```
import matplotlib.pyplot as plt
import networkx as nx
import random

rank = 10
```

Define Methods

Generate base graph and scenarios

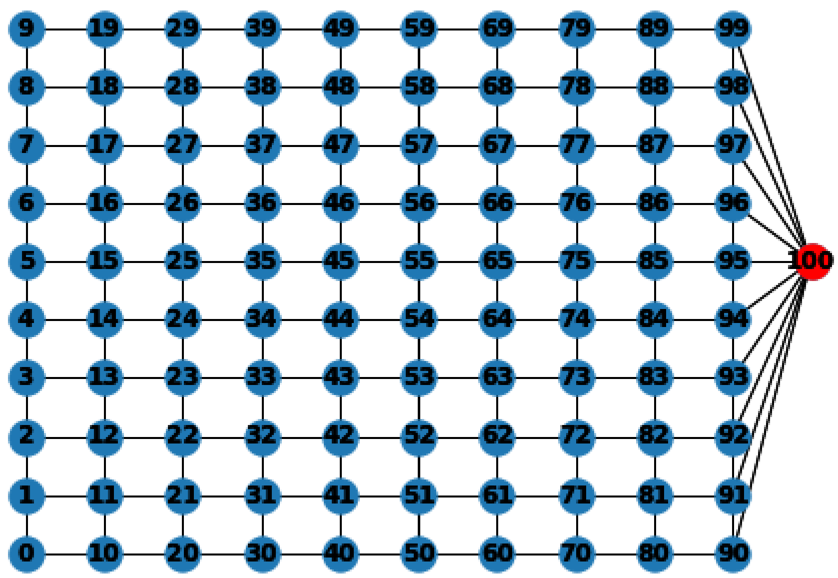
```
import numpy as np

def generateGraphAndTarget(rank):
  G = nx.Graph()

  #food truck
  target = rank ** 2
  G.add_node(target)

  # open space desks
  for i in range(rank):
    for j in range(rank):
      n = i * rank + j
      G.add_node(n)
      if i > 0: G.add_edge(n - rank, n)
      if j > 0: G.add_edge(n - 1, n)
      if i == rank - 1: G.add_edge(n, target)

  return G, target


def getSource(rank):
  return random.choice(range(rank))

def getOccupiedDesks(G, source, target, p):
  available_desks = set(G.nodes).difference(set([source, target]))
  return random.sample(available_desks, int(p * len(available_desks)))

def removeDesks(G, desks):
  for n in desks:
    G.remove_node(n)

def removeOccupiedDesks(G, source, target, p):
  occupied_desks = getOccupiedDesks(G, source, target, p)
  removeDesks(G, occupied_desks)

def generateScenario(G, target, p):
  # todo initialise random seed
  source = getSource(rank)
  removeOccupiedDesks(G, source, target, p)
  return source
```

```
def canHazHamburger(G, source, target):
   return nx.has_path(G, source, target)
```

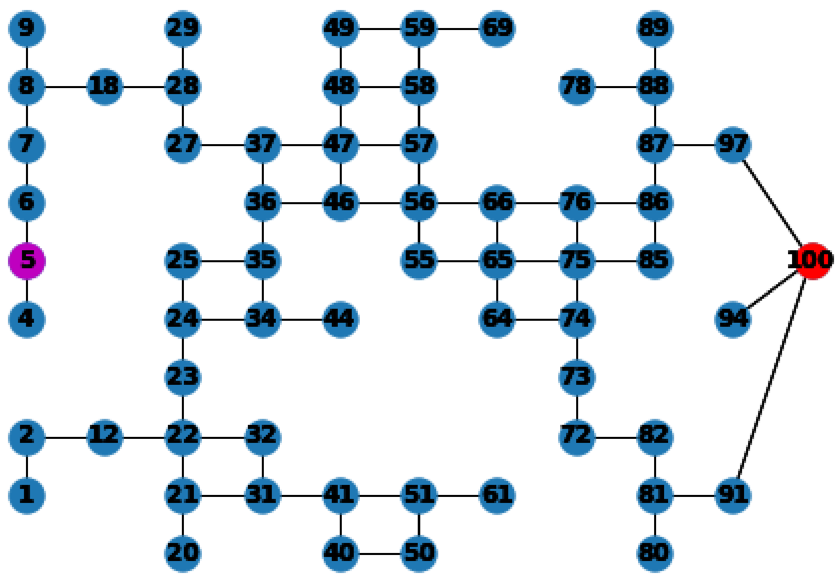
Single Run

Step through a single scenario and visualise

```
def grid_pos(rank):
  pos = {}
  for i in range(rank):
    for j in range(rank):
      n = i * rank + j
      pos[n] = np.array([(2 * i / rank - 1), (2 * j / rank - 1)])
  pos[rank ** 2] = np.array([1.0, 0])
  return pos

def drawGraph(G, pos, target, source=None, path=None):
  nx.draw(G, pos=g_pos, with_labels=True, font_weight='bold')
  nx.draw(G, pos=g_pos, nodelist=[target], node_color='r', with_labels=True, font
  if source != None:
    nx.draw(G, pos=g_pos, nodelist=[source], node_color='m', with_labels=True, fo
  if path:
    nx.draw(G, pos=g_pos, nodelist=path[1:-1], node_color='g', with_labels=True,


G, target = generateGraphAndTarget(rank)
g_pos = grid_pos(rank)

drawGraph(G, g_pos, target)
plt.show()
```



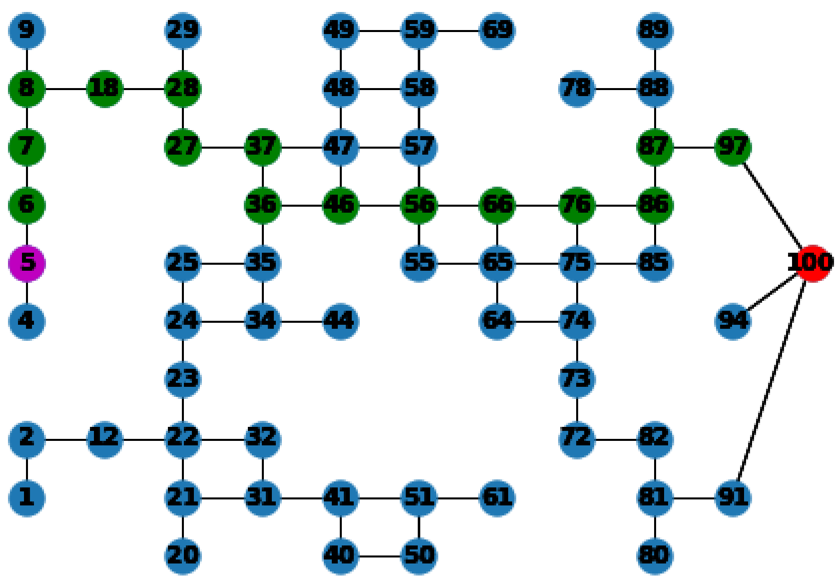

```
p = 0.4
source = generateScenario(G, target, p)
drawGraph(G, g_pos, target, source=source)
plt.show()
```




```
iCanHaz = canHazHamburger(G, source, target)
print(iCanHaz)
```

```
True
```

```
if iCanHaz:
  short_path = nx.shortest_path(G, source, target)
  drawGraph(G, g_pos, target, source=source, path=short_path)
  plt.show()
else:
  print('Can NOT Haz')
```



Repeated Trials

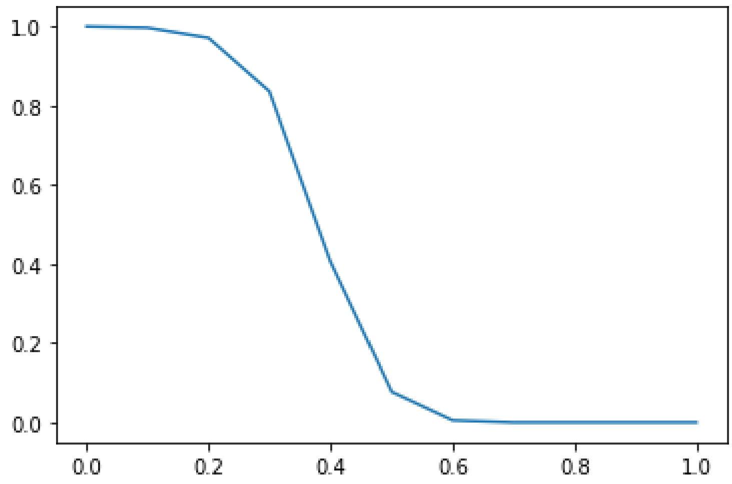
Run test for values of p

```
trials = 10000
```

```
ps = [x / 10.0 for x in range(11)]
ps.reverse()
success = []

for p in ps:
  s = 0
  for i in range(trials):
    G, target = generateGraphAndTarget(rank)
    source = generateScenario(G, target, p)
    s = s + canHazHamburger(G, source, target)
  success.append(s)

success_rate = [s/trials for s in success]

# todo prettier formatting
print(f"Number of samples for each p: {trials}")
for i, p in enumerate(ps):
  print(f"{p} {success_rate[i]}")
```

```
Number of samples for each p: 10000
1.0 0.0
0.9 0.0
0.8 0.0
0.7 0.0
0.6 0.0053
0.5 0.0773
0.4 0.4065
0.3 0.8358
0.2 0.9712
0.1 0.9967
0.0 1.0
```

```
#todo labels
plt.plot(ps, success_rate)
plt.show()
```



Statistical Analysis

How many different graphs could we produce for each value of p; do we have the right number of t results?

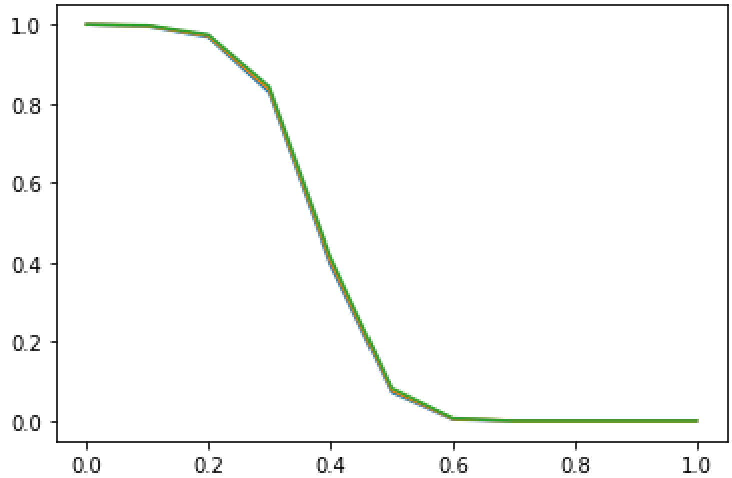
```
import operator as op
from functools import reduce

def ncr(n, r):
    r = min(r, n-r)
    numer = reduce(op.mul, range(n, n-r, -1), 1)
    denom = reduce(op.mul, range(1, r+1), 1)
    return numer / denom


desks_avail = rank ** 2 - 1
combns = [ncr(desks_avail, int(p * desks_avail)) for p in ps]


from statsmodels.stats.proportion import proportion_confint
#todo np.array version
confints = [proportion_confint(s, trials) for s in success]
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: Futur
  import pandas.util.testing as tm
```

```
import pandas as pd

stat_data = {'p': ps,
             'combinations': combns,
             'trials': [trials] * len(ps),
             'success': success,
             'rate': success_rate,
             '95% confidence interval': confints}

df = pd.DataFrame(stat_data)
df.head(11)
```

	p	combinations	trials	success	rate	95% confidence i
0	1.0	1.000000e+00	10000	0	0.0000	
1	0.9	1.557928e+13	10000	0	0.0000	
2	0.8	4.287867e+20	10000	0	0.0000	
3	0.7	2.056064e+25	10000	0	0.0000	
4	0.6	8.247740e+27	10000	53	0.0053	(0.0038769109243829616, 0.006723089075
5	0.5	5.044567e+28	10000	773	0.0773	(0.07206558564533766, 0.0825344143
6	0.4	5.498494e+27	10000	4065	0.4065	(0.3968730497466834, 0.4161269502
7	0.3	8.811702e+24	10000	8358	0.8358	(0.8285391769849445, 0.843060823
8	0.2	1.071967e+20	10000	9712	0.9712	(0.9679220775721235, 0.974477922
9	0.1	1.731031e+12	10000	9967	0.9967	(0.9955759457031916, 0.997824054
10	0.0	1.000000e+00	10000	10000	1.0000	

```
#todo labels
plt.plot(ps, [c[0] for c in confints])
plt.plot(ps, success_rate)
plt.plot(ps, [c[1] for c in confints])
plt.show()
```




```
#todo labels
plt.plot(ps, [c[1] - c[0] for c in confints])
plt.show()
```