Generating a power set

- Given a set of elements, would like to find set of all subsets
  - [1, 2, 3]
  - Leads to [], [1], [2], [3], [1, 2], [1, 3], [2, 3], [1,2,3]
- To find, can use recursive approach:
  - Find power set of all but first element
  - Then copy each element of that set, with first element added
  - Combine both sets as answer

Powerset

```
def powerSet(elts):
    if len(elts) == 0:
        return [[]]
    else:
        smaller = powerSet(elts[1:])
        elt = [elts[0]]
        withElt = []
        for s in smaller:
            withElt.append(s + elt)
        allofthem = smaller + withElt
        return allofthem
```

Finding cliques

- Generate power set of nodes – gives set of all possible subgraphs
- Test each one to see if complete (i.e., all nodes connected)
- Keep track of largest clique found

Power Graph

```
def powerGraph(gr):
    nodes = gr.nodes
    nodesList = []
    for elt in nodes:
        nodesList.append(elt)
    pSet = powerSet(nodesList)
    return pSet
```

Complete graphs

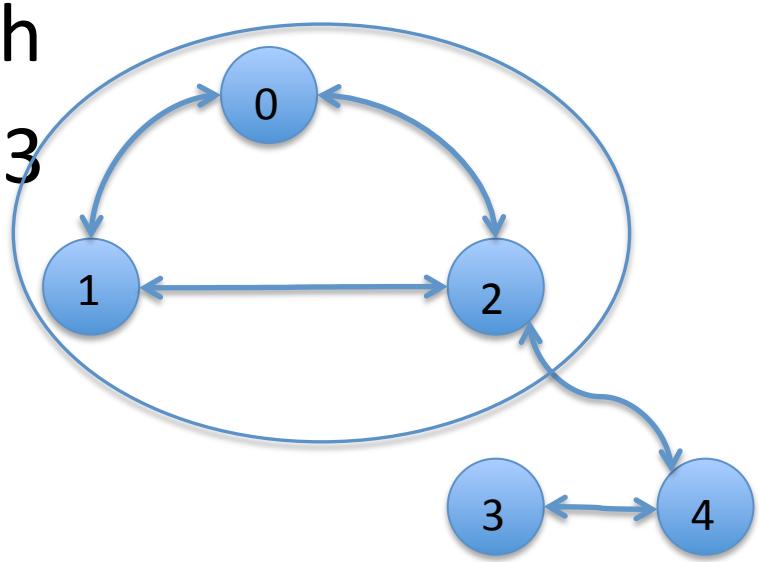
```
def allConnected(gr,candidate):
    for n in candidate:
        for m in candidate:
            if not n == m:
                if n not in gr.childrenOf(m):
                    return False
    return True
```

Max Clique implementation

```
def maxClique(gr):
    candidates = powerGraph(gr)
    keepEm = []
    for candidate in candidates:
        if allConnected(gr, candidate):
            keepEm.append(candidate)
    bestLength = 0
    bestSoln = None
    for test in keepEm:
        if len(test) > bestLength:
            bestLength = len(test)
            bestSoln = test
    return bestSoln
```

An example

- Simple example of graph
- Largest Clique is of size 3



Graphs

- Useful data structure for capturing relationships between objects
- Optimization problems can often be cast as graph search problems
  - Depth first and breadth first searches are common ways to find optimal paths through graph