Graph elements

```
class Node(object):
    def __init__(self, name):
        self.name = str(name)
    def getName(self):
        return self.name
    def __str__(self):
        return self.name

class Edge(object):
    def __init__(self, src, dest):
        self.src = src
        self.dest = dest
    def getSource(self):
        return self.src
    def getDestination(self):
        return self.dest
    def __str__(self):
        return str(self.src) + '->' + str(self.dest)
```

Graph elements

```
class WeightedEdge(Edge):
    def __init__(self, src, dest, weight = 1.0):
        self.src = src
        self.dest = dest
        self.weight = weight
    def getWeight(self):
        return self.weight
    def __str__(self):
        return str(self.src) + '->(‘\
            + str(self.weight) + ')'\
            + str(self.dest)
```

Graph definition

```
class Digraph(object):
    def __init__(self):
        self.nodes = set([])
        self.edges = {}
    def addNode(self, node):
        if node in self.nodes:
            raise ValueError('Duplicate node')
        else:
            self.nodes.add(node)
            self.edges[node] = []
    def addEdge(self, edge):
        src = edge.getSource()
        dest = edge.getDestination()
        if not(src in self.nodes and dest in self.nodes):
            raise ValueError('Node not in graph')
        self.edges[src].append(dest)
    # more later
```

Graph definition

```
class Digraph(object):
    # more above
    def childrenOf(self, node):
        return self.edges[node]
    def hasNode(self, node):
        return node in self.nodes
    def __str__(self):
        res = ''
        for k in self.edges:
            for d in self.edges[k]:
                res = res + str(k) + '->' + str(d) + '\n'
        return res[:-1]

class Graph(Digraph):
    def addEdge(self, edge):
        Digraph.addEdge(self, edge)
        rev = Edge(edge.getDestination(), edge.getSource())
        Digraph.addEdge(self, rev)
```

Graph optimization problems

- **Shortest path**. For some pair of nodes, N1 and N2, find the shortest sequence of edges <sn, dn> (source node and destination node), such that
  - The source node in the first edge is N1
  - The destination node of the last edge is N2
  - For all a and b, if eb follows ea in the sequence, the source node of eb is the destination node of ea.
- **Shortest weighted path**. Like the shortest path, except instead of choosing the shortest sequence of edges that connects two nodes, define a function on the weights of the edges in the sequence (e.g., their sum) and minimize that value.
- **Cliques**. Find a set of nodes such that there is a path (or often a path with a maximum length) in the graph between each pair of nodes in the set.
- **Min cut**. Given two sets of nodes in a graph, a **cut** is a set of edges whose removal eliminates all paths from every node in one set to each node in the other. The minimum cut is the smallest set of edges whose removal accomplishes this.

Example

```
def testSP():
    nodes = []
    for name in range(6):
        nodes.append(Node(str(name)))
    g = Digraph()
    for n in nodes:
        g.addNode(n)
    g.addEdge(Edge(nodes[0],nodes[1]))
    g.addEdge(Edge(nodes[1],nodes[2]))
    g.addEdge(Edge(nodes[2],nodes[3]))
    g.addEdge(Edge(nodes[2],nodes[4]))
    g.addEdge(Edge(nodes[3],nodes[4]))
    g.addEdge(Edge(nodes[3],nodes[5]))
    g.addEdge(Edge(nodes[0],nodes[2]))
    g.addEdge(Edge(nodes[1],nodes[0]))
    g.addEdge(Edge(nodes[3],nodes[1]))
    g.addEdge(Edge(nodes[4],nodes[0]))
    sp = DFS(g, nodes[0], nodes[5])
    print 'Shortest path found by DFS:', printPath(sp)
```