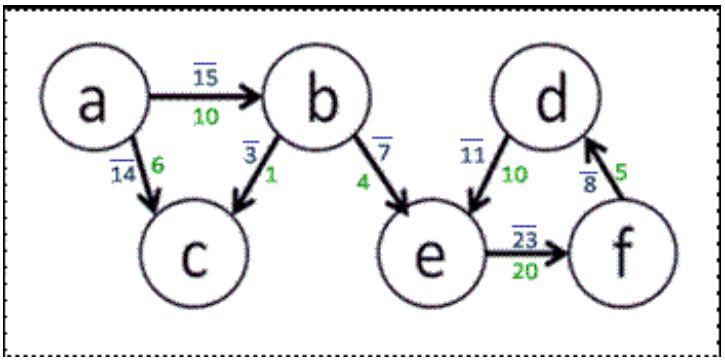
Problem 1 - Creating the Data Structure Representation

In this problem set, we are dealing with edges that have different weights. In the figure below, the blue numbers with a bar above them show the cost of traversing an edge in terms of total distance traveled, while the green numbers show the cost of traversing an edge in terms of distance spent outdoors. Note that the distance spent outdoors for a single edge is always less than or equal to the total distance it takes to traverse that edge. Now the cost of going from "a" to "b" to "e" is a total distance traveled of 22 meters, where 14 of those meters are spent outdoors. These weights are important when comparing multiple paths because you want to look at the weights associated with the edges in the path instead of just the number of edges traversed.

In `graph.py`, you'll find the `Digraph`, `Node`, and `Edge` classes, which do not store information about weights associated with each edge.



Extend the classes so that it fits our case of a weighted graph. Think about how you can modify the classes to store the weights shown above. Make modifications directly in `graph.py`. **We highly recommend that you read through the entire problem set before settling on a particular implementation and representation of nodes and edges.**

Hint: Creating subclasses

Subclass the provided classes to add your own functionality to the new classes. Deciding what representation to use in order to build up the graph is the most challenging part of the problem set, so think through the problem carefully. As a start, `WeightedEdge` should be a subclass of `Edge`, and `WeightedGraph` should be a subclass of `Digraph`

Define a `WeightedDigraph` class to represent your graph. You will also need to define a `WeightedEdge` class to represent the edges of your graph. Be sure to use subclassing and inheritance.

With your `WeightedDigraph` implementation, you should be able to replicate the following transcript, which begins to model the above graph:

```
>>> g = WeightedDigraph()
>>> na = Node('a')
>>> nb = Node('b')
>>> nc = Node('c')
>>> g.addNode(na)
>>> g.addNode(nb)
>>> g.addNode(nc)
>>> e1 = WeightedEdge(na, nb, 15, 10)
>>> print e1
a->b (15, 10)
>>> print e1.getTotalDistance()
15
>>> print e1.getOutdoorDistance()
10
>>> e2 = WeightedEdge(na, nc, 14, 6)
```

```
>>> e3 = WeightedEdge(nb, nc, 3, 1)
>>> print e2
a->c (14, 6)
>>> print e3
b->c (3, 1)
>>> g.addEdge(e1)
>>> g.addEdge(e2)
>>> g.addEdge(e3)
>>> print g
a->b (15.0, 10.0)
a->c (14.0, 6.0)
b->c (3.0, 1.0)
```

Hint: Which class methods?

From the transcript above, you can see which methods should be implemented.

The `WeightedEdge` class will have:

- `__init__(self, src, dest, weight1, weight2)`
- `getTotalDistance(self)`
- `getOutdoorDistance(self)`
- `__str__(self)`

The `WeightedDigraph` class will have:

- `__init__(self)`
- `addEdge(self,edge)`
- `childrenOf(self,node)`
- `__str__(self)`

Common mistakes and (hopefully) helpful implementation tips

How to store `WeightedDigraph` edges?

- source_node:[[dest_node, (total_dist, outdoor_dist)], [dest_node, (total_dist, outdoor_dist)]]
- In the following example, `{a: [ [b,(2,1)], [c,(3,2)]], b: [[c,(4,2)]], c:[] }`, there are nodes `Node('a')` and `Node('b')` and `Node('c')`. Node c does not have any edges associated with it. Edges `a->b` (with total distance 2 and outdoor distace 1) and `a->c` (with total distance 3 and outdoor distace 2) and `b->c` (with total distance 4 and outdoor distace 2).

Careful using the `__eq__` method

- In object oriented programming, you overload certain methods, like `__str__` so that you can inspect he values of the variables during debugging. One pitfall of this is that you may get confused about types.
- An error such as `AttributeError: 'str' object has no attribute 'name'` means that you are trying to access the data attribute called `name` of an object of type `string`). Instead, you should be accessing data attribute `name` of an object of type `Node`.

- Make sure you are storing graph nodes as something like `Node('some string')` instead of just `'some string'`

"See Full Output" and debugging

- If you are creating your own small play graphs, you should add `WeightedEdge`s with `some_edge = WeightedEdge(Node('a'), Node('b'), 45, 30)`. For readability, the grader shows you this same test case as: `some_edge = WeightedEdge(a, b, 45, 30)`

Paste your code for both `WeightedEdge` and `WeightedDigraph` classes below. You may assume the grader has provided implementations for Node, Edge, and Digraph.

Test: 1 WeightedEdges

```
Initialize some WeightedEdges
```

Output:

```
    na = Node('a')
    nb = Node('b')
    nc = Node('c')
    e1 = WeightedEdge(na, nb, 15, 10)
    isinstance(e1, Edge): True
    isinstance(e1, WeightedEdge): True
    e1.getSource(): a
    e1.getDestination(): b
    e1.getTotalDistance(): 15
    e1.getOutdoorDistance(): 10

    Test completed
```

Test: 2 WeightedEdges randomized

```
Initialize some WeightedEdges
```

Output:

```
    nj = Node('j')
    nk = Node('k')
    nm = Node('m')
    ng = Node('g')
    randomEdge = WeightedEdge(j, ng, 65, 22)
    isinstance(randomEdge, Edge): True
    isinstance(randomEdge, WeightedEdge): True
    randomEdge.getSource(): j
    randomEdge.getDestination(): g
    randomEdge.getTotalDistance(): 65
    randomEdge.getOutdoorDistance(): 22
    randomEdge = WeightedEdge(j, ng, 64, 23)
    isinstance(randomEdge, Edge): True
    isinstance(randomEdge, WeightedEdge): True
```

```
randomEdge.getSource(): j
randomEdge.getDestination(): g
randomEdge.getTotalDistance(): 64
randomEdge.getOutdoorDistance(): 23
randomEdge = WeightedEdge(m, ng, 53, 24)
isinstance(randomEdge, Edge): True
isinstance(randomEdge, WeightedEdge): True
randomEdge.getSource(): m
randomEdge.getDestination(): g
randomEdge.getTotalDistance(): 53
randomEdge.getOutdoorDistance(): 24
randomEdge = WeightedEdge(k, ng, 21, 7)
isinstance(randomEdge, Edge): True
isinstance(randomEdge, WeightedEdge): True
randomEdge.getSource(): k
randomEdge.getDestination(): g
randomEdge.getTotalDistance(): 21
randomEdge.getOutdoorDistance(): 7
randomEdge = WeightedEdge(m, ng, 39, 26)
isinstance(randomEdge, Edge): True
isinstance(randomEdge, WeightedEdge): True
randomEdge.getSource(): m
randomEdge.getDestination(): g
randomEdge.getTotalDistance(): 39
randomEdge.getOutdoorDistance(): 26

Test completed
```

Test: 3 WeightedDigraph 1

```
Initialize a WeightedDigraph and add nodes
```

Output:

```
na = Node('a')
nb = Node('b')
nc = Node('c')
g = WeightedDigraph()
isinstance(g, Digraph): True
isinstance(g, WeightedDigraph): True
g.addNode(na)
g.addNode(nb)
g.hasNode(na): True
g.hasNode(nb): True
g.hasNode(nc): False

Test completed
```

Test: 4 WeightedDigraph 2

```
Initialize a WeightedDigraph and add nodes and edges
```

Output:

```
    na = Node('a')
    nb = Node('b')
    nc = Node('c')
    g = WeightedDigraph()
    isinstance(g, Digraph): True
    isinstance(g, WeightedDigraph): True
    g.addNode(na)
    g.addNode(nb)
    g.addNode(nc)
    e1 = WeightedEdge(na, nb, 15, 10)
    e2 = WeightedEdge(na, nc, 14, 6)
    e3 = WeightedEdge(nb, nc, 3, 1)
    g.addEdge(e1)
    g.addEdge(e2)
    g.addEdge(e3)

    Test completed
```

Test: 5 WeightedDigraph 3

```
Initialize a WeightedDigraph and add nodes and edges
```

