1. Thrift Store [5 pts]

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)	2.50	5.000	2.268	5.670

In [1]:
```
# data
coins  = [0.01, 0.05, 0.10, 0.25]
weights = [2.50, 5.00, 2.268, 5.670]
```

In [2]:
```
using JuMP, Clp, Gurobi, Mosek, GLPK

m = Model(solver = MosekSolver())

@variable(m, x[1:4] >= 0, Int)

@constraint(m, x[1]*coins[1] + x[2]*coins[2] + x[3]*coins[3] + x[4]*coins[

@expression(m, total_weight, x[1]*weights[1] + x[2]*weights[2] + x[3]*weig

@objective(m, Min, total_weight)
```

Out[2]: :Optimal

In [3]:
```
println("The total weight (in grams) is ", getobjectivevalue(m))

for i = 1:4
    println("Quantity of ", coins[i], ": ", getvalue(x[i]))
end
```

```
The total weight (in grams) is 31.546
Quantity of 0.01: 4.0
Quantity of 0.05: 0.0
Quantity of 0.1: 2.0
Quantity of 0.25: 3.0
```

2. Comquat Computers [15 pts]

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

In [1]:
```
# data
prod_caps = [10000, 8000, 9000, 6000]
plant_fixed_cost = [9, 5, 3, 1]
cost_per_computer = [1000, 1700, 2300, 2900]

max_computers_per_year = 20000
price_per_computer = 3500
```

In [24]:
```
using JuMP, Cbc, Gurobi, Mosek, GLPK

m = Model(solver = MosekSolver())

@variable(m, x[1:4] >= 0, Int) # quantities from each plant
@variable(m, z[1:4], Bin) # whether plant is active
@constraint(m, x .<= 10000*z) # if x > 0 then z = 1

@constraint(m, x[1:4] .<= prod_caps[1:4]) # cap plant capacities according to data

@constraint(m, sum(x[1:4]) <= 20000) # sale constraint

@expression(m, max_revenue, sum(x[1:4]) * 3500) # price per computer constraint

@expression(m, total_prod_cost, (cost_per_computer[1]*x[1] + cost_per_computer[2]*x[2] + cost_per_computer[3]*
@expression(m, total_plant_cost, (plant_fixed_cost[1]*z[1] + plant_fixed_cost[2]*z[2] + plant_fixed_cost[3]*z[

@objective(m, Max, max_revenue - (total_plant_cost + total_prod_cost))

solve(m)
```

Out[24]:
```
:Optimal
```

In [25]:
```
for i = 1:4
        println("Plant ", i, " produced ", getvalue(x[i]), " computers")
```

```
Plant 1 produced 10000.0 computers
Plant 2 produced 8000.0 computers
Plant 3 produced 0.0 computers
Plant 4 produced 2000.0 computers
```

In [26]:

```
With this configuration, Comquat Computers will make a profit of $2.56e7
```

3. ABC Investments [15 pts]

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?

In [1]:
```
# data
min_investments = [3, 2, 9, 5, 12, 4]
max_investments = [27, 12, 35, 15, 46, 18]
expected_returns = [13, 9, 17, 10, 22, 12]*0.01 + 1

max_investment = 80
```

In [8]:
```
using JuMP, Cbc, Gurobi, Mosek, GLPK

m = Model(solver = MosekSolver())

@variable(m, x[1:6] >= 0)
@variable(m, z[1:6], Bin)
@constraint(m, sum(x[1:6]) <= max_investment)

@constraint(m, x[1] >= min_investments[1]z[1])
@constraint(m, x[1] <= max_investments[1]z[1])

@constraint(m, x[2] >= min_investments[2]z[2])
@constraint(m, x[2] <= max_investments[2]z[2])

@constraint(m, x[3] >= min_investments[3]z[3])
@constraint(m, x[3] <= max_investments[3]z[3])

@constraint(m, x[4] >= min_investments[4]z[4])
@constraint(m, x[4] <= max_investments[4]z[4])

@constraint(m, x[5] >= min_investments[5]z[5])
@constraint(m, x[5] <= max_investments[5]z[5])
@constraint(m, x[5] <= (x[2] + x[4] + x[6]))

@constraint(m, x[6] >= min_investments[6]z[6])
@constraint(m, x[6] <= max_investments[6]z[6])
@constraint(m, z[3] <= z[6])

@expression(m, total_returns, expected_returns[1]*x[1] + expected_returns[2]*x[2] + expected_returns[3]*x[3] + expected_re

@objective(m, Max, total_returns)

solve(m)
```

Out[8]:
```
:Optimal
```

In [9]:
```
println("ABC should invest as follows:")

for i = 1:6
        println("Option ", i, " with \$", getvalue(x[i]), " million invested")
end

println()
println("This brings the total investment to \$", sum(getvalue(x[1:6])), " million, with an expected return of \$", getobj
```

```
ABC should invest as follows:
Option 1 with $0.0 million invested
Option 2 with $0.0 million invested
Option 3 with $35.0 million invested
Option 4 with $5.0 million invested
Option 5 with $22.5 million invested
Option 6 with $17.5 million invested

This brings the total investment to $80.0 million, with an expected return of $93.5 million
```

4. Lights Out [15 pts]

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).

Hints: The order in which the cells are clicked doesn't matter (think about it!), and there is no need to click any cell more than once.

https://en.wikipedia.org/wiki/Lights_Out_(game (https://en.wikipedia.org/wiki/Lights_Out_(game))

In [104]:
```
board = [0 0 0 0 0 0 0
         0 1 1 1 1 1 0
         0 1 1 1 1 1 0
         0 1 1 1 1 1 0
         0 1 1 1 1 1 0
         0 1 1 1 1 1 0
         0 0 0 0 0 0 0]

using JuMP, Cbc, Gurobi, Mosek, GLPK

m = Model(solver = CbcSolver())

@variable(m, b[1:7, 1:7] >= 0, Int)
@variable(m, z[1:7, 1:7], Bin)

# edges must be 0 constraints
@constraint(m, b[1, :] .== 0)
@constraint(m, b[7, :] .== 0)
@constraint(m, b[:, 1] .== 0)
@constraint(m, b[:, 7] .== 0)

# all cells must be toggled off
for r = 2:6
    for c = 2:6
        @constraint(m, (b[r, c] + b[r-1, c] + b[r+1, c] + b[r, c-1] + b[r, c+1])  == 1)
    end
end

@expression(m, innersum, sum(b[2:6, 2:6]))

@objective(m, Min, innersum)
```

```
WARNING: Not solved to optimality, status: Infeasible
WARNING: Infeasibility ray (Farkas proof) not available
```

Out[104]: :Infeasible