Lesson 5.

Solving shortest path problems with networkx

Overview

- In this lesson, we'll learn how to use some Python packages to solve shortest path problems.

An example

You have just purchased a new car for $22,000. The cost of maintaining a car during a year depends on its age at the beginning of the year:

Age of car (years)	0	1	2	3	4
Annual maintenance cost ($)	2,000	3,000	4,000	8,000	12,000

To avoid the high maintenance costs associated with an older car, you may trade in your car and purchase a new car. The price you receive on a trade-in depends on the age of the car at the time of the trade-in:

Age of car (years)	1	2	3	4	5
Trade-in price ($)	15,000	12,000	9,000	5,000	2,000

For now, assume that at any time, it costs $22,000 to purchase a new car. Your goal is to minimize the net cost (purchasing costs + maintenance costs − money received in trade-ins) incurred over the next five years.

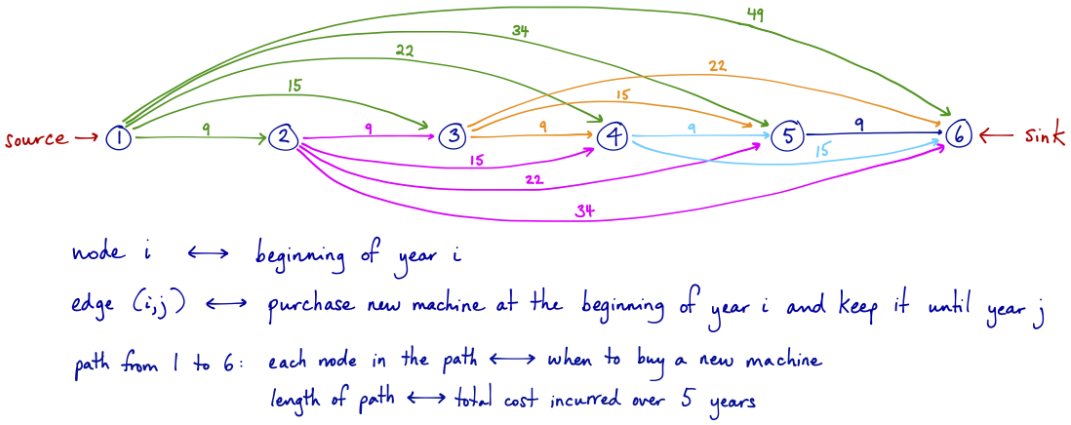
1. Formulate your problem as a shortest path problem.
2. Solve your shortest path formulation: give the shortest path length and a shortest path.
3. Interpret the shortest path length and shortest path in the context of the problem.

- Recall that we formulated this problem as a shortest path problem back in Lesson 1, like this:

- Let's solve this shortest path problem — that is,
  - obtain the length of a shortest path
  - obtain the nodes/edges in a shortest path
- First obstacle: how do we represent graphs in Python?

Finding Python packages

- If you want to do something in Python, there's a very good chance that there's a package that will help you out.
- The Python Package Index, or PyPI, is the principal repository for Python software.



Solution to car maintenance example

- Let's try searching PyPI for "graphs".
- We can also Google "python graphs" and see if any useful packages pop up.
- Don't just choose the first package you find — read the package's documentation (*make sure it has good documentation*) and see if it is suitable for you.

Installing Python packages

- In this class, we'll use `networkx` to represent graphs.
    - It has good documentation, it's pretty easy to use, and has a lot of built-in functionality.
- To install `networkx`, open a WinPython Command Prompt and type:

```
pip install networkx
```

- `pip` might tell you that networkx is already installed. If not, it should go ahead and install it for you.

Building a graph in networkx

- To use `networkx`, we first need to **`import`** it so that we can access its functions, like this:

```
In [2]: import networkx as nx
```

- `as nx` in the cell above lets us refer to `networkx` as `nx`.
    - This helps us save some keystrokes and keeps our code a bit cleaner.
- Let's build the directed graph for the example above in `networkx`.
- We can start by creating an empty digraph called `G`, like this:

```
In [3]: # Create empty digraph
        G = nx.DiGraph()
```

- Next, let's add nodes 1-6 to `G`. We can do this by using the **`.add_node()`** method on `G`, like so:

```
In [4]: # Add nodes
        G.add_node(1)
        G.add_node(2)
        G.add_node(3)
        G.add_node(4)
        G.add_node(5)
        G.add_node(6)
```

- *Food for thought.* What is a shorter way of expressing the code in the above cell?

We can use a `for` loop to add the nodes instead.

- Now we need to add the edges, as well as the length of each edge.
- We can do this by using the **`.add_edge()`** method on `G`.
- For example, to add the edge $(1, 2)$ with length 9, we would write:

```
In [5]: # Add edge (1,2) with length 9
        G.add_edge(1, 2, length=9)
```

- In the above statement, we are assigning edge $(1, 2)$ an **attribute** called `length` with value 9.
- Note that `length` is a name of our choosing.
- We could have used `weight` or `cost` or `monkey` instead of `length`.
- As long as we are consistent across all the edges we define, then we can use the edge lengths easily later on.
- Let's add the rest of the edges and edge lengths:

```
In [6]: # Add the rest of the edges outgoing from node 1
        G.add_edge(1, 3, length=15)
        G.add_edge(1, 4, length=22)
        G.add_edge(1, 5, length=34)
        G.add_edge(1, 6, length=49)

        # Add edges outgoing from node 2
        G.add_edge(2, 3, length=9)
        G.add_edge(2, 4, length=15)
        G.add_edge(2, 5, length=22)
        G.add_edge(2, 6, length=34)

        # Add edges outgoing from node 3
        G.add_edge(3, 4, length=9)
        G.add_edge(3, 5, length=15)
        G.add_edge(3, 6, length=22)

        # Add edges outgoing from node 4
        G.add_edge(4, 5, length=9)
        G.add_edge(4, 6, length=15)
```

```
        # Add edges outgoing from node 5
        G.add_edge(5, 6, length=9)
```

Accessing edge information

- Two nodes are **adjacent** if they are endpoints of the same edge.
- We can find the nodes adjacent to node 2 by outgoing edges like this:

```
In [7]: # Print nodes adjacent to node 2 by outgoing edges
        print(G[2])
```

```
{3: {'length': 9}, 4: {'length': 15}, 5: {'length': 22}, 6: {'length': 34}}
```

- We see that $(2, 3)$ is an edge in G with length 9, as desired.
- Same goes for $(2, 4)$, $(2, 5)$, and $(2, 6)$.
- We can find the length of edge $(2, 5)$ directly like this:

```
In [8]: # Print the length of edge (2, 5)
        print(G[2][5]["length"])
```

```
22
```

- Note that if we used a different attribute name above, we would need to use that instead of `"length"`.
- For example,

```
print(G[2][5]["monkey"])
```

- *Fine point.* `networkx` stores graph information as dictionaries nested within dictionaries.

Solving shortest path problems

- The **Bellman-Ford algorithm** solves shortest path problems on directed graphs with negative edge lengths.
    - The algorithm will output either (i) a shortest path from the source node to the target node, or (ii) declare that there is a negative directed cycle.
- `networkx` has an implementation of the Bellman-Ford algorithm, but it doesn't automatically output the most useful information.
- We will use a package called `bellmanford` that extends `networkx`'s implementation of the Bellman-Ford algorithm to output useful information easily.
- To install `bellmanford`, open a WinPython Command Prompt and type:

```
pip install bellmanford
```

- To use `bellmanford`, we must first import it, just like with `networkx`:

```
In [9]: import bellmanford as bf
```

- To find the shortest path from node 1 to node 6 in the graph `G` we created above, we can do this:

```
In [10]: path_length, path_nodes, negative_cycle = bf.bellman_ford(G, source=1, target=6, weight="length")
```

- The `weight` argument corresponds to the edge attribute we defined to have the edge lengths.
- `bellman_ford` has three outputs: see what they look like below:

```
In [11]: print("Is there a negative cycle? {0}".format(negative_cycle))
         print("Shortest path length: {0}".format(path_length))
         print("Shortest path: {0}".format(path_nodes))
```

```
Is there a negative cycle? False
Shortest path length: 37
Shortest path: [1, 3, 6]
```

Interpreting the output

Example. How does the output from the previous code cell translate to the original problem?

The length of the shortest path represents the minimum cost of owning and maintaining a car over the next 5 years, which in this case is 37,000 dollars.

The nodes in the shortest path indicate when to buy a new car (and trade-in the old car). In this case, we should buy a new car in year 1 and year 3.

