2. Paint production [15 pts].

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 on 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 A_{ij} denotes the cleaning time after batch i if it is followed by batch j.

$$\begin{bmatrix} 0 & 11 & 7 & 13 & 11 \\ 5 & 0 & 13 & 15 & 15 \\ 13 & 15 & 0 & 23 & 11 \\ 9 & 13 & 5 & 0 & 3 \\ 3 & 7 & 7 & 7 & 0 \end{bmatrix}$$

Since the company has other activities, it wishes to deal with this weekly production in the shortest possible time (blending and cleaning). 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.

In [2]:
```
using NamedArrays

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]


paints = [1:5;]

blending_times = [40, 35, 45, 32, 50]

c = NamedArray(A, (paints, paints))
N = size(A,1);
```

In [3]:
```
using JuMP, Cbc, Gurobi, Mosek, GLPK

m = Model(solver = MosekSolver())

@variable(m, x[paints,paints], Bin)

@constraint(m, c1[j in paints], sum(x[i,j] for i in paints) == 1)
@constraint(m, c2[i in paints], sum(x[i,j] for j in paints) == 1)
@constraint(m, c3[i in paints], x[i,i] == 0)

@objective(m, Min, sum(x[i,j]*c[i,j] for i in paints, j in paints))

# Miller-Tucker-Zemlin variables and constraints
@variable(m, u[paints])
@constraint(m, c4[i in paints, j in paints[2:end]], u[i] - u[j] + N*x[i,j] <= (N - 1))

solve(m)
```

Out[3]: :Optimal

In [14]:
```
x_opt = getvalue(x)
order_opt = getAllSubtours(x_opt)

println("The paints should be made in the order: ", (order_opt))

println()
println("The total cleaning time taken is therefore: ", getobjectivevalue(m), " minutes")
```

```
The paints should be made in the order: Any[[1,2,5,3,4,1]]

The total cleaning time taken is therefore: 41.0 minutes
```

In [1]:
```
# HELPER FUNCTION: returns the cycle containing the paint START.
function getSubtour(x,start)
    subtour = [start]
    while true
        j = subtour[end]
        for k in paints
            if x[k,j] == 1
                push!(subtour,k)
                break
            end
        end
        if subtour[end] == start
            break
        end
    end
    return subtour
end

# HELPER FUNCTION: returns a list of all cycles
function getAllSubtours(x)
    nodesRemaining = paints
    subtours = []
    while length(nodesRemaining) > 0
        subtour = getSubtour(x,nodesRemaining[1])
        push!(subtours, subtour)
        nodesRemaining = setdiff(nodesRemaining,subtour)
    end
    return subtours
end
```

Out[1]: getAllSubtours (generic function with 1 method)