Auror Location Problem

The wizarding world has been divided into eight districts.
The time (in seconds) required to travel from one district to another via the floo network is shown below:

	1	2	3	4	5	6	7	8
1		3	4	6	8	9	8	10
2	3		5	4	8	6	12	9
3	4	5		2	2	3	5	7
4	6	4	2		3	2	5	4
5	8	8	2	3		2	2	4
6	9	6	3	2	2		3	2
7	8	12	5	5	2	3		2
8	10	9	7	4	4	2	2	

The wizard population of each district (in thousands) is: district 1, 40; 2, 30; 3, 35; 4, 20; 5, 15; 6, 50; 7, 45; 8, 60. The Ministry of Magic has declared that there will be n auror locations.

Determine the locations of the aurors that maximize the number of people who live within two seconds of an auror. Do this for $n = 1, 2, 3, 4$.

In [46]:
```
import sys
import pandas as pd
import numpy as np

from gamspy import (
    Container,Set,Alias,Parameter,Variable,Equation,Model,Problem,Sense,Opti
    Domain,Number,Sum,Product,Smax,Smin,Ord,Card,SpecialValues,
    ModelStatus,SolveStatus,
)

# any optimum within <1 of the true optimum must BE the true optimum!
options = Options(equation_listing_limit=0,absolute_optimality_gap=0.999)

m = Container(options=options)
```

In [50]:
```
i = Set(m,'i',records=[ind+1 for ind in range(8)])
n = Parameter(m,'n',description='number of stations to build')

s1 = Set(m, 's1', records=[ind + 1 for ind in range(8)]) #make sure sets use
s2 = Set(m, 's2', records=[ind + 1 for ind in range(8)])
```

```
people = Parameter(m, 'people', domain=[s1], records=np.array([40, 30, 35, 2

time = Parameter(m, 'time', domain=[s1, s2], records=pd.DataFrame(data=np.ar
```

```
limit = Parameter(m, 'limit', domain=[s1, s2])
limit[s1, s2] = Number(1).where[time[s1, s2] <= 2]

x = Variable(m, 'x', 'binary', domain=[s1])
y = Variable(m, 'y', 'binary', domain=[s2])

wiz = Equation(m, 'wiz')
wiz[:] = Sum(s2, y[s2]) <= n

bound = Equation(m, 'bound', domain=[s1])
bound[s1] = Sum(s2, limit[s1, s2] * y[s2]) >= x[s1]

auror = m.addModel('auror',
    equations=m.getEquations(),
    problem=Problem.MIP,
    sense=Sense.MAX,
    objective=Sum(s1, x[s1] * people[s1]),
)

# PUT YOUR CODE HERE
```

In [48]:
```
# tobuild is a dataframe with columns for what is built, where people is a c
tobuild = pd.DataFrame(index=i.toList(),columns=['1','2','3','4'])
people = {}
```

In [49]:
```
n.setRecords(1)
auror.solve(options=options)
# EXTRACT YOUR SOLUTION INTO tobuild and people
tobuild['1'] = y.toList()
people['1'] = auror.objective_value

n.setRecords(2)
auror.solve(options=options)
# EXTRACT YOUR SOLUTION INTO tobuild and people
tobuild['2'] = y.toList()
people['2'] = auror.objective_value

n.setRecords(3)
auror.solve(options=options)
# EXTRACT YOUR SOLUTION INTO tobuild and people
tobuild['3'] = y.toList()
```

```
people['3'] = auror.objective_value

n.setRecords(4)
auror.solve(options=options)
# EXTRACT YOUR SOLUTION INTO tobuild and people
tobuild['4'] = y.toList()
people['4'] = auror.objective_value

display(tobuild,people)
```

	1	2	3	4
1	(1, 0.0)	(1, 0.0)	(1, 1.0)	(1, 1.0)
2	(2, 0.0)	(2, 0.0)	(2, 0.0)	(2, 1.0)
3	(3, 0.0)	(3, 0.0)	(3, 0.0)	(3, 0.0)
4	(4, 0.0)	(4, 0.0)	(4, 0.0)	(4, 0.0)
5	(5, 0.0)	(5, 1.0)	(5, 1.0)	(5, 1.0)
6	(6, 0.0)	(6, 1.0)	(6, 1.0)	(6, 1.0)
7	(7, 0.0)	(7, 0.0)	(7, 0.0)	(7, 0.0)
8	(8, 1.0)	(8, 0.0)	(8, 0.0)	(8, 0.0)

```
{'1': 155.0, '2': 225.0, '3': 265.0, '4': 295.0}
```

In [ ]: