

Lesson 6.

The mileage running problem

The problem

Professor May B. Wright needs to fly from Baltimore (BWI) to Los Angeles (LAX) to attend a conference. She thinks this would be the perfect opportunity to accumulate some frequent flyer miles on American Airlines (AA), where she already has Platinum status.

Looking into flights on AA, she sees that every itinerary from BWI to LAX costs roughly the same. She has a full day to spare for travel, so she wants to know: which sequence of AA domestic flights starting at BWI and ending at LAX over the course of one day will allow her to accumulate the most miles?

- Yes, people actually do this. This is known as **mileage running**.
 - Apparently, this has become harder to do in recent years.
 - [A recent article from the New York Times](#).
 - [An older article from Wired](#).

Modeling the problem

- Suppose we have a database of every AA domestic flight on a given day.
- In particular, for each flight, we have:
 - the flight number
 - the origin airport
 - the destination airport
 - the departure time at the origin airport
 - the arrival time at the destination airport
 - the distance traveled in miles
- How can we formulate Professor Wright's problem as a shortest path problem?

pandas (the package, not the animals)

- In the same folder as this notebook, there is a file called `aa_domestic_flights.csv` with the database described above.
- `.csv` stands for **comma-separated values**.
- We can view `.csv` files in Excel - let's see what's in this file. *Cut to Excel...*
- How can we use this data in Python? With **pandas**.
- pandas is a Python package for data analysis.
 - It's especially useful for cleaning and manipulating datasets.

- pandas does a lot of stuff — here are a few resources:
 - [Here is the official documentation for pandas.](#)
 - [Chris Albon's notes](#) are also a good resource on how to get things done with pandas (look in the *Data Wrangling* section).
- In this lesson, we'll use pandas in a very basic way to help us set up the shortest path problem we formulated above.
- To install pandas, open a WinPython Command Prompt and type

```
pip install pandas
```

- pip might tell you that pandas is already installed. If not, it should go ahead and install it for you.
- To use pandas, we first need to import it, like this:

```
In [2]: import pandas as pd
```

- A pandas **DataFrame** is just a two-dimensional table, with rows and columns.
- We can use the `read_csv()` function in pandas to read `aa_domestic_flights.csv` into a DataFrame called `df`, like this:

```
In [3]: # Read csv file into a DataFrame
        # Designate departure and arrival time columns as dates
        df = pd.read_csv('aa_domestic_flights.csv', parse_dates=['DEP_TIME', 'ARR_TIME'])
```

- By default, `read_csv()` assumes the first row of the csv file contains the names of each column.
- The `parse_dates` argument tells `read_csv()` which columns correspond to dates, so that we can perform date-specific calculations on these columns later.
- [Here is the official documentation for read_csv\(\).](#)
- It's a good idea to take a quick look at the DataFrame `read_csv()` creates, just in case something went wrong.
- To examine the first 5 rows of a DataFrame, we can use the `.head()` method:

```
In [4]: # Print the first 5 rows of df
        df.head()
```

```
Out[4]:
```

	FLIGHT	ORIGIN	DEST	DEP_TIME	ARR_TIME	DISTANCE
0	1-BOS-JFK	BOS	JFK	2016-09-01 06:00:00	2016-09-01 07:15:00	187.0
1	40-BOS-ORD	BOS	ORD	2016-09-01 19:12:00	2016-09-01 22:02:00	867.0
2	147-BOS-LAX	BOS	LAX	2016-09-01 15:15:00	2016-09-01 21:45:00	2611.0
3	197-BOS-ORD	BOS	ORD	2016-09-01 15:30:00	2016-09-01 18:24:00	867.0
4	198-BOS-JFK	BOS	JFK	2016-09-01 13:10:00	2016-09-01 14:31:00	187.0

- Another useful method is `.describe()`.
- By default, `.describe()` only provides summary statistics for the columns with numeric data.
- To get summary statistics for all the columns, include the argument `include="all"`, like this:

```
In [5]: # Get summary statistics for all columns in df
df.describe(include="all")
```

```
Out[5]:
```

	FLIGHT	ORIGIN	DEST	DEP_TIME	ARR_TIME
count	2607	2607	2607	2607	2607
unique	2607	94	94	643	1046
top	32-LAX-JFK	DFW	DFW	2016-09-01 07:00:00	2016-09-01 16:45:00
freq	1	408	407	43	12
first	NaN	NaN	NaN	2016-09-01 02:15:00	2016-09-01 06:01:00
last	NaN	NaN	NaN	2016-09-02 03:50:00	2016-09-02 09:33:00
mean	NaN	NaN	NaN	NaN	NaN
std	NaN	NaN	NaN	NaN	NaN
min	NaN	NaN	NaN	NaN	NaN
25%	NaN	NaN	NaN	NaN	NaN
50%	NaN	NaN	NaN	NaN	NaN
75%	NaN	NaN	NaN	NaN	NaN
max	NaN	NaN	NaN	NaN	NaN

