(s)
          for i in 1:n queens
              # ensure one queen per row and column
              @constraint(m \text{ queen a, } sum(q[i,j] \text{ for } j \text{ in } 1:n \text{ queens}) == 1)
              @constraint(m_queen_a, sum(q[j,i] for j in 1:n_queens) == 1)
              # ensure one queen per left and right upper diagonal
              @constraint(m queen a, sum(q[i+k,1+k] for k in 0:n queens-i) <= 1)</pre>
              @constraint(m queen a, sum(q[i-k,1+k] \text{ for } k \text{ in } 0:i-1) <= 1)
          end
          # ensure one queen per left and right lower diagonal
          for j in 2:n_queens
              @constraint(m_queen_a, sum(q[1+k,j+k] for k in 0:n_queens-j) <= 1)</pre>
              @constraint(m queen a, sum(q[n queens-k,j+k] for k in 0:n queens-j) <= 1)
          end
          # no two queens threaten each other w/o 180 rotational symmetry
          @objective(m queen a, Min, sum(q))
          optimize!(m_queen_a)
          Q = value.(q)
          printBoard(Q, n queens)
```



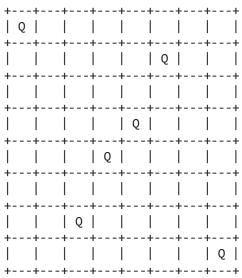
Question 3b

```
In [67]: using JuMP, Cbc
         n_queens = 8
          m queen b = Model(with optimizer(Cbc.Optimizer, logLevel=0))
         @variable(m queen b, q[1:n queens,1:n queens], Bin) # position of queen(s)
         for i in 1:n queens
              # ensure one queen per row and column
              @constraint(m_queen_b, sum(q[i,j] for j in 1:n_queens) == 1)
              @constraint(m_queen_b, sum(q[j,i] for j in 1:n_queens) == 1)
              # ensure one queen per left and right upper diagonal
              @constraint(m queen b, sum(q[i+k,1+k] for k in 0:n queens-i) <= 1)</pre>
              @constraint(m queen b, sum(q[i-k,1+k] \text{ for } k \text{ in } 0:i-1) <= 1)
          end
          # ensure one queen per left and right lower diagonal
          for j in 2:n_queens
              @constraint(m_queen_b, sum(q[1+k,j+k] for k in 0:n_queens-j) <= 1)</pre>
              @constraint(m queen b, sum(q[n queens-k,j+k] for k in 0:n queens-j) <= 1)
          end
          # symmetry constraint
          for i in 1:n_queens
```

Question 3c

```
+ sum(q[i+k,j-k] for k = max(1-i,n_queens-j):min(n_queens-i,1-j)) >= 1
    end
end
# each empty cell is threatened w/o 180 rotational symmetry
@objective(m_queen_c, Min, sum(q))
optimize!(m_queen_c)
Q = value.(q)
printBoard(Q, n_queens)
```

Question 3d



In []: