SFWR 4C03 – Assignment 2 Report

Andrew Lutz
0664122
lutza
March 7, 2011

**Python Dynamic Routing Policy (OSPF)**

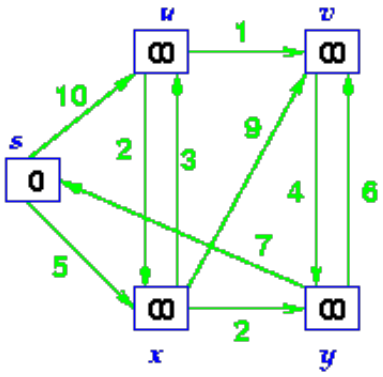
I solved this problem by doing a fine decomposition of tasks that needed to be executed by our program. For instance I have a method to initially parse the user input into 2 parts, command and input nodes. From this I have a method that checks what command was entered as well as performing extensive error checking on the input command. I then tackle the problem of the input nodes, this includes checking if nodes already exist in network, also if the user wants to input a range of nodes (ie. `add rt 10-20`) I have a separate function that takes '10-20' and returns a range in the form of a list, of desired input nodes. The add function then adds a single node, or if a range was entered the method will iterate over the range and execute the add function for each node. The 'con' method has error checking against nonnumeric input, negative and zero edge cost, routers that do not exist, non integer edge cost, connecting a node to itself, it also checks that 3 input arguments are received. Con method basically consists of error checking on input, establishing connection between nodes requires only one line of code. For display formatting I implemented a padding adjustment procedure, basically if the edge cost was two digits, shave a space off the pad to keep all the numbers in the column aligned. The point of this was to maintain consistency in the output of the display command.

The biggest challenge I encountered during programming this was learning how to use dictionary data types in python, they are very handy for our type of problem as each key in a dictionary can have sub-keys, connected nodes in our case, and store a value for each sub key, edge cost in our case. However they are a bit tricky to use at first, as they have different access methods than List's and a few other kinks. I also had a problem when trying to store my shortest paths for each node, initially I tried storing the paths in dictionary's but later found out that dictionary's have no sense of order. This was what I read at http://diveintopython.org, "Dictionaries have no concept of order among elements. It is incorrect to say that the elements are "out of order"; they are simply unordered. This is an important distinction which will annoy you when you want to access the elements of a dictionary in a specific, repeatable order (like alphabetical order by key)." I then resorted to using a 2-dimensional array to store the pathes. Other than this, everything else went pretty smoothly. I found the majority of programming for this program was error checking on input and such, implementing Dijkstra's Algorithm was not difficult.

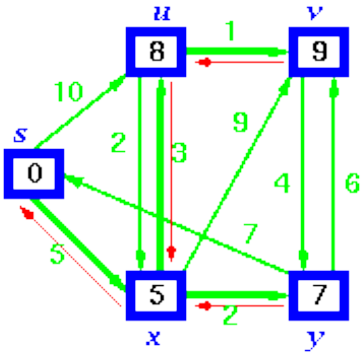
**Program Testing**

**Test case 1** – The graph for this problem was taken from the website, http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/dijkstraAlgor.htm

Before relaxation




After relaxation




In my program the graph was represented as follows:

```
nodes = {'rt1':{'rt2':10, 'rt4':5}, 'rt2':{'rt3':1, 'rt4':2},
'rt3':{'rt5':4}, 'rt4':{'rt2':3, 'rt3':9, 'rt5':2},
'rt5':{'rt1':7, 'rt3':6}}
```

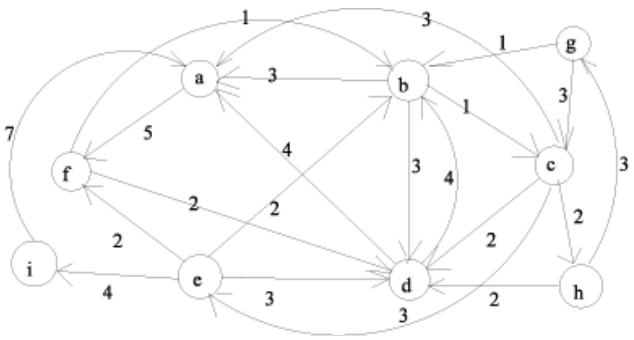
Where rt1=s, rt2=u, rt3=v, rt4=x, rt5=y

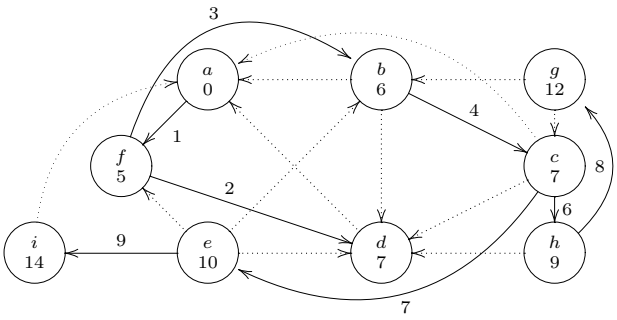
Program dump:

```
Andrew-Lutzs-MacBook-Pro:a2 Andrew$ ./routed.py

Enter a command...
display

    rt1  rt2  rt3  rt4  rt5
rt1                     7
rt2  10             3
rt3       1         9    6
rt4  5    2
rt5            4    2

Enter a command...
tree rt1

       7 : rt1, rt4, rt5
       5 : rt1, rt4
       9 : rt1, rt4, rt2, rt3
       8 : rt1, rt4, rt2
```

The shortest path costs listed on the final graph clearly match the output of the tree command. Another website with the same example states, "The shortest path from s to v is ['s', 'x', 'u', 'v'] and has length 9." This is clearly true here as the shortest path from rt1 to rt3 has length 9 and goes from rt1 -> rt4 -> rt2 -> rt3. Test case passed.

**Test case 2** – The graph for this test case was taken off an old assignment I had in my SFWR 2C03 Algorithms class.




The graph was represented as follows:

```
nodes = {'rt1':{'rt6':5}, 'rt2':{'rt1':3, 'rt3':1, 'rt4':3},
'rt3':{'rt1':3, 'rt8':2, 'rt4':2, 'rt5':3}, 'rt4':{'rt1':4,
'rt2':4}, 'rt5':{'rt9':4, 'rt2':2, 'rt6':2, 'rt4':3},
'rt6':{'rt2':1, 'rt4':2}, 'rt7':{'rt2':1, 'rt3':3},
'rt8':{'rt7':3, 'rt4':2}, 'rt9':{'rt1':7}}
```

Where rt1=a, rt2=b, rt3=c, rt4=d, rt5=e, rt6=f, rt7=g, rt8=h, rt9=i

Program dump:

```
Andrew-Lutzs-MacBook-Pro:a2 Andrew$ ./routed.py

Enter a command...
display

    rt1  rt2  rt3  rt4  rt5  rt6  rt7  rt8  rt9
rt1       3    3    4                        7
rt2                 4    2    1    1
rt3       1                        3
rt4       3    2         3    2         2
rt5            3
rt6  5                   2
rt7                                  3
rt8            2
rt9                      4

Enter a command...
tree rt1

       10 : rt1, rt6, rt2, rt3, rt5
       7 : rt1, rt6, rt4
       12 : rt1, rt6, rt2, rt3, rt8, rt7
       5 : rt1, rt6
       7 : rt1, rt6, rt2, rt3
       6 : rt1, rt6, rt2
       14 : rt1, rt6, rt2, rt3, rt5, rt9
       9 : rt1, rt6, rt2, rt3, rt8
```

The shortest path costs listed on the final graph clearly match the output of the tree command. The solutions say, the path to e is a → f → b → c → e. This is clearly true here as the shortest path from rt1 to rt5 has length 10 and goes from rt1 -> rt6 -> rt2 -> rt3 -> rt5. Test case passed.

Sample program dump:

```
Andrew-Lutzs-MacBook-Pro:a2 Andrew$ ./routed.py

Enter a command...
display

    rt1  rt2  rt3
rt1       1    4
rt2  10
rt3  5    2

Enter a command...
add rt 1-3
ERROR: Router,  rt1  already exists.
ERROR: Router,  rt2  already exists.
ERROR: Router,  rt3  already exists.

Enter a command...
add rtadsfasd4
Command not recognized

Enter a command...
add rt 4
Router, rt4 successfully added.

Enter a command...
add nt 3-5
Network, nt3 successfully added.
Network, nt4 successfully added.
Network, nt5 successfully added.

Enter a command...
display

    nt3  nt4  nt5  rt1  rt2  rt3  rt4
nt3
nt4
nt5
rt1                      1    4
rt2                 10
rt3                 5    2
rt4

Enter a command...
con nt4 rt1 99
Connection from nt4 to rt1 made successfully

Enter a command...
display

    nt3  nt4  nt5  rt1  rt2  rt3  rt4
nt3
nt4
nt5
rt1       99             1    4
rt2                 10
rt3                 5    2
rt4

Enter a command...
del rt 1-4
Router,  rt1 successfully removed.
Router,  rt2 successfully removed.
Router,  rt3 successfully removed.
Router,  rt4 successfully removed.

Enter a command...
display
```

```
    nt3  nt4  nt5
nt3
nt4
nt5

Enter a command...
quit
Andrew-Lutzs-MacBook-Pro:a2 Andrew$
```