Thrift Store

In [5]:
```
Pkg.add("Cbc")
```

```
INFO: Nothing to be done
```

In [125]:
```
using JuMP
using Cbc
m = Model(solver=CbcSolver())
@variable(m, X[1:4] >= 0, Int)
coin_val = zeros(4)
weight   = zeros(4)

coin_val[1] = 1
coin_val[2] = 5
coin_val[3] = 10
coin_val[4] = 25

weight[1]   = 2.5
weight[2]   = 5.0
weight[3]   = 2.268
weight[4]   = 5.67

@constraint(m, transpose(coin_val) * X .== 99)
@objective(m, Min, sum(weight[i] * X[i] for i = 1:4))
solve(m)

v = getobjectivevalue(m)
opt = getvalue(X)
println("Pennies: ", opt[1])
println("Nickles: ", opt[2])
println("Dimes: ", opt[3])
println("Quarters: ", opt[4])
println("Minimal weight: ", v)
```

```
Pennies: 4.0
Nickles: 0.0
Dimes: 7.000000000000001
Quarters: 1.0
Minimal weight: 31.546
```

Comquat Computers

In [126]:
```
m = Model(solver=CbcSolver())
@variable(m, X[1:4] >= 0, Int)
@variable(m, W[1:4], Bin)

Max            = [10000, 8000, 9000, 6000]
Unit_Cost      = [1000, 1700, 2300, 2900]
Plant_Cost     = [9000000, 5000000, 3000000, 1000000]
Production_Max = 20000
Sell_Price     = 3500

pc = @expression(m, sum(X[i] for i = 1:4))
oc = @expression(m, sum(W[i] * Plant_Cost[i] for i = 1:4))
uc = @expression(m, sum(X[i] * Unit_Cost[i] for i = 1:4))
@constraint(m, pc <= Production_Max)

for i = 1:4
    @constraint(m, X[i] <= Max[i]*W[i])
end

@objective(m, Max, Sell_Price * pc - (oc + uc))
solve(m)
println("Maximum profit: ", getobjectivevalue(m))
x = getvalue(X)
println("Computers produced per plant")
for i = 1:4
    println("Plant ", i, ": ", x[i])
end
```

```
Maximum profit: 2.56e7
Computers produced per plant
Plant 1: 10000.0
Plant 2: 8000.0
Plant 3: 0.0
Plant 4: 2000.0
```

Abc Investments

In [127]:
```
m = Model(solver=CbcSolver())
@variable(m, X[1:6] >= 0)
@variable(m, W[1:6], Bin)
Total_Investment = @expression(m, sum(X[i] for i = 1:6));
```

In [128]:
```
Lower = [3, 2, 9, 5, 12, 4]
Upper = [27, 12, 35, 15, 46, 18]
Investment_Max = 80

@constraint(m, X[2] + X[4] + X[6] >= X[5])
@constraint(m, W[3] <= W[6])
@constraint(m, Total_Investment <= Investment_Max)

for i = 1:6
    @constraint(m, X[i] <= Upper[i]*W[i])
    @constraint(m, X[i] >= Lower[i]*W[i])
end
```

In [129]:
```
Exp_Return = [13, 9, 17, 10, 22, 12]

@objective(m, Max, sum(X[i]*Exp_Return[i]*(1/100) for i = 1:6))
solve(m)
```

Out[129]:
```
:Optimal
```

In [130]:
```
v = getobjectivevalue(m)
x = getvalue(X)
println("Maximum expected return: ", v)
println("Amount invested in each option")
for i = 1:6
    println("Option ", i, ": ", x[i])
end
```

```
Maximum expected return: 13.500000000000002
Amount invested in each option
Option 1: 0.0
Option 2: 0.0
Option 3: 34.99999999999999
Option 4: 5.000000000000001
Option 5: 22.500000000000004
Option 6: 17.500000000000004
```

Lights Out

In [131]:
```
#Generate a random beginning grid
gen  = zeros(7,7)
Grid = zeros(5,5)
for i = 2:6
    for j = 2:6
        gen[i,j] = abs(rand(Int) % 31)  # rand button presses on a device. should be invertible
    end
end

for i = 1:5
    for j = 1:5
        l = i + 1
        k = j + 1
        Grid[i,j] = (gen[l,k] + gen[l-1,k] + gen[l+1,k] + gen[l,k-1] + gen[l,k+1]) % 2
    end
end
Grid
```

Out[131]:
```
5×5 Array{Float64,2}:
 0.0  0.0  1.0  1.0  0.0
 0.0  1.0  0.0  0.0  0.0
 1.0  1.0  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
```

In [132]:
```
m = Model(solver=CbcSolver())
@variable(m, press[1:7,1:7], Bin)     #1 if we press the button located at i,j. else it is 0.
@variable(m, W[1:5,1:5,1:2], Bin)     #forces each square to be a sum of on,off pairs
@variable(m, out[1:5,1:5] >= 0, Int)
```

Out[132]: $out_{i,j} \geq 0, \in \mathbb{Z}, \quad \forall i \in \{1,2,3,4,5\}, j \in \{1,2,3,4,5\}$

In [133]:
```
for i = 1:7
    @constraint(m, press[1,i] == 0)
    @constraint(m, press[7,i] == 0)
end
for i = 2:6
    @constraint(m, press[i,1] == 0)
    @constraint(m, press[i,7] == 0)
end
```

In [134]:
```
for i = 1:5
    for j = 1:5
        l = i + 1
        k = j + 1
        @constraint(m, out[i,j] == Grid[i,j] + press[l,k] + press[l-1,k] + press[l+1,k] + press[l,k-1] +
press[l,k+1])
        @constraint(m, out[i,j] == 2*W[i,j,1] + 4*W[i,j,2])
        @constraint(m, 0 <= out[i,j] <= 6)  #6 is max number of flips possible
    end
end
```

In [135]:
```
@objective(m, Min, sum(press[i,j] for i = 2:6, j = 2:6))
solve(m)
```

Out[135]:
```
:Optimal
```

In [136]:
```
println("Below is the minimal amount of button presses which will solve the puzzle.")
println("The outer rows and columns are padding.")
getvalue(press)
```

```
Below is the minimal amount of button presses which will solve the puzzle.
The outer rows and columns are padding.
```

Out[136]:
```
7×7 Array{Float64,2}:
 0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  1.0  0.0  0.0  0.0
 0.0  0.0  1.0  0.0  0.0  0.0  0.0
 0.0  1.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  1.0  1.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  1.0  1.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0
```