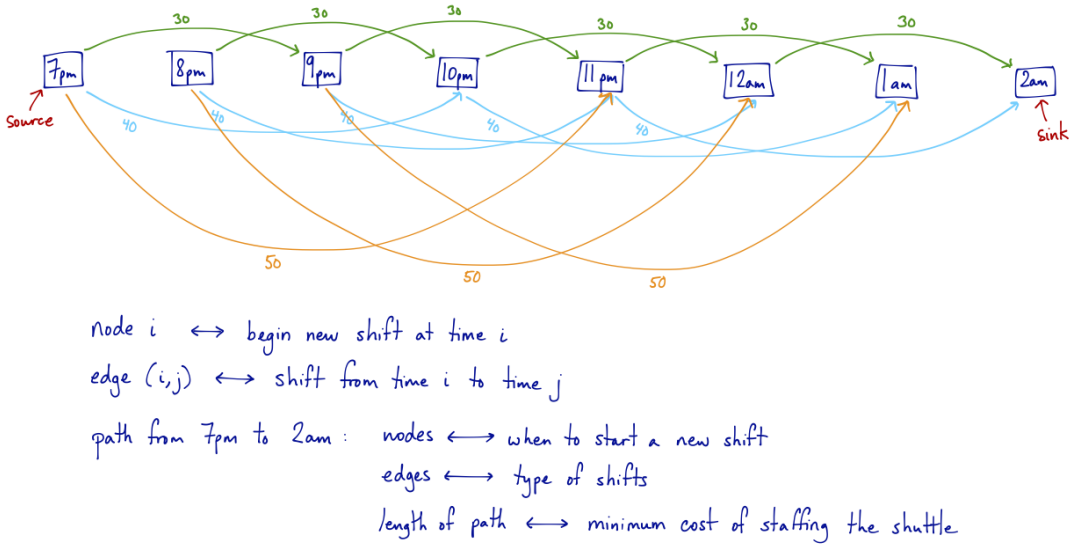
Another example — on your own

- Here's another example from Lesson 1.

The Simplexville College campus shuttle bus begins running at 7:00pm and continues until 2:00am. Several drivers will be used, but only one should be on duty at any time. If a shift starts at or before 9:00pm, a regular driver can be obtained for a 4-hour shift at a cost of $50. Otherwise, part-time drivers need to be used. Several part-time drivers can work 3-hour shifts at $40, and the rest are limited to 2-hour shifts at $30. The college's goal is to schedule drivers in a way that minimizes the total cost of staffing the shuttle bus.

1. Formulate this problem as a shortest path problem.
2. Solve your shortest path formulation: give the shortest path length and a shortest path.
3. Interpret the shortest path length and shortest path in the context of the problem.

- We formulated this as a shortest path problem as follows:



Solution to shuttle bus example

```
In [12]: # Write your code here
         # Create empty graph
         G = nx.DiGraph()

         # Add nodes
         G.add_node("7pm")
         G.add_node("8pm")
         G.add_node("9pm")
         G.add_node("10pm")
         G.add_node("11pm")
         G.add_node("12am")
         G.add_node("1am")
         G.add_node("2am")

         # Add edges for 4-hour shifts
         G.add_edge("7pm", "11pm", length=50)
         G.add_edge("8pm", "12am", length=50)
         G.add_edge("9pm", "1am", length=50)

         # Add edges for 3-hour shifts
         G.add_edge("7pm", "10pm", length=40)
         G.add_edge("8pm", "11pm", length=40)
         G.add_edge("9pm", "12am", length=40)
         G.add_edge("10pm", "1am", length=40)
         G.add_edge("11pm", "2am", length=40)

         # Add edges for 2-hour shifts
         G.add_edge("7pm", "9pm", length=30)
         G.add_edge("8pm", "10pm", length=30)
         G.add_edge("9pm", "11pm", length=30)
         G.add_edge("10pm", "12pm", length=30)
         G.add_edge("11pm", "1am", length=30)
         G.add_edge("12pm", "2am", length=30)

         # Solve shortest path problem
         path_length, path_nodes, negative_cycle = bf.bellman_ford(G, source="7pm", target="2am", weight="length")
```

```
            print("Is there a negative cycle? {0}".format(negative_cycle))
            print("Shortest path length: {0}".format(path_length))
            print("Shortest path: {0}".format(path_nodes))
```

```
Is there a negative cycle? False
Shortest path length: 90
Shortest path: ['7pm', '11pm', '2am']
```

The length of the shortest path is the minimum cost of staffing the shuttle from 7pm to 2am, which in this case is 90 dollars.

The nodes in the shortest path indicate when to change drivers, and the edges in the shortest path represent the shifts used. In this case, we should have a driver on a 4-hour shift from 7pm to 11pm, and then a 3-hour shift from 11pm to 2am.