Output:

```
    nh = Node('h')
    nj = Node('j')
    nk = Node('k')
    nm = Node('m')
    ng = Node('g')
    g = WeightedDigraph()
    g.addNode(nh)
    g.addNode(nj)
    g.addNode(nk)
    g.addNode(nm)
    g.addNode(ng)
    randomEdge = WeightedEdge(h, k, 100, 74)
    g.addEdge(randomEdge)
    randomEdge = WeightedEdge(h, m, 51, 34)
    g.addEdge(randomEdge)
    randomEdge = WeightedEdge(m, k, 61, 37)
    g.addEdge(randomEdge)
    randomEdge = WeightedEdge(j, h, 83, 41)
    g.addEdge(randomEdge)
    randomEdge = WeightedEdge(j, k, 76, 39)
    g.addEdge(randomEdge)
    randomEdge = WeightedEdge(k, h, 41, 11)
    g.addEdge(randomEdge)
```

```
randomEdge = WeightedEdge(j, m, 16, 11)
g.addEdge(randomEdge)
randomEdge = WeightedEdge(m, k, 65, 64)
g.addEdge(randomEdge)
g.childrenOf(nh): [k, m]
g.childrenOf(nj): [h, k, m]
g.childrenOf(nk): [h]
g.childrenOf(nm): [k, k]
g.childrenOf(ng): []

Test completed
```

Test: 6 str method for WeightedEdges

```
Defining WeightedEdges, then printing them out
```

Output:

```
nx = Node('x')
ny = Node('y')
nz = Node('z')
e1 = WeightedEdge(nx, ny, 18, 8)
print e1
x->y (18, 8)
e2 = WeightedEdge(ny, nz, 20, 1)
print e2
y->z (20, 1)
e3 = WeightedEdge(nz, nx, 7, 6)
print e3
z->x (7, 6)

Test completed
```

Test: 7 str method for WeightedEdges randomized

```
Initialize some WeightedEdges
```

Output:

```
nj = Node('j')
nk = Node('k')
nm = Node('m')
ng = Node('g')
randomEdge = WeightedEdge(k, ng, 61, 20)
print randomEdge
k->g (61, 20)
randomEdge = WeightedEdge(m, ng, 99, 56)
print randomEdge
m->g (99, 56)
randomEdge = WeightedEdge(k, ng, 40, 23)
```

```
print randomEdge
k->g (40, 23)
randomEdge = WeightedEdge(k, ng, 82, 79)
print randomEdge
k->g (82, 79)
randomEdge = WeightedEdge(k, ng, 27, 14)
print randomEdge
k->g (27, 14)
Test completed
```

Test: 8 str method for graphs

`Defining a WeightedDigraph, then printing it out`

Output:

```
nx = Node('x')
ny = Node('y')
nz = Node('z')
e1 = WeightedEdge(nx, ny, 18, 8)
e2 = WeightedEdge(ny, nz, 20, 1)
e3 = WeightedEdge(nz, nx, 7, 6)
g = WeightedDigraph()
g.addNode(nx)
g.addNode(ny)
g.addNode(nz)
g.addEdge(e1)
g.addEdge(e2)
g.addEdge(e3)
print g
y->z (20.0, 1.0)
x->y (18.0, 8.0)
z->x (7.0, 6.0)

Test completed
```

Test: 9 str method for graphs randomized

`Initialize a WeightedDigraph and add nodes and edges`

Output:

```
nj = Node('j')
nk = Node('k')
nm = Node('m')
ng = Node('g')
g = WeightedDigraph()
g.addNode(nj)
g.addNode(nk)
g.addNode(nm)
```

```
g.addNode(ng)
randomEdge = WeightedEdge(g, j, 42, 14)
g.addEdge(randomEdge)
randomEdge = WeightedEdge(m, j, 96, 14)
g.addEdge(randomEdge)
randomEdge = WeightedEdge(j, g, 95, 20)
g.addEdge(randomEdge)
randomEdge = WeightedEdge(m, j, 31, 14)
g.addEdge(randomEdge)
randomEdge = WeightedEdge(j, g, 73, 73)
g.addEdge(randomEdge)
randomEdge = WeightedEdge(j, m, 70, 49)
g.addEdge(randomEdge)
randomEdge = WeightedEdge(j, g, 21, 21)
g.addEdge(randomEdge)
randomEdge = WeightedEdge(g, m, 46, 43)
g.addEdge(randomEdge)
print g
j->g (95.0, 20.0)
j->g (73.0, 73.0)
j->m (70.0, 49.0)
j->g (21.0, 21.0)
m->j (96.0, 14.0)
m->j (31.0, 14.0)
g->j (42.0, 14.0)
g->m (46.0, 43.0)

Test completed
```