In [1]:

```
using JuMP, Gurobi
```

1. Voting

Governor Blue of the state of Berry is attempting to get the state legislator to gerrymander Berry's congressional districts. The state consists of ten cities, and the numbers of registered Republicans and Democrats (in thousands) in each city are shown below.

City	Republicans	Democrats
1	80	34
2	60	44
3	40	44
4	20	24
5	40	114
6	40	64
7	70	14
8	50	44
9	70	54
10	70	64

Berry has five congressional representatives. To form the five congressional districts, cities must be grouped together according to the following restrictions:

1. Districts cannot subdivide cities; all voters in a city must be in the same district.
2. Each district must contain between 150,000 and 250,000 voters (there are no independent voters).

Governor Blue is a Democrat. Assume 100% voter turnout and that each voter always votes according to their registered party. Formulate and solve an optimization problem to help Governor Blue maximize the number of congressional districts that have a Democratic majority.

In [2]:

```
# I haven't separately formulated the problem but have written comments here in
the Julia code itself
# for each step

# Given number of registered republicans and democratics (in thousands) in 10 ci
ties
republicans = [80 60 40 20 40 40 70 50 70 70]
democrats = [34 44 44 24 114 64 14 44 54 64]

m1 = Model(solver=GurobiSolver(OutputFlag=0))

# Binary variable to keep track of city assigned to a particular district
@variable(m1, z1[1:10,1:5], Bin)

# to keep track if district 1-5 was won by democrats
@variable(m1, z2[1:5], Bin)

# Each district must contain between 150,000 and 250,000 voters (there are no in
dependent voters).
for j in 1:5
    @constraint(m1, sum((republicans[i] + democrats[i])*z1[i,j] for i in 1:10) >
= 150)
end

# Max constraint
for j in 1:5
    @constraint(m1, sum((republicans[i] + democrats[i])*z1[i,j] for i in 1:10) <
= 250)
end

# Districts cannot subdivide cities; all voters in a city must be in the same di
strict.
for i in 1:10
    @constraint(m1, sum(z1[i, j] for j in 1:5) == 1)
end

M = 10000
# republican win scenario -- the idea was to see the constraint for logic statem
ent from the class slide
for j in 1:5
    @constraint(m1, sum((republicans[i] - democrats[i]) * z1[i, j] for i in 1:10
) <= M*(1 - z2[j]))
end

# Governor Blue maximize the number of congressional districts that have a Democ
ratic majority.
@objective(m1, Max, sum(z2[j] for j in 1:5))

solve(m1)
```

Out[2]:

:Optimal

In [3]:

```
println("District that have democratic majority are 2, 3 and 4")
getvalue(z2)
```

```
District that have democratic majority are 2, 3 and 4
```

Out[3]:

```
5-element Array{Float64,1}:
 0.0
 1.0
 1.0
 1.0
 0.0
```

In [4]:

```
z1opt = getvalue(z1)
```

Out[4]:

```
10×5 Array{Float64,2}:
 0.0  0.0  0.0  0.0  1.0
 1.0  0.0  0.0  0.0  0.0
 0.0  1.0  0.0  0.0  0.0
 0.0  1.0  0.0  0.0  0.0
 0.0  0.0  0.0  1.0  0.0
 0.0  0.0  1.0  0.0  0.0
 0.0  0.0  0.0  0.0  1.0
 0.0  1.0  0.0  0.0  0.0
 1.0  0.0  0.0  0.0  0.0
 0.0  0.0  1.0  0.0  0.0
```

District 1 --> 2, 9
District 2 --> 3, 4, 8
District 3 --> 6, 10
District 4 --> 5
District 5 --> 1, 7

2. Paint production

As part of its weekly production, a paint company produces five batches of paints, always the same, for some big clients who have a stable demand. Every paint batch is produced in a single production process, all in the same blender that needs to be cleaned between each batch. The durations of blending paint batches 1 to 5 are 40, 35, 45, 32 and 50 minutes respectively. The cleaning times depend of the colors and the paint types. For example, a long cleaning period is required if an oil-based paint is produced after a water-based paint, or to produce white paint after a dark color. The times are given in minutes in the following matrix A where Aij denotes the cleaning time after batch i if it is followed by batch j.

In [5]:

```
# Number of batches in the paint production problem
num_of_batches = 5

# The durations of blending paint batches 1 to 5 are 40, 35, 45, 32 and 50 minut
es respectively.
duration = [40 35 45 32 50]

# The times are given in minutes in the following matrix A where Aij denotes the
cleaning time
# after batch i if it is followed by batch j.
A = [
0 11 7 13 11
5 0 13 15 15
13 15 0 23 11
9 13 5 0 3
3 7 7 7 0
]

m2 = Model(solver=GurobiSolver(OutputFlag=0))

# Binary variable to store which batch follows which batch - this will be set to
1 if batch i is followed by batch j
@variable(m2, z[1:5,1:5], Bin)

# Variable to make sure there is no subcycle
@variable(m2, y[1:5])

# To make sure we have one follower and one successor for each batch
for j in 1:5
    @constraint(m2, sum(z[i, j] for i in 1:5) == 1)
end

for i in 1:5
    @constraint(m2, sum(z[i, j] for j in 1:5) == 1)
end

# Assuming that batch don't appear twice in a processing cycle. Hence when i ==
j we mark them as 0
for i in 1:5
```

```
    for j in 1:5
        if i == j
            @constraint(m2, z[i, j] == 0)
        end
    end
end

# constraints to not include sub-batch cycles in processing of batch
for i in 1:5
    for j in 2:5
        if i != j
            @constraint(m2, y[j] >= ((y[i] + 1) - (num_of_batches * (1 - z[i, j]
))))
        end
    end
end

# Objective is to minimize the batch cycle time - which is nothing but the sum o
f all blending time + cleaning
# time for the batch if selected

@objective(m2, Min, sum((duration[i] + A[i, j]) * z[i, j] for i in 1:5, j in 1:5
))
solve(m2)

println("Total cleaning and duration time: ", getobjectivevalue(m2))

total_blending_time = 0
for i in 1:5
    total_blending_time += duration[i]
end
println("Time spent in cleaning: ", getobjectivevalue(m2) - total_blending_time)
```

```
Total cleaning and duration time: 243.0
Time spent in cleaning: 41.0
```

In [6]:

```
for i in 1:5
    for j in 1:5
        print(Int64(getvalue(z[i,j])), " ")
    end
    print("\n")
end
```

```
0 0 0 1 0
1 0 0 0 0
0 0 0 0 1
0 0 1 0 0
0 1 0 0 0
```

Solution for other part What is the corresponding order of paint batches? The order will be applied every week, so the cleaning time between the last batch of one week and the first of the following week needs to be accounted for in the total duration of cleaning.
Using the above matrix to identify the order of paint batches -- and let's assume that we start the paint production cycle from batch 1.

1 --> 4 --> 3 --> 5 --> 2 --> 1

Basically, if we observe Paint 1 has 1 set to 4th (1, 4). Going to 4 it has 3 set to 1 (4, 3) and so (3, 5), (5, 2) and back to 1. Also, intuitively, between batches the cleaning time b/w the last batch of one week and the first of the following week should be min cleaning time among all 5 batches.

3. The Queens problem.

You are given a standard 8×8 chess board. The following problems involve placing queens on the board such that certain constraints are satisfied. For each of the following problems, model the optimization task as an integer program, solve it, and show what an optimal placement of queens on the board looks like.

Solution 3a Find a way to place 8 queens on the board so that no two queens threaten each other. We say that two queens threaten each other if they occupy the same row, column, or diagonal. Show what this placement looks like

In [7]:

```
m3a = Model(solver=GurobiSolver(OutputFlag=0))

@variable(m3a, z[1:8, 1:8], Bin)

# Horizontal sum for each row should be < equal to 1
for i in 1:8
    @constraint(m3a, sum(z[i, j] for j in 1:8) <= 1)
end

# Vertical sum for each row should be < equal to 1
for j in 1:8
    @constraint(m3a, sum(z[i, j] for i in 1:8) <= 1)
end

# Looking for each diagonals
diag2 = [-7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7]
diag1 = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]

for k in diag1
    temp = 0
    for i in 1:8
        for j in 1:8
            if i + j == k
                temp += z[i, j]
            end
        end
    end
    @constraint(m3a, temp <= 1)
end

for k in diag2
    temp = 0
    for i in 1:8
        for j in 1:8
            if i - j == k
                temp += z[i, j]
            end
        end
    end
    @constraint(m3a, temp <= 1)
end

@objective(m3a, Max, sum(z[i, j] for i in 1:8, j in 1:8))

solve(m3a)
getobjectivevalue(m3a)
```

Out[7]:

```
8.0
```

In [8]:

```
println("\n+---+---+---+---+---+---+---+---+")
for i in 1:8
    for j in 1:8
        if Int64(getvalue(z[i,j])) == 1
            print("| X ")
        else
            print("|   ")
        end
    end
    println("|\n+---+---+---+---+---+---+---+---+")
end
```

```
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   | X |
+---+---+---+---+---+---+---+---+
|   |   | X |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
| X |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   | X |   |   |
+---+---+---+---+---+---+---+---+
|   | X |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   | X |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   | X |   |
+---+---+---+---+---+---+---+---+
|   |   |   | X |   |   |   |   |
+---+---+---+---+---+---+---+---+
```

Solution 3b Repeat part (a) but this time find a placement of the 8 queens that has point symmetry. In other words, find a placement that looks the same if you rotate the board 180◦.

In [9]:

```
# The problem is similar to the 3a problem but with an additional constraint for
symmetry mentioned below in the code

m3b = Model(solver=GurobiSolver(OutputFlag=0))

@variable(m3b, z[1:8, 1:8], Bin)

# Horizontal sum for each row should be equal to 1
for i in 1:8
    @constraint(m3b, sum(z[i, j] for j in 1:8) == 1)
end

# Vertical sum for each row should be equal to 1
for j in 1:8
    @constraint(m3b, sum(z[i, j] for i in 1:8) == 1)
end
```

```
diag2 = [-7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7]
diag1 = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]

for k in diag1
    temp = 0
    for i in 1:8
        for j in 1:8
            if i + j == k
                temp += z[i, j]
            end
        end
    end
    @constraint(m3b, temp <= 1)
end

for k in diag2
    temp = 0
    for i in 1:8
        for j in 1:8
            if i - j == k
                temp += z[i, j]
            end
        end
    end
    @constraint(m3b, temp <= 1)
end

# Constraint for 180 degrees -- this is the additional constraint we will need f
or the problem to place queen
# in symmetric fashion
for i in 1:8
    for j in 1:8
        if j >= i
            @constraint(m3b, z[i, j] == z[8-i+1, 8-j+1])
        end
    end
end

solve(m3b)
```

Out[9]:

:Optimal

In [10]:

```
println("\n+---+---+---+---+---+---+---+---+")
for i in 1:8
    for j in 1:8
        if Int64(getvalue(z[i,j])) == 1
            print("| X ")
        else
            print("|   ")
        end
    end
    println("|\n+---+---+---+---+---+---+---+---+")
end
```

```
+---+---+---+---+---+---+---+---+
|   |   |   |   | X |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   | X |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
| X |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   | X |   |
+---+---+---+---+---+---+---+---+
|   | X |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   | X |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   | X |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   | X |   |   |   |   |
+---+---+---+---+---+---+---+---+
```

Solution 3c What is the smallest number of queens that we can place on the board so that each empty cell is threatened by at least one queen? Show a possible optimal placement.

In [11]:

```
# For this problem we will consider each cell and for each cell the sum of horiz
ontal, vertical and diagonals should
# be greater than equal to 1. So basically, we will iterate over each row and co
lumn (each cell) and will sum up the
# binary variables across that cell's horizontal, vertical and diagonals

m3c = Model(solver=GurobiSolver(OutputFlag=0))

@variable(m3c, z[1:8, 1:8], Bin)

# for each row in the board 1 - 8
for i in 1:8
    # for each column in the board 1 - 8
    for j in 1:8
        # for cell [i, j] we will calculate the horizontal row sum
        @expression(m3c, h, sum(z[i, k] for k in 1:8))

        # for cell [i, j] we will calculate the vertical row sum
        @expression(m3c, v, sum(z[k, j] for k in 1:8))

        # Lower right diagonal sum for the the cell [i, j]
        @expression(m3c, lr, sum(z[i+k, j+k] for k in 0:8-max(i, j)))

        # Upper left diagonal for the cell [i, j]
        @expression(m3c, ul, sum(z[i-k, j-k] for k in 0:min(i, j)-1))

        # Left Lower diagonal sum for the cell [i, j]
        @expression(m3c, ll, sum(z[i+k, j-k] for k in 0:min(8-i+1, j)-1))

        # Right Upper diagonal sum for the cell [i, j]
        @expression(m3c, ru, sum(z[i-k, j+k] for k in 0:min(i, 8-j+1)-1))

        # Sum of all the binary variables for cell [i, j] in all directions shou
ld be >= 1
        @constraint(m3c, h + v + lr + ul + ll + ru >= 1)
    end
end

# Objective is to minimize the sum of binary variables on the 8 x 8 board
@objective(m3c, Min, sum(z[i, j] for i in 1:8, j in 1:8))
solve(m3c)
```

Out[11]:

:Optimal

In [12]:

```
println("Placement of with smallest number of queens that we can place on the bo
ard so that each empty cell is
threatened by at least one queen")

println("\n+---+---+---+---+---+---+---+---+")
for i in 1:8
    for j in 1:8
        if Int64(getvalue(z[i,j])) == 1
            print("| X ")
        else
            print("|   ")
        end
    end
    println("|\n+---+---+---+---+---+---+---+---+")
end
```

```
Placement of with smallest number of queens that we can place on the
board so that each empty cell is
threatened by at least one queen

+---+---+---+---+---+---+---+---+
|   |   |   |   |   | X |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
| X |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   | X |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   | X |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   | X |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
```

In [13]:

```
getobjectivevalue(m3c)
```

Out[13]:

```
5.0
```

Solution 3d Repeat part (c) but this time find a placement of the queens that also has point symmetry. Does the minimum number of queens required change? Show a possible optimal placement.

In [14]:

```
# For this problem we will consider each cell and for each cell the sum of horiz

ontal, vertical and diagonals should
# be greater than equal to 1. So basically, we will iterate over each row and co
lumn (each cell) and will sum up the
# binary variables across that cell's horizontal, vertical and diagonals

# +

# Symmetry constraint

m3d = Model(solver=GurobiSolver(OutputFlag=0))

# Binary variable for each cell of the board
@variable(m3d, z[1:8, 1:8], Bin)

# for each row in the board 1 - 8
for i in 1:8
    # for each column in the board 1 - 8
    for j in 1:8
        # for cell [i, j] we will calculate the horizontal row sum
        @expression(m3d, h, sum(z[i, k] for k in 1:8))
        # for cell [i, j] we will calculate the vertical row sum
        @expression(m3d, v, sum(z[k, j] for k in 1:8))

        # Lower right diagonal sum for the the cell [i, j]
        @expression(m3d, lr, sum(z[i+k, j+k] for k in 0:8-max(i, j)))

        # Upper left diagonal for the cell [i, j]
        @expression(m3d, ul, sum(z[i-k, j-k] for k in 0:min(i, j)-1))

        # Left Lower diagonal sum for the cell [i, j]
        @expression(m3d, ll, sum(z[i+k, j-k] for k in 0:min(8-i+1, j)-1))

        # Right Upper diagonal sum for the cell [i, j]
        @expression(m3d, ru, sum(z[i-k, j+k] for k in 0:min(i, 8-j+1)-1))

        # Sum of all the binary variables for cell [i, j] in all directions shou
ld be >= 1
        @constraint(m3d, h + v + lr + ul + ll + ru >= 1)
    end
end

# Constraint for 180 degrees
for i in 1:8
    for j in 1:8
        if j >= i
            @constraint(m3d, z[i, j] == z[8-i+1, 8-j+1])
        end
    end
end


@objective(m3d, Min, sum(z[i, j] for i in 1:8, j in 1:8))
```

```
solve(m3d)
```

Out[14]:

```
:Optimal
```

In [15]:

```
println("Placement of with smallest number of queens that we can place on the bo
ard so that each empty cell is
threatened by at least one queen")
println("\n+---+---+---+---+---+---+---+---+")
for i in 1:8
    for j in 1:8
        if Int64(getvalue(z[i,j])) == 1
            print("| X ")
        else
            print("|   ")
        end
    end
    println("|\n+---+---+---+---+---+---+---+---+")
end
```

```
Placement of with smallest number of queens that we can place on the
board so that each empty cell is
threatened by at least one queen

+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   | X |   |
+---+---+---+---+---+---+---+---+
|   |   |   | X |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   | X |   |
+---+---+---+---+---+---+---+---+
|   | X |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   | X |   |   |   |
+---+---+---+---+---+---+---+---+
|   | X |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
```

The number of queens seems to be 6 here in the Solution of 3d. One more than 3c solution

In [ ]:

```
```