	DISTANCE
count	2607.000000
unique	NaN
top	NaN
freq	NaN
first	NaN
last	NaN
mean	986.709628
std	645.910443
min	83.000000
25%	507.000000
50%	868.000000
75%	1272.000000
max	3784.000000

- A column by itself is called a **Series**.
- You can select the Series `DEST` of the DataFrame `df` like this:

```
df["DEST"]
```

- So, to print the Series `DEST`, we could write:

```
In [6]: # Print the DEST column
print(df["DEST"])
```

```
0    JFK
1    ORD
2    LAX
3    ORD
4    JFK
5    JFK
6    ORD
7    LAX
8    MIA
9    DFW
10   PHX
11   ORD
```

```
12    MIA
13    DFW
14    MIA
15    LAX
16    DFW
17    ORD
18    ORD
19    ORD
20    MIA
21    ORD
22    LAX
23    MIA
24    JFK
25    DFW
26    DFW
27    ORD
28    PHX
29    MIA
...
2577  PHL
2578  CLT
2579  CLT
2580  CLT
2581  CLT
2582  CLT
2583  CLT
2584  CLT
2585  CLT
2586  CLT
2587  CLT
2588  CLT
2589  CLT
2590  CLT
2591  CLT
2592  CLT
2593  CLT
2594  CLT
2595  CLT
2596  CLT
2597  CLT
2598  CLT
2599  PHX
2600  PHX
2601  PHX
2602  DFW
2603  DFW
2604  DFW
2605  PHX
2606  PHX
```

```
Name: DEST, dtype: object
```

Setting up the shortest path problem in networkx

- Now that we can access the flight database in Python, we can use its contents to setup the shortest path problem we formulated above.
- First, let's import `networkx` and `bellmanford` so we can use them:

```
In [7]: import networkx as nx
import bellmanford as bf
```

Creating a list of flights

- It will be useful to create a variable `flights` containing a list of all the flights.
- What part of the dataset contains this information?

We need to look at the `FLIGHT` column of the dataset.

- From the `.describe()` output above, we see that the flights in `df["FLIGHT"]` are unique.
- We can convert the Series `df["FLIGHT"]` to a list with the function `list()`.
 - Then we can use the list methods we learned about earlier, such as `.append()`, if necessary.

```
In [8]: # Take the FLIGHT column from df, convert it to a list
flights = list(df["FLIGHT"])
```

- It's a good idea to make sure nothing funny happened — let's inspect the variable `flights` we just created:

```
In [9]: # Print flights
print("Flights: {0}".format(flights))
```

Flights: ['1-BOS-JFK', '40-BOS-ORD', '147-BOS-LAX', '197-BOS-ORD', '198-BOS-JFK', '85-BOS-JFK', '252-BOS-ORD', '333-BOS-LAX', '334-BOS-LAX', '335-BOS-LAX', '336-BOS-LAX', '337-BOS-LAX', '338-BOS-LAX', '339-BOS-LAX', '340-BOS-LAX', '341-BOS-LAX', '342-BOS-LAX', '343-BOS-LAX', '344-BOS-LAX', '345-BOS-LAX', '346-BOS-LAX', '347-BOS-LAX', '348-BOS-LAX', '349-BOS-LAX', '350-BOS-LAX', '351-BOS-LAX', '352-BOS-LAX', '353-BOS-LAX', '354-BOS-LAX', '355-BOS-LAX', '356-BOS-LAX', '357-BOS-LAX', '358-BOS-LAX', '359-BOS-LAX', '360-BOS-LAX', '361-BOS-LAX', '362-BOS-LAX', '363-BOS-LAX', '364-BOS-LAX', '365-BOS-LAX', '366-BOS-LAX', '367-BOS-LAX', '368-BOS-LAX', '369-BOS-LAX', '370-BOS-LAX', '371-BOS-LAX', '372-BOS-LAX', '373-BOS-LAX', '374-BOS-LAX', '375-BOS-LAX', '376-BOS-LAX', '377-BOS-LAX', '378-BOS-LAX', '379-BOS-LAX', '380-BOS-LAX', '381-BOS-LAX', '382-BOS-LAX', '383-BOS-LAX', '384-BOS-LAX', '385-BOS-LAX', '386-BOS-LAX', '387-BOS-LAX', '388-BOS-LAX', '389-BOS-LAX', '390-BOS-LAX', '391-BOS-LAX', '392-BOS-LAX', '393-BOS-LAX', '394-BOS-LAX', '395-BOS-LAX', '396-BOS-LAX', '397-BOS-LAX', '398-BOS-LAX', '399-BOS-LAX', '400-BOS-LAX', '401-BOS-LAX', '402-BOS-LAX', '403-BOS-LAX', '404-BOS-LAX', '405-BOS-LAX', '406-BOS-LAX', '407-BOS-LAX', '408-BOS-LAX', '409-BOS-LAX', '410-BOS-LAX', '411-BOS-LAX', '412-BOS-LAX', '413-BOS-LAX', '414-BOS-LAX', '415-BOS-LAX', '416-BOS-LAX', '417-BOS-LAX', '418-BOS-LAX', '419-BOS-LAX', '420-BOS-LAX', '421-BOS-LAX', '422-BOS-LAX', '423-BOS-LAX', '424-BOS-LAX', '425-BOS-LAX', '426-BOS-LAX', '427-BOS-LAX', '428-BOS-LAX', '429-BOS-LAX', '430-BOS-LAX', '431-BOS-LAX', '432-BOS-LAX', '433-BOS-LAX', '434-BOS-LAX', '435-BOS-LAX', '436-BOS-LAX', '437-BOS-LAX', '438-BOS-LAX', '439-BOS-LAX', '440-BOS-LAX', '441-BOS-LAX', '442-BOS-LAX', '443-BOS-LAX', '444-BOS-LAX', '445-BOS-LAX', '446-BOS-LAX', '447-BOS-LAX', '448-BOS-LAX', '449-BOS-LAX', '450-BOS-LAX', '451-BOS-LAX', '452-BOS-LAX', '453-BOS-LAX', '454-BOS-LAX', '455-BOS-LAX', '456-BOS-LAX', '457-BOS-LAX', '458-BOS-LAX', '459-BOS-LAX', '460-BOS-LAX', '461-BOS-LAX', '462-BOS-LAX', '463-BOS-LAX', '464-BOS-LAX', '465-BOS-LAX', '466-BOS-LAX', '467-BOS-LAX', '468-BOS-LAX', '469-BOS-LAX', '470-BOS-LAX', '471-BOS-LAX', '472-BOS-LAX', '473-BOS-LAX', '474-BOS-LAX', '475-BOS-LAX', '476-BOS-LAX', '477-BOS-LAX', '478-BOS-LAX', '479-BOS-LAX', '480-BOS-LAX', '481-BOS-LAX', '482-BOS-LAX', '483-BOS-LAX', '484-BOS-LAX', '485-BOS-LAX', '486-BOS-LAX', '487-BOS-LAX', '488-BOS-LAX', '489-BOS-LAX', '490-BOS-LAX', '491-BOS-LAX', '492-BOS-LAX', '493-BOS-LAX', '494-BOS-LAX', '495-BOS-LAX', '496-BOS-LAX', '497-BOS-LAX', '498-BOS-LAX', '499-BOS-LAX', '500-BOS-LAX', '501-BOS-LAX', '502-BOS-LAX', '503-BOS-LAX', '504-BOS-LAX', '505-BOS-LAX', '506-BOS-LAX', '507-BOS-LAX', '508-BOS-LAX', '509-BOS-LAX', '510-BOS-LAX', '511-BOS-LAX', '512-BOS-LAX', '513-BOS-LAX', '514-BOS-LAX', '515-BOS-LAX', '516-BOS-LAX', '517-BOS-LAX', '518-BOS-LAX', '519-BOS-LAX', '520-BOS-LAX', '521-BOS-LAX', '522-BOS-LAX', '523-BOS-LAX', '524-BOS-LAX', '525-BOS-LAX', '526-BOS-LAX', '527-BOS-LAX', '528-BOS-LAX', '529-BOS-LAX', '530-BOS-LAX', '531-BOS-LAX', '532-BOS-LAX', '533-BOS-LAX', '534-BOS-LAX', '535-BOS-LAX', '536-BOS-LAX', '537-BOS-LAX', '538-BOS-LAX', '539-BOS-LAX', '540-BOS-LAX', '541-BOS-LAX', '542-BOS-LAX', '543-BOS-LAX', '544-BOS-LAX', '545-BOS-LAX', '546-BOS-LAX', '547-BOS-LAX', '548-BOS-LAX', '549-BOS-LAX', '550-BOS-LAX', '551-BOS-LAX', '552-BOS-LAX', '553-BOS-LAX', '554-BOS-LAX', '555-BOS-LAX', '556-BOS-LAX', '557-BOS-LAX', '558-BOS-LAX', '559-BOS-LAX', '560-BOS-LAX', '561-BOS-LAX', '562-BOS-LAX', '563-BOS-LAX', '564-BOS-LAX', '565-BOS-LAX', '566-BOS-LAX', '567-BOS-LAX', '568-BOS-LAX', '569-BOS-LAX', '570-BOS-LAX', '571-BOS-LAX', '572-BOS-LAX', '573-BOS-LAX', '574-BOS-LAX', '575-BOS-LAX', '576-BOS-LAX', '577-BOS-LAX', '578-BOS-LAX', '579-BOS-LAX', '580-BOS-LAX', '581-BOS-LAX', '582-BOS-LAX', '583-BOS-LAX', '584-BOS-LAX', '585-BOS-LAX', '586-BOS-LAX', '587-BOS-LAX', '588-BOS-LAX', '589-BOS-LAX', '590-BOS-LAX', '591-BOS-LAX', '592-BOS-LAX', '593-BOS-LAX', '594-BOS-LAX', '595-BOS-LAX', '596-BOS-LAX', '597-BOS-LAX', '598-BOS-LAX', '599-BOS-LAX', '600-BOS-LAX', '601-BOS-LAX', '602-BOS-LAX', '603-BOS-LAX', '604-BOS-LAX', '605-BOS-LAX', '606-BOS-LAX', '607-BOS-LAX', '608-BOS-LAX', '609-BOS-LAX', '610-BOS-LAX', '611-BOS-LAX', '612-BOS-LAX', '613-BOS-LAX', '614-BOS-LAX', '615-BOS-LAX', '616-BOS-LAX', '617-BOS-LAX', '618-BOS-LAX', '619-BOS-LAX', '620-BOS-LAX', '621-BOS-LAX', '622-BOS-LAX', '623-BOS-LAX', '624-BOS-LAX', '625-BOS-LAX', '626-BOS-LAX', '627-BOS-LAX', '628-BOS-LAX', '629-BOS-LAX', '630-BOS-LAX', '631-BOS-LAX', '632-BOS-LAX', '633-BOS-LAX', '634-BOS-LAX', '635-BOS-LAX', '636-BOS-LAX', '637-BOS-LAX', '638-BOS-LAX', '639-BOS-LAX', '640-BOS-LAX', '641-BOS-LAX', '642-BOS-LAX', '643-BOS-LAX', '644-BOS-LAX', '645-BOS-LAX', '646-BOS-LAX', '647-BOS-LAX', '648-BOS-LAX', '649-BOS-LAX', '650-BOS-LAX', '651-BOS-LAX', '652-BOS-LAX', '653-BOS-LAX', '654-BOS-LAX', '655-BOS-LAX', '656-BOS-LAX', '657-BOS-LAX', '658-BOS-LAX', '659-BOS-LAX', '660-BOS-LAX', '661-BOS-LAX', '662-BOS-LAX', '663-BOS-LAX', '664-BOS-LAX', '665-BOS-LAX', '666-BOS-LAX', '667-BOS-LAX', '668-BOS-LAX', '669-BOS-LAX', '670-BOS-LAX', '671-BOS-LAX', '672-BOS-LAX', '673-BOS-LAX', '674-BOS-LAX', '675-BOS-LAX', '676-BOS-LAX', '677-BOS-LAX', '678-BOS-LAX', '679-BOS-LAX', '680-BOS-LAX', '681-BOS-LAX', '682-BOS-LAX', '683-BOS-LAX', '684-BOS-LAX', '685-BOS-LAX', '686-BOS-LAX', '687-BOS-LAX', '688-BOS-LAX', '689-BOS-LAX', '690-BOS-LAX', '691-BOS-LAX', '692-BOS-LAX', '693-BOS-LAX', '694-BOS-LAX', '695-BOS-LAX', '696-BOS-LAX', '697-BOS-LAX', '698-BOS-LAX', '699-BOS-LAX', '700-BOS-LAX', '701-BOS-LAX', '702-BOS-LAX', '703-BOS-LAX', '704-BOS-LAX', '705-BOS-LAX', '706-BOS-LAX', '707-BOS-LAX', '708-BOS-LAX', '709-BOS-LAX', '710-BOS-LAX', '711-BOS-LAX', '712-BOS-LAX', '713-BOS-LAX', '714-BOS-LAX', '715-BOS-LAX', '716-BOS-LAX', '717-BOS-LAX', '718-BOS-LAX', '719-BOS-LAX', '720-BOS-LAX', '721-BOS-LAX', '722-BOS-LAX', '723-BOS-LAX', '724-BOS-LAX', '725-BOS-LAX', '726-BOS-LAX', '727-BOS-LAX', '728-BOS-LAX', '729-BOS-LAX', '730-BOS-LAX', '731-BOS-LAX', '732-BOS-LAX', '733-BOS-LAX', '734-BOS-LAX', '735-BOS-LAX', '736-BOS-LAX', '737-BOS-LAX', '738-BOS-LAX', '739-BOS-LAX', '740-BOS-LAX', '741-BOS-LAX', '742-BOS-LAX', '743-BOS-LAX', '744-BOS-LAX', '745-BOS-LAX', '746-BOS-LAX', '747-BOS-LAX', '748-BOS-LAX', '749-BOS-LAX', '750-BOS-LAX', '751-BOS-LAX', '752-BOS-LAX', '753-BOS-LAX', '754-BOS-LAX', '755-BOS-LAX', '756-BOS-LAX', '757-BOS-LAX', '758-BOS-LAX', '759-BOS-LAX', '760-BOS-LAX', '761-BOS-LAX', '762-BOS-LAX', '763-BOS-LAX', '764-BOS-LAX', '765-BOS-LAX', '766-BOS-LAX', '767-BOS-LAX', '768-BOS-LAX', '769-BOS-LAX', '770-BOS-LAX', '771-BOS-LAX', '772-BOS-LAX', '773-BOS-LAX', '774-BOS-LAX', '775-BOS-LAX', '776-BOS-LAX', '777-BOS-LAX', '778-BOS-LAX', '779-BOS-LAX', '780-BOS-LAX', '781-BOS-LAX', '782-BOS-LAX', '783-BOS-LAX', '784-BOS-LAX', '785-BOS-LAX', '786-BOS-LAX', '787-BOS-LAX', '788-BOS-LAX', '789-BOS-LAX', '790-BOS-LAX', '791-BOS-LAX', '792-BOS-LAX', '793-BOS-LAX', '794-BOS-LAX', '795-BOS-LAX', '796-BOS-LAX', '797-BOS-LAX', '798-BOS-LAX', '799-BOS-LAX', '800-BOS-LAX', '801-BOS-LAX', '802-BOS-LAX', '803-BOS-LAX', '804-BOS-LAX', '805-BOS-LAX', '806-BOS-LAX', '807-BOS-LAX', '808-BOS-LAX', '809-BOS-LAX', '810-BOS-LAX', '811-BOS-LAX', '812-BOS-LAX', '813-BOS-LAX', '814-BOS-LAX', '815-BOS-LAX', '816-BOS-LAX', '817-BOS-LAX', '818-BOS-LAX', '819-BOS-LAX', '820-BOS-LAX', '821-BOS-LAX', '822-BOS-LAX', '823-BOS-LAX', '824-BOS-LAX', '825-BOS-LAX', '826-BOS-LAX', '827-BOS-LAX', '828-BOS-LAX', '829-BOS-LAX', '830-BOS-LAX', '831-BOS-LAX', '832-BOS-LAX', '833-BOS-LAX', '834-BOS-LAX', '835-BOS-LAX', '836-BOS-LAX', '837-BOS-LAX', '838-BOS-LAX', '839-BOS-LAX', '840-BOS-LAX', '841-BOS-LAX', '842-BOS-LAX', '843-BOS-LAX', '844-BOS-LAX', '845-BOS-LAX', '846-BOS-LAX', '847-BOS-LAX', '848-BOS-LAX', '849-BOS-LAX', '850-BOS-LAX', '851-BOS-LAX', '852-BOS-LAX', '853-BOS-LAX', '854-BOS-LAX', '855-BOS-LAX', '856-BOS-LAX', '857-BOS-LAX', '858-BOS-LAX', '859-BOS-LAX', '860-BOS-LAX', '861-BOS-LAX', '862-BOS-LAX', '863-BOS-LAX', '864-BOS-LAX', '865-BOS-LAX', '866-BOS-LAX', '867-BOS-LAX', '868-BOS-LAX', '869-BOS-LAX', '870-BOS-LAX', '871-BOS-LAX', '872-BOS-LAX', '873-BOS-LAX', '874-BOS-LAX', '875-BOS-LAX', '876-BOS-LAX', '877-BOS-LAX', '878-BOS-LAX', '879-BOS-LAX', '880-BOS-LAX', '881-BOS-LAX', '882-BOS-LAX', '883-BOS-LAX', '884-BOS-LAX', '885-BOS-LAX', '886-BOS-LAX', '887-BOS-LAX', '888-BOS-LAX', '889-BOS-LAX', '890-BOS-LAX', '891-BOS-LAX', '892-BOS-LAX', '893-BOS-LAX', '894-BOS-LAX', '895-BOS-LAX', '896-BOS-LAX', '897-BOS-LAX', '898-BOS-LAX', '899-BOS-LAX', '900-BOS-LAX', '901-BOS-LAX', '902-BOS-LAX', '903-BOS-LAX', '904-BOS-LAX', '905-BOS-LAX', '906-BOS-LAX', '907-BOS-LAX', '908-BOS-LAX', '909-BOS-LAX', '910-BOS-LAX', '911-BOS-LAX', '912-BOS-LAX', '913-BOS-LAX', '914-BOS-LAX', '915-BOS-LAX', '916-BOS-LAX', '917-BOS-LAX', '918-BOS-LAX', '919-BOS-LAX', '920-BOS-LAX', '921-BOS-LAX', '922-BOS-LAX', '923-BOS-LAX', '924-BOS-LAX', '925-BOS-LAX', '926-BOS-LAX', '927-BOS-LAX', '928-BOS-LAX', '929-BOS-LAX', '930-BOS-LAX', '931-BOS-LAX', '932-BOS-LAX', '933-BOS-LAX', '934-BOS-LAX', '935-BOS-LAX', '936-BOS-LAX', '937-BOS-LAX', '938-BOS-LAX', '939-BOS-LAX', '940-BOS-LAX', '941-BOS-LAX', '942-BOS-LAX', '943-BOS-LAX', '944-BOS-LAX', '945-BOS-LAX', '946-BOS-LAX', '947-BOS-LAX', '948-BOS-LAX', '949-BOS-LAX', '950-BOS-LAX', '951-BOS-LAX', '952-BOS-LAX', '953-BOS-LAX', '954-BOS-LAX', '955-BOS-LAX', '956-BOS-LAX', '957-BOS-LAX', '958-BOS-LAX', '959-BOS-LAX', '960-BOS-LAX', '961-BOS-LAX', '962-BOS-LAX', '963-BOS-LAX', '964-BOS-LAX', '965-BOS-LAX', '966-BOS-LAX', '967-BOS-LAX', '968-BOS-LAX', '969-BOS-LAX', '970-BOS-LAX', '971-BOS-LAX', '972-BOS-LAX', '973-BOS-LAX', '974-BOS-LAX', '975-BOS-LAX', '976-BOS-LAX', '977-BOS-LAX', '978-BOS-LAX', '979-BOS-LAX', '980-BOS-LAX', '981-BOS-LAX', '982-BOS-LAX', '983-BOS-LAX', '984-BOS-LAX', '985-BOS-LAX', '986-BOS-LAX', '987-BOS-LAX', '988-BOS-LAX', '989-BOS-LAX', '990-BOS-LAX', '991-BOS-LAX', '992-BOS-LAX', '993-BOS-LAX', '994-BOS-LAX', '995-BOS-LAX', '996-BOS-LAX', '997-BOS-LAX', '998-BOS-LAX', '999-BOS-LAX', '1000-BOS-LAX']

- You might want to click on the left of the output above — this will collapse the output so it doesn't take over your browser window.
- Let's also make sure we have the right number of flights in the variable `flights`:

```
In [10]: print("Number of flights: {0}".format(len(flights)))
```

Number of flights: 2607

Creating a list of airports

- It will also be useful to have a variable `airports` containing a list of all the airports.
- What part of the dataset contains this information?

We need to look at the `ORIGIN` and `DEST` columns of the dataset.

- We can create the list `airports` like this:

```
In [11]: # Convert the ORIGIN and DEST columns from df into sets,
# take their union, convert to a list
airports = list(set(df["ORIGIN"]) | set(df["DEST"]))
```

- Um... what does this do??
- Let's try a smaller example and look at what's going on step-by-step.
- Pretend that A and B defined below are the ORIG and DEST columns from df

```
In [12]: # Pretend that A and B are the ORIG and DEST columns from df
A = ['BWI', 'BWI', 'ORD', 'ORD']
B = ['LAX', 'ORD', 'SFO', 'LAX']
```

- In Python, a **set** is an unordered collection of unique elements, just like the usual mathematical definition.
- `set(A)` takes all entries A and converts it into a set. This eliminates all duplicates within A.
- Same goes for `set(B)`.

```
In [13]: # Convert A and B into sets, print them out
print(set(A))
print(set(B))
```

```
{'BWI', 'ORD'}
{'SFO', 'ORD', 'LAX'}
```

- The `|` operator takes the **union** of the sets, like this:

```
In [14]: # Take the union of set(A) and set(B), print it out
print(set(A) | set(B))
```

```
{'BWI', 'SFO', 'ORD', 'LAX'}
```

- This is almost what we want: we have a list of all the airports, but...
- Sets are similar to lists, but have their own methods. We can turn the set into a list with the function `list()`, like this:

```
In [15]: # Print out the union of set(A) and set(B), converted to a list
print(list(set(A) | set(B)))
```

```
['BWI', 'SFO', 'ORD', 'LAX']
```

- See how that works? That's why `airports` defined above contains a list of all the airports in our dataset.
- Let's make sure everything looks OK with `airports`:

```
In [16]: # Print list of airports
print('Airports: {0}'.format(airports))

# Print number of airports
print('Number of airports: {0}'.format(len(airports)))
```

Airports: ['MFE', 'SLC', 'SNA', 'STX', 'DEN', 'FAT', 'MIA', 'SFO', 'HNL', 'MCO', 'MDT', 'FLL', 'STL', 'IAD', 'TPA', 'MEM']
 Number of airports: 94

Adding nodes with attributes

- Now we're ready to build the shortest path graph. Let's start with an empty directed graph:

```
In [17]: # Create empty NetworkX digraph
G = nx.DiGraph()
```

- Next, let's create a "start" and "end" node.

```
In [18]: # Create start and end nodes
G.add_node("start")
G.add_node("end")
```

- Now, we need to add a node for each flight, or each row of our database.
- We can quickly iterate through the rows of a DataFrame using the `.itertuples()` method:

```
for row in df.itertuples():
    # Put some code here
    # row.COLUMN_NAME = value of column COLUMN_NAME in the current row
```

- So we can add a node for each flight like this:

```
In [19]: # Add a node for each flight
for row in df.itertuples():
    G.add_node(row.FLIGHT, origin=row.ORIGIN, dest=row.DEST, dep_time=row.DEP_TIME, arr_time=row.ARR_TIME,
               distance=row.DISTANCE)
```

- Wait —

```
G.add_node(row.FLIGHT)
```

adds a node whose name is the value of `row.FLIGHT`. What is all the other stuff?

- Remember in the last lesson when we added the "length" attribute to each edge? Like this?

```
G.add_edge(1, 2, length=9)
```

- We can add attributes to nodes as well.
- The code above adds attributes called `origin`, `dest`, `dep_time`, `arr_time`, and `distance` to each node.
 - This will be handy later.
- To access a particular attribute of a node, we write something like this:

```
In [20]: # print the departure time of flight "1-BOS-JFK"
print(G.node["1-BOS-JFK"]["dep_time"])
```

2016-09-01 06:00:00

- The `.number_of_nodes()` method applied to a networkx graph — well, you can guess what it does. Or, you can just try it out:

```
In [21]: # Print number of nodes in G
print(G.number_of_nodes())
```

2609

Adding edges

- Now we can check every pair of flight nodes, and check if we need to add an edge between them.
 - Remember the length of these edges is the negative of the distance of the first flight.
- To add or subtract times, we need to use `pd.to_timedelta()` — [here is the documentation](#).
 - For example, to subtract 30 minutes, we would write
`some_time_variable - pd.to_timedelta(30, unit="m")`
- This might seem awkward, but if you think about it, working with dates and time *is* awkward — you need to keep track of different (non-base-10) units.

```
In [22]: # Iterate through every pair of flight nodes
for first in flights:
    for second in flights:

        # If the first flight arrives where the second flight departs...
        if (G.node[first]["dest"] == G.node[second]["origin"]):

            # And if the first flight arrives 45 minutes before the second flight leaves,
            # add an edge from the first flight to the second
            if (G.node[first]["arr_time"] + pd.to_timedelta(45, unit="m") < G.node[second]["dep_time"]):
                G.add_edge(first, second, length=-G.node[first]["distance"])
```

- Finally, we need to add edges:
 - from the start node to all flights departing from BWI, and
 - from all flights arriving at LAX to the end node.

```
In [23]: # Iterate through all flights
for flight in flights:

    # If the flight departs from BWI,
    # add an edge from start to this flight
    if G.node[flight]["origin"] == "BWI":
        G.add_edge("start", flight, length=0)

    # If the flight arrives at LAX,
```



```
# add an edge from this flight to end
if G.node[flight]["dest"] == "LAX":
    G.add_edge(flight, "end", length=-G.node[flight]["distance"])
```

- Similar to `G.number_of_nodes()`, we can perform a sanity check with our work with `G.number_of_edges()`.

```
In [24]: # Print the number of edges in G
print(G.number_of_edges())
```

158335

Solving the shortest path problem, interpreting the output

- Now that we have our directed graph set up, we can solve for the shortest path from the start node to the end node just like we did in the last lesson:

```
In [25]: # Solve the shortest path problem using Bellman-Ford
length, nodes, negative_cycle = bf.bellman_ford(G, source="start", target="end", weight="length")

# Print output from Bellman-Ford
print("Negative cycle? {0}".format(negative_cycle))
print("Shortest path length: {0}".format(length))
print("Shortest path: {0}".format(nodes))
```

Negative cycle? False

Shortest path length: -8005.0

Shortest path: ['start', '1817-BWI-CLT', '658-CLT-LAS', '1584-LAS-PHX', '694-PHX-HNL', '298-HNL-LAX', 'end']

- What does the output tell us about how to solve Professor Wright's problem?
- The shortest path length gives us the *negative* of the maximum possible total distance Professor Wright can travel on a feasible sequence of flights from BWI to LAX, which in this case, is 8005 miles.
- The nodes in the shortest path give the sequence of flights from BWI to LAX with the maximum possible total distance, which in this case is:
 1. Flight 1817, BWI-CLT
 2. Flight 658, CLT-LAS
 3. Flight 1584, LAS-PHX
 4. Flight 694, PHX-HNL
 5. Flight 298, HNL-LAX

On your own...

Suppose Professor Wright wants to find the longest itinerary from IAD (Washington DC - Dulles) to SAN (San Diego) instead.

In the cell below, write the code that sets up and solves the shortest path formulation for her problem from start to finish.

In the cell after, describe in words what the output from the Bellman-Ford algorithm tells you about how to solve Professor Wright's problem.

```
In [26]: # Import packages
import pandas as pd
import networkx as nx
import bellmanford as bf

# Read csv file into a DataFrame
# Designate departure and arrival time columns as dates
df = pd.read_csv('aa_domestic_flights.csv', parse_dates=['DEP_TIME', 'ARR_TIME'])

# Create empty NetworkX digraph
G = nx.DiGraph()

# Create start and end nodes
G.add_node("start")
G.add_node("end")

# Add a node for each flight
for row in df.itertuples():
    G.add_node(row.FLIGHT, origin=row.ORIGIN, dest=row.DEST, dep_time=row.DEP_TIME, arr_time=row.ARR_TIME,
               distance=row.DISTANCE)

# Iterate through every pair of flight nodes
for first in flights:
    for second in flights:

        # If the first flight arrives where the second flight departs...
        if (G.node[first]["dest"] == G.node[second]["origin"]):

            # And if the first flight arrives 45 minutes before the second flight leaves,
            # add an edge from the first flight to the second
            if (G.node[first]["arr_time"] + pd.to_timedelta(45, unit="m") < G.node[second]["dep_time"]):
                G.add_edge(first, second, length=-G.node[first]["distance"])

# Iterate through all flights
for flight in flights:

    # If the flight departs from IAD,
    # add an edge from start to this flight
    if G.node[flight]["origin"] == "IAD":
        G.add_edge("start", flight, length=0)

    # If the flight arrives at SAN,
    # add an edge from this flight to end
    if G.node[flight]["dest"] == "SAN":
        G.add_edge(flight, "end", length=-G.node[flight]["distance"])

# Solve the shortest path problem using Bellman-Ford
length, nodes, negative_cycle = bf.bellman_ford(G, source="start", target="end", weight="length")
```

```
# Print output from Bellman-Ford
print("Negative cycle? {0}".format(negative_cycle))
print("Shortest path length: {0}".format(length))
print("Shortest path: {0}".format(nodes))
```

Negative cycle? False

Shortest path length: -6005.0

Shortest path: ['start', '2636-IAD-LAX', '2503-LAX-ORD', '2375-ORD-DFW', '435-DFW-SAN', 'end']

- The shortest path length gives us the *negative* of the maximum possible total distance Professor Wright can travel on a feasible sequence of flights from IAD to SAN, which in this case, is 6005 miles.
- The nodes in the shortest path give the sequence of flights from IAD to SAN with the maximum possible total distance, which in this case is:
 1. Flight 2636, IAD-LAX
 2. Flight 2503, LAX-ORD
 3. Flight 2375, ORD-DFW
 4. Flight 435, DFW-SAN