In [2]:

```python
class Node(object):
    """This class represents a node in a graph."""

    def __init__(self, label: str=None):
        """
        Initialize a new node.

        Args:
            label: the string identifier for the node
        """
        self.label = label
        self.children = []

    def __lt__(self,other):
        """
        Perform the less than operation (self < other).

        Args:
            other: the other Node to compare to
        """
        return (self.label < other.label)

    def __gt__(self,other):
        """
        Perform the greater than operation (self > other).

        Args:
            other: the other Node to compare to
        """
        return (self.label > other.label)

    def __repr__(self):
        """Return a string form of this node."""
        return '{} -> {}'.format(self.label, self.children)

    def add_child(self, node, cost=1):
        """
        Add a child node to this node.

        Args:
            node: the node to add to the children
            cost: the cost of the edge (default 1)
        """
        if type(node) is list:
            [self.add_child(sub_node) for sub_node in node]
            return
        edge = Edge(self, node, cost)
        self.children.append(edge)


class Edge(object):
    """This class represents an edge in a graph."""

    def __init__(self, source: Node, destination: Node, cost: int=1, bidirectiona
        """
        Initialize a new edge.
        Args:
            source: the source of the edge
            destination: the destination of the edge
            cost: the cost of the edge (default 1)
            bidirectional: whether source is accessible (default False)
        """
        self.source = source
        self.destination = destination
        self.cost = cost
        self.bidirectional = bidirectional

    def __repr__(self):
        """Return a string form of this edge."""
        return '{}: {}'.format(self.cost, self.destination.label)
```

In [3]:

```

A = Node('A')
B = Node('B')
C = Node('C')
D = Node('D')
E = Node('E')
F = Node('F')
G = Node('G')

A.add_child([B, C, E])
B.add_child([A, D, F])
C.add_child([G, A])
D.add_child(B)
E.add_child([F, A])
F.add_child([E, B])
G.add_child(C)
```

In [4]:

```
_ = [print(node) for node in [A, B, C, D, E, F, G]]
```

```
A -> [1: B, 1: C, 1: E]
B -> [1: A, 1: D, 1: F]
C -> [1: G, 1: A]
D -> [1: B]
E -> [1: F, 1: A]
F -> [1: E, 1: B]
G -> [1: C]
```

In [5]:

```
def iddfs(root: Node, goal: str, maximum_depth: int=10):
    """
    Return the IDDFS path from the root node to the node with the goal label.

    Args:
        root: the node to start at
        goal: the label of the goal node
        maximum_depth: the maximum depth to search

    Returns: a list with the nodes from root to goal

    Raises: value error if the goal isn't in the graph
    """
    for depth in range(0, maximum_depth):
        result = _dls([root], goal, depth)
        if result is None:
            continue
        return result

    raise ValueError('goal not in graph with depth {}'.format(maximum_depth))

def _dls(path: list, goal: str, depth: int):
    """
    Return the depth limited search path from a subpath to the goal.

    Args:
        path: the current path of Nodes being taken
        goal: the label of the goal node
        depth: the depth in the graph to search

    Returns: the path if it exists, none otherwise
    """
    current = path[-1]
    if current.label == goal:
        return path
    if depth <= 0:
        return None
    for edge in current.children:
        new_path = list(path)
        new_path.append(edge.destination)
        result = _dls(new_path, goal, depth - 1)
        if result is not None:
            return result
```

In [6]:

```
iddfs(D, 'G')
```

Out[6]:

```
[D -> [1: B],
 B -> [1: A, 1: D, 1: F],
 A -> [1: B, 1: C, 1: E],
 C -> [1: G, 1: A],
 G -> [1: C]]
```

In [7]:

```
iddfs(A, 'not a real goal node')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-7-0915aa7afb64> in <module>()
----> 1 iddfs(A, 'not a real goal node')

<ipython-input-5-d00ebd6ff194> in iddfs(root, goal, maximum_depth)
     18         return result
     19 
---> 20     raise ValueError('goal not in graph with depth {}'.format(maximum_d
epth))
     21 
     22 def _dls(path: list, goal: str, depth: int):

ValueError: goal not in graph with depth 10
```

In [ ]: