In [1]:

```
from mip import Model, BINARY, minimize, xsum
import numpy as np
```

Objective: Minimize distance travelled

Contraints: Must travel to and from each city only once

	City 1	City 2	City 3	City 4
City 1	0	10	15	20
City 2	10	0	35	25
City 3	15	35	0	30
City 4	20	25	30	0

We have a table containing the distances between each city. The table is symmetric since you can travel to each city along the same path in either direction. Think of distance travelled overall as a cost. This translates well to optimization as we will try to minimize this cost. Since we only want to visit each city once, we want to enter and leave each city only once, ending up in the same final spot.

Cost

c_{ij}: Distance from city $\bf{i}$ to city $\bf{j}$

$c_{ij} = c_{ji}$: Distance is symmetric

Route variable

$x_{ij} = 1$: If the salesperson travels from city $\bf{i}$ to city $\bf{j}$

$x_{ij} = 0$: Otherwise (i,j = 1,...,N and i $\neq$ j)

Auxiliary variable

u_{i}: Eliminates subtours (i = 1,...,N)

Minimize cost

$\sum_{i=1}^{N} \sum_{j=1}^{N} c_{ij}x_{ij}$

Subject to constraints

$\sum_{i=1}^{N} x_{ij} = 1, j = 1$,...,$N$

$\sum_{j=1}^{N} x_{ij} = 1, i = 1$,...,$N$

$u_{i} - u_{j} + N * x_{ij} \leq N - 1; i \neq j; i,j = 2$,...,$N$

$x_{ij} \in \{0, 1\}; i,j = 1$,...,$N$

$u_{i} \geq 0, i = 1$,...,N

In [2]:

```
city = ['City 1', 'City 2', 'City 3', 'City 4']

#create an arbitrarily large cost for staying within same city
M = 1000

#cost matrix
c = [[M, 10, 15, 20],
     [10, M, 35, 25],
     [15, 35, M, 30],
     [20, 25, 30, M]]

N = len(c)
node = set((range(0,N)))
options = set((range(1,N)))

TSP = Model(sense = 'Minimize', solver_name='CBC')

x = [[TSP.add_var(var_type = BINARY, lb = 0.0, ub = 1.0) for j in node] for i in node]
u = [TSP.add_var() for i in node]

#Optimization
#minimize cost
TSP.objective = minimize(xsum(c[i][j]*x[i][j] for i in node for j in node))

#Constraints
#leave only once
for i in node:
    TSP.add_constr( xsum(x[i][j] for j in node) == 1 )
#enter only once
for j in node:
    TSP.add_constr( xsum(x[i][j] for i in node) == 1 )
#no subtours
for i in options:
    for j in options:
        if i != j:
            TSP.add_constr( u[i] - u[j] + N * x[i][j] <= N - 1 )

TSP.optimize()

if TSP.num_solutions > 0:
    #print distance and starting city
    print('Route found with total distance {}. Order: {}'.format(TSP.objective_value, city
[0]))
    #print city order
    cNum = 0
    while True:
        cNum = [i for i in node if x[cNum][i].x > 0][0]
        print('to {}'.format(city[cNum]))
        if cNum == 0:
            break
if TSP.num_solutions == 0:
    print('No solutions were found.')
```

```
Route found with total distance 80.0. Order: City 1
to City 3
to City 4
to City 2
to City 1
```

```
In [ ]:
```