# CT5132/CT5148 Lab Week 8

James McDermott

## Small exercises

Suppose we have a **graph** represented as a dictionary where each node points to a single *parent* or *predecessor*, e.g. (you can paste this into your IPython):

```
G = {
'd': 'c',
'c': 'f',
'a': 's',
'f': 'g',
'e': 'g',
'b': 's',
'g': 'b',
}
```

1. Draw this on paper.

Suppose we are at some node, and we want to traverse to the root `s`.

```
x = 'c'
root = 's'
```

2. Write a small snippet of code to do this, printing out all nodes encountered on the way.
3. In the above graph, let's say that each node represents a sub-task to be carried out, and `s` is the ultimate goal. But to do `s`, we need to first do `a` and `b`, according to the links shown above, and so on. Write out a valid **topological sort** of these nodes.
4. Write out an **algorithm** for topological sort and convince yourself it's correct.
5. Find the NetworkX implementation of topological sort and run it on this graph. Take care of ordering, i.e. check whether dependencies are forward (`a` depends on `s`) or backward (`s` depends on `a`).
6. Suppose we add an edge (`c`, `a`). What happens?

## What was the most important family in Renaissance Florence?

This is a short demo of **centrality** in `networkx`.

Suppose we are given the data in this format (`florentine_families.csv`):

```
Acciaiuoli,Medici
Medici,Barbadori
Medici,Ridolfi
Medici,Tornabuoni
```

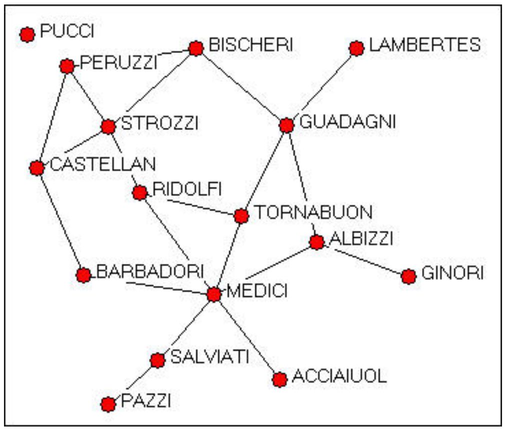

Figure 1: This graph represents the **marriages** between important families in Renaissance Florence (Padgett, figure from Hanneman and Riddle):

```
Medici,Albizzi
Medici,Salviati
Castellani,Peruzzi
Castellani,Strozzi
Castellani,Barbadori
Peruzzi,Strozzi
Peruzzi,Bischeri
Strozzi,Ridolfi
Strozzi,Bischeri
Ridolfi,Tornabuoni
Tornabuoni,Guadagni
Albizzi,Ginori
Albizzi,Guadagni
Salviati,Pazzi
Bischeri,Guadagni
Guadagni,Lamberteschi
```

We can use the following code to read it and create the undirected graph:

```python
import networkx as nx
import matplotlib.pyplot as plt
G = nx.Graph() # empty graph, undirected
filename = "../data/florentine_families.csv"
f = open(filename)
for line in f:
    n1, n2 = line.strip().split(",") # strip() removes \n
    G.add_edge(n1, n2)
```

We might like to create an image also:

```
nx.draw_networkx(G)
plt.savefig("florence_nx.pdf")
```

Now, let's use NetworkX to calculate the **importance** of each node. There are several methods, including `degree_centrality`, `betweenness_centrality`, `eigenvector_centrality`. Try these out and compare their results. Can you **interpret** what each method is looking for? Which family is most important in each case?