Homework 7 Question 1: Thrift Store

How should you make change for 99 cents if the goal is to minimize the total weight of the coins used? The following table shows the weight of each type of coin. You may use any number of each type of coin

Type of Coin	penny	nickel	dime	quarter
Weight (grams)	2.500	5.000	2.268	5.670

Problem Data

In [1]:
```
typeOfCoins = [:penny, :nickel, :dime, :quarter]
weight = Dict(zip(typeOfCoins, [2.5, 5.0, 2.268, 5.670]))
value = Dict(zip(typeOfCoins, [1, 5, 10, 25]));
```

Problem Model

In [2]:
```
using JuMP, Cbc

m = Model(solver=CbcSolver())

@variable(m, coins[typeOfCoins] >= 0, Int)  # number of each type of coins

@constraint(m, sum(value[c]*coins[c] for c in typeOfCoins) == 99)  # constraint for total value 99

@objective(m, Min, sum(weight[c]*coins[c] for c in typeOfCoins))  # minimize total weight of change

m
```

Out[2]:

$$\begin{aligned} \min \quad & 2.5 coins_{penny} + 5 coins_{nickel} + 2.268 coins_{dime} + 5.67 coins_{quarter} \\ \text{Subject to} \quad & coins_{penny} + 5 coins_{nickel} + 10 coins_{dime} + 25 coins_{quarter} = 99 \\ & coins_i \geq 0, \in \mathbb{Z}, \quad \forall i \in \{penny, nickel, dime, quarter\} \end{aligned}$$

In [3]:
```
status = solve(m)
println("Status: ", status)
println("Coins ", getvalue(coins))
println("Total weight ", getobjectivevalue(m), "gms")
```

```
Status: Optimal
Coins coins: 1 dimensions:
[  penny] = 4.0
[ nickel] = 0.0
[   dime] = 7.000000000000001
[quarter] = 1.0

Total weight 31.546gms
```

Homework 7 Question 2: Comquat Computers.

Comquat owns four production plants at which personal computers are produced. Comquat can sell up to 20,000 computers per year at a price of $3,500 per computer. For each plant the production capacity, cost per computer, and fixed cost of operating the plant for a year are given below. Determine how Comquat can maximize its yearly profit from computer production.

Plant Production	capacity	Plant fixed cost ($ Million)	Cost per computer ($)
1	10,000	9	1,000
2	8,000	5	1,700
3	9,000	3	2,300
4	6,000	1	2,900

Problem Data

In [1]:
```
Plants = [:one, :two, :three, :four]
raw_capacity = [10000, 8000, 9000, 6000]
capacity = Dict(zip(Plants, raw_capacity))
fixedCost = Dict(zip(Plants, [9, 5, 3, 1]))
costPerPc = Dict(zip(Plants, [1000, 1700, 2300, 2900]));
```

Problem Model

In [2]:
```
using JuMP, Cbc

m = Model(solver=CbcSolver())

@variable(m, production[Plants] >= 0, Int)  # Production per plant
@variable(m, plantUsed[Plants], Bin)

  # capacity constraints
@constraint(m, cap_constr[p in Plants], production[p] <= plantUsed[p]*capacity[p])
@constraint(m, sum(production[p] for p in Plants) <= 20000)  # Selling constraints

# expression for cost
@expression(m, cost, sum(costPerPc[p]*production[p] + plantUsed[p]*fixedCost[p]*1000000
                    for p in Plants))
# expression for sell price
@expression(m, sellPrice, sum(production[p]*3500 for p in Plants))

# objective to maximize profit
@objective(m, Max, sellPrice - cost)
m
```

Out[2]:

$$\begin{aligned}
\max\quad & 2500production_{one} + 1800production_{two} + 1200production_{three} + 600production_{four} - 9.0e6plantUsed_{one} - \\
& - 3.0e6plantUsed_{three} - 1.0e6plantUsed_{four} \\
\text{Subject to}\quad & production_{one} - 10000plantUsed_{one} \leq 0 \\
& production_{two} - 8000plantUsed_{two} \leq 0 \\
& production_{three} - 9000plantUsed_{three} \leq 0 \\
& production_{four} - 6000plantUsed_{four} \leq 0 \\
& production_{one} + production_{two} + production_{three} + production_{four} \leq 20000 \\
& production_i \geq 0, \in \mathbb{Z}, \quad \forall i \in \{one, two, three, four\} \\
& plantUsed_i \in \{0, 1\} \quad \forall i \in \{one, two, three, four\}
\end{aligned}$$

In [3]:
```
status = solve(m)
println("Status: ",status)
println("Max profit ", getobjectivevalue(m))
println("Production ", getvalue(production))
println("Plant Used ", getvalue(plantUsed))
```

```
Status: Optimal
Max profit 2.56e7
Production production: 1 dimensions:
[  one] = 10000.0
[  two] = 8000.0
[three] = 0.0
[ four] = 2000.0

Plant Used plantUsed: 1 dimensions:
[  one] = 1.0
[  two] = 1.0
[three] = 0.0
[ four] = 1.0
```

Homework 7 Question 3: ABC Investments.

ABC Inc. is considering several investment options. Each option has a minimum investment required as well as a maximum investment allowed. These restrictions, along with the expected return are summarized in the following table (figures are in millions of dollars):

Option	Minimum investment	Maximum investment	Expected return (%)
1	3	27	13
2	2	12	9
3	9	35	17
4	5	15	10
5	12	46	22
6	4	18	12

Because of the high-risk nature of Option 5, company policy requires that the total amount invested in Option 5 be no more that the combined amount invested in Options 2, 4 and 6. In addition, if an investment is made in Option 3, it is required that at least a minimum investment be made in Option 6. ABC has $80 million to invest and obviously wants to maximize its total expected return on investment. Which options should ABC invest in, and how much should be invested?

Problem Data

In [1]:
```
options = [:one, :two, :three, :four, :five, :six]
minInv = Dict(zip(options, [3, 2, 9, 5, 12, 4]))
maxInv = Dict(zip(options, [27, 12, 35, 15, 46, 18]))
expReturn = Dict(zip(options, [13, 9, 17, 10, 22, 12]));
```

Problem Model

In [2]:
```
using JuMP, Cbc

m = Model(solver=CbcSolver())

# Boolean decision variable to correspond to stocks invested in
@variable(m, investedIn[options], Bin)
@variable(m, investment[options]) # Investement made in each stock

# Min investment constraint
@constraint(m, minConstr[o in options], investment[o] >= investedIn[o]*minInv[o])
# Max investment constraint
@constraint(m, maxConstr[o in options], investment[o] <= investedIn[o]*maxInv[o])

# Constraint for 5 equal to 2+4+6
@constraint(m, investment[:five] <= sum(investment[o] for o in [:two, :four, :six]))
# Constraint for 6 to be compulsory if invested in three
@constraint(m, investment[:six] >= investedIn[:three]*minInv[:six])

# Total investment available constraint
@constraint(m, sum(investment[o] for o in options) <= 80)

# Objective to maximize profit / return.
@objective(m, Max, sum(investment[o]*(expReturn[o]/100.) for o in options))
m
```

Out[2]:

$$
\begin{aligned}
\max \quad & 0.13investment_{one} + 0.09investment_{two} + 0.17investment_{three} + 0.1investment_{four} + 0.22investment_{five} + 0.12i \\
\text{Subject to} \quad & investment_{one} - 3investedIn_{one} \geq 0 \\
& investment_{two} - 2investedIn_{two} \geq 0 \\
& investment_{three} - 9investedIn_{three} \geq 0 \\
& investment_{four} - 5investedIn_{four} \geq 0 \\
& investment_{five} - 12investedIn_{five} \geq 0 \\
& investment_{six} - 4investedIn_{six} \geq 0 \\
& investment_{one} - 27investedIn_{one} \leq 0 \\
& investment_{two} - 12investedIn_{two} \leq 0 \\
& investment_{three} - 35investedIn_{three} \leq 0 \\
& investment_{four} - 15investedIn_{four} \leq 0 \\
& investment_{five} - 46investedIn_{five} \leq 0 \\
& investment_{six} - 18investedIn_{six} \leq 0 \\
& investment_{five} - investment_{two} - investment_{four} - investment_{six} \leq 0 \\
& investment_{six} - 4investedIn_{three} \geq 0 \\
& investment_{one} + investment_{two} + investment_{three} + investment_{four} + investment_{five} + investment_{six} \leq 80 \\
& investedIn_i \in \{0, 1\} \quad \forall i \in \{one, two, \ldots, five, six\} \\
& investment_{free} \quad \forall i \in \{one, two, \ldots, five, six\}
\end{aligned}
$$

In [3]:
```
status = solve(m)
println("Status: ", status)
println("Investments ", getvalue(investment))
println("Invested In ", getvalue(investedIn))
println("Expected Return ", getobjectivevalue(m))
println("Expected Return (%) ", getobjectivevalue(m)*100/sum(getvalue(investment)))
```

```
Status: Optimal
Investments investment: 1 dimensions:
[  one] = 0.0
[  two] = 0.0
[three] = 34.99999999999999
[ four] = 5.000000000000001
[ five] = 22.500000000000004
[  six] = 17.5

Invested In investedIn: 1 dimensions:
[  one] = 0.0
[  two] = 0.0
[three] = 1.0
[ four] = 1.0
[ five] = 1.0
[  six] = 1.0

Expected Return 13.500000000000004
Expected Return (%) 16.875000000000007
```

Homework 7 Question 4: Lights Out.

In Tiger Electronic's handheld solitaire game Lights Out, the player strives to turn out all 25 lights that make up a 5 × 5 grid of cells. On each turn, the player is allowed to click on any one cell. Clicking on a cell activates a switch that causes the states of the cell and its (edge) neighbors to change from on to off, or from off to on. Corner cells are considered to have 2 neighbors, edge cells to have three, and interior cells to have four. Find a way to turn out all the lights in as few turns as possible (starting from the state where all lights are on).

Problem Data

In [1]:
```
rows = 5
columns = 5;
```

In [2]:
```
# Operations allowed to get neighbor of a cell.
# First column element is for row and second column for column operation
ops = [1 0;
       0 1;
      -1 0;
       0 -1]

# Helper function to generate neihbours
function get_nbr(i, j, max_row, max_col)
    retval = []
    for o in 1:length(ops[:,1])
        if i + ops[o,1] <= rows && i + ops[o,1] >= 1
            if j + ops[o,2] <= columns && j + ops[o,2] >= 1
                push!(retval,(i + ops[o,1], j + ops[o,2]))
            end
        end
    end
    return retval
end;
```

In [3]:
```
println("Neighbours")
[println("(",i,",",j,"): ",get_nbr(i,j, rows,columns)) for i in 1:rows for j in 1:columns];
```

```
Neighbours
(1,1): Any[(2,1),(1,2)]
(1,2): Any[(2,2),(1,3),(1,1)]
(1,3): Any[(2,3),(1,4),(1,2)]
(1,4): Any[(2,4),(1,5),(1,3)]
(1,5): Any[(2,5),(1,4)]
(2,1): Any[(3,1),(2,2),(1,1)]
(2,2): Any[(3,2),(2,3),(1,2),(2,1)]
(2,3): Any[(3,3),(2,4),(1,3),(2,2)]
(2,4): Any[(3,4),(2,5),(1,4),(2,3)]
(2,5): Any[(3,5),(1,5),(2,4)]
(3,1): Any[(4,1),(3,2),(2,1)]
(3,2): Any[(4,2),(3,3),(2,2),(3,1)]
(3,3): Any[(4,3),(3,4),(2,3),(3,2)]
(3,4): Any[(4,4),(3,5),(2,4),(3,3)]
(3,5): Any[(4,5),(2,5),(3,4)]
(4,1): Any[(5,1),(4,2),(3,1)]
(4,2): Any[(5,2),(4,3),(3,2),(4,1)]
(4,3): Any[(5,3),(4,4),(3,3),(4,2)]
(4,4): Any[(5,4),(4,5),(3,4),(4,3)]
(4,5): Any[(5,5),(3,5),(4,4)]
(5,1): Any[(5,2),(4,1)]
(5,2): Any[(5,3),(4,2),(5,1)]
(5,3): Any[(5,4),(4,3),(5,2)]
(5,4): Any[(5,5),(4,4),(5,3)]
(5,5): Any[(4,5),(5,4)]
```

Problem Model

In [4]:
```

```

```
using JuMP, Cbc

m = Model(solver=CbcSolver())

# dummy decision variable to make number of toggles odd
@variable(m, t[1:rows,1:columns] >= 0, Int)
# Number of times a cell has been toggled
@variable(m, toggled[1:rows,1:columns])
# Number of times a cell has been clicked.
@variable(m, clicked[1:rows,1:columns], Bin)

# Constraint for number of toggles to be odd. To switch off the light in the end.
@constraint(m, oddToggles[i=1:rows,j=1:columns], toggled[i,j] == 2*t[i,j] + 1)
# Constraint to make number of toggles equal to the sum of clicks on this cell and its
# neighbours
@constraint(m, nbr[i=1:rows,j=1:columns], clicked[i,j] + sum(clicked[i_n,j_n]
                        for (i_n,j_n) in get_nbr(i,j,rows,columns)) == toggled[i,j])

# Objective to minimize the number of clicks.
@objective(m, Min, sum(clicked[i,j] for i in 1:rows for j in 1:columns))

m
```

Out[4]:

$$
\begin{aligned}
\min \quad & clicked_{1,1} + clicked_{1,2} + clicked_{1,3} + clicked_{1,4} + clicked_{1,5} + clicked_{2,1} + clicked_{2,2} + clicked_{2,3} + clicked_{2,4} \\
& + clicked_{3,2} + clicked_{3,3} + clicked_{3,4} + clicked_{3,5} + clicked_{4,1} + clicked_{4,2} + clicked_{4,3} + clicked_{4,4} + clicked_{4} \\
& + clicked_{5,3} + clicked_{5,4} + clicked_{5,5} \\
\text{Subject to} \quad & toggled_{1,1} - 2t_{1,1} = 1 \\
& toggled_{1,2} - 2t_{1,2} = 1 \\
& toggled_{1,3} - 2t_{1,3} = 1 \\
& toggled_{1,4} - 2t_{1,4} = 1 \\
& toggled_{1,5} - 2t_{1,5} = 1 \\
& toggled_{2,1} - 2t_{2,1} = 1 \\
& toggled_{2,2} - 2t_{2,2} = 1 \\
& toggled_{2,3} - 2t_{2,3} = 1 \\
& toggled_{2,4} - 2t_{2,4} = 1 \\
& toggled_{2,5} - 2t_{2,5} = 1 \\
& toggled_{3,1} - 2t_{3,1} = 1 \\
& toggled_{3,2} - 2t_{3,2} = 1 \\
& toggled_{3,3} - 2t_{3,3} = 1 \\
& toggled_{3,4} - 2t_{3,4} = 1 \\
& toggled_{3,5} - 2t_{3,5} = 1 \\
& toggled_{4,1} - 2t_{4,1} = 1 \\
& toggled_{4,2} - 2t_{4,2} = 1 \\
& toggled_{4,3} - 2t_{4,3} = 1 \\
& toggled_{4,4} - 2t_{4,4} = 1 \\
& toggled_{4,5} - 2t_{4,5} = 1 \\
& toggled_{5,1} - 2t_{5,1} = 1 \\
& toggled_{5,2} - 2t_{5,2} = 1 \\
& toggled_{5,3} - 2t_{5,3} = 1 \\
& toggled_{5,4} - 2t_{5,4} = 1 \\
& toggled_{5,5} - 2t_{5,5} = 1 \\
& clicked_{1,1} + clicked_{2,1} + clicked_{1,2} - toggled_{1,1} = 0 \\
& clicked_{1,2} + clicked_{2,2} + clicked_{1,3} + clicked_{1,1} - toggled_{1,2} = 0 \\
& clicked_{1,3} + clicked_{2,3} + clicked_{1,4} + clicked_{1,2} - toggled_{1,3} = 0 \\
& clicked_{1,4} + clicked_{2,4} + clicked_{1,5} + clicked_{1,3} - toggled_{1,4} = 0 \\
& clicked_{1,5} + clicked_{2,5} + clicked_{1,4} - toggled_{1,5} = 0 \\
& clicked_{2,1} + clicked_{3,1} + clicked_{2,2} + clicked_{1,1} - toggled_{2,1} = 0 \\
& clicked_{2,2} + clicked_{3,2} + clicked_{2,3} + clicked_{1,2} + clicked_{2,1} - toggled_{2,2} = 0 \\
& clicked_{2,3} + clicked_{3,3} + clicked_{2,4} + clicked_{1,3} + clicked_{2,2} - toggled_{2,3} = 0 \\
& clicked_{2,4} + clicked_{3,4} + clicked_{2,5} + clicked_{1,4} + clicked_{2,3} - toggled_{2,4} = 0 \\
& clicked_{2,5} + clicked_{3,5} + clicked_{1,5} + clicked_{2,4} - toggled_{2,5} = 0 \\
& clicked_{3,1} + clicked_{4,1} + clicked_{3,2} + clicked_{2,1} - toggled_{3,1} = 0 \\
& clicked_{3,2} + clicked_{4,2} + clicked_{3,3} + clicked_{2,2} + clicked_{3,1} - toggled_{3,2} = 0 \\
& clicked_{3,3} + clicked_{4,3} + clicked_{3,4} + clicked_{2,3} + clicked_{3,2} - toggled_{3,3} = 0 \\
& clicked_{3,4} + clicked_{4,4} + clicked_{3,5} + clicked_{2,4} + clicked_{3,3} - toggled_{3,4} = 0 \\
& clicked_{3,5} + clicked_{4,5} + clicked_{2,5} + clicked_{3,4} - toggled_{3,5} = 0
\end{aligned}
$$

$$clicked_{4,1} + clicked_{5,1} + clicked_{4,2} + clicked_{3,1} - toggled_{4,1} = 0$$
$$clicked_{4,2} + clicked_{5,2} + clicked_{4,3} + clicked_{3,2} + clicked_{4,1} - toggled_{4,2} = 0$$
$$clicked_{4,3} + clicked_{5,3} + clicked_{4,4} + clicked_{3,3} + clicked_{4,2} - toggled_{4,3} = 0$$
$$clicked_{4,4} + clicked_{5,4} + clicked_{4,5} + clicked_{3,4} + clicked_{4,3} - toggled_{4,4} = 0$$
$$clicked_{4,5} + clicked_{5,5} + clicked_{3,5} + clicked_{4,4} - toggled_{4,5} = 0$$
$$clicked_{5,1} + clicked_{5,2} + clicked_{4,1} - toggled_{5,1} = 0$$
$$clicked_{5,2} + clicked_{5,3} + clicked_{4,2} + clicked_{5,1} - toggled_{5,2} = 0$$
$$clicked_{5,3} + clicked_{5,4} + clicked_{4,3} + clicked_{5,2} - toggled_{5,3} = 0$$
$$clicked_{5,4} + clicked_{5,5} + clicked_{4,4} + clicked_{5,3} - toggled_{5,4} = 0$$
$$clicked_{5,5} + clicked_{4,5} + clicked_{5,4} - toggled_{5,5} = 0$$
$$t_{i,j} \geq 0, \in \mathbb{Z}, \quad \forall i \in \{1,2,3,4,5\}, j \in \{1,2,3,4,5\}$$
$$toggled_{i,j} free \quad \forall i \in \{1,2,3,4,5\}, j \in \{1,2,3,4,5\}$$
$$clicked_{\cdots} \in \{0,1\} \quad \forall i \in \{1,2,3,4,5\}, j \in \{1,2,3,4,5\}$$

In [5]:
```
status=solve(m)
println("Status: ",status)
println("ObjectiveValue: ",getobjectivevalue(m))
clickedCells=getvalue(clicked)
println("Clicks")
[println(clickedCells[i,:]) for i in 1:rows]
numToggles=getvalue(toggled)
println("# of Toggles")
[println(numToggles[i,:]) for i in 1:rows];
```

```
Status: Optimal
ObjectiveValue: 15.0
Clicks
[0.0,1.0,1.0,0.0,1.0]
[0.0,1.0,1.0,1.0,0.0]
[0.0,0.0,1.0,1.0,1.0]
[1.0,1.0,0.0,1.0,1.0]
[1.0,1.0,0.0,0.0,0.0]
# of Toggles
[1.0,3.0,3.0,3.0,1.0]
[1.0,3.0,5.0,3.0,3.0]
[1.0,3.0,3.0,5.0,3.0]
[3.0,3.0,3.0,3.0,3.0]
[3.0,3.0,1.0,1.0,1.0]
```