Interstate Trade Relations

You are not logged in.

Please Log In for full access to the web site.
Note that this link will take you to an external site (`https://shimmer.mit.edu`) to authenticate, and then you will be redirected back to this page.

Table of Contents

1) Preparation

This lab assumes you have Python 3.6 or later installed on your machine (3.11 recommended).

The following file contains code and other resources as a starting point for this practice exercise: ZIP FOLDER

> This practice exercise is optional and ungraded but it is designed to help you prepare for this week's Bacon Number lab.

These problems are also a good way to practice writing code with good style. Upon submitting a correct solution to a problem, you will get access to staff solutions and explanations for each problem.

While we encourage students to collaborate on the concept questions, please refrain from collaborating on the coding problems except with staff members. To allow all students the benefit of working through the problems individually, the course academic integrity policy applies to your solution and the official solution code-- meaning no sharing or posting of answers allowed, especially on public or shared internet spaces.

2) Introduction

You have been recently appointed as the Chief Trade Officer (CTO) of the U.S. Commerce Department. Your job as CTO is to organize all of the trade transactions between different states and determine what states have the strongest trade relations.

To accomplish this task you are given a database of real U.S. trade data[1] represented as a list of lists of the following form

['ORIGIN_STATE', 'DESTINATION_STATE', 'COMMODITY', 'MODