

**6.00 Handout, Lecture 16B**  
**(Not intended to make sense outside of lecture)**

```
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)

class Digraph(object):
    def __init__(self):
        self.nodes = []
        self.edges = {}
    def addNode(self, node):
        if node in self.nodes:
            raise ValueError('Duplicate node')
        else:
            self.nodes.append(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)
    def childrenOf(self, node):
        return self.edges[node]
    def hasNode(self, node):
        return node in self.nodes
    def hasEdge(self, edge):
        if edge.getSource() not in self.nodes:
            return False
        return edge.getDestination() in self.edges[edge.getSource()]
    def numNodes(self):
        return len(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)
```

```

def bfs(g, start, end):
    """Warning: has a problem"""
    if not g.hasNode(start):
        raise ValueError('Node not in graph')
    paths = {}
    paths[0] = [start]
    for depth in range(1, g.numNodes()):
        paths[depth] = []
        for path in paths[depth - 1]:
            for child in g.childrenOf(path[-1]):
                if child == end:
                    print paths
                    return path + child
                if not child in path:
                    paths[depth].append(path + child)
    print paths
    return None

def dfTraversal(g, start, reachable = []):
    reachable = reachable + [start]
    for node in g.childrenOf(start):
        if not node in reachable:
            reachable = dfTraversal(g, node, reachable)
    return reachable

g = Digraph()
for c in 'abcdefg':
    g.addNode(c)
g.addEdge(Edge('a', 'b'))
g.addEdge(Edge('a', 'c'))
g.addEdge(Edge('c', 'b'))
g.addEdge(Edge('b', 'c'))
g.addEdge(Edge('b', 'e'))
g.addEdge(Edge('d', 'f'))
print g
print bfs(g, 'a', 'e')
print bfs(g, 'a', 'f')
print dfTraversal(g, 'a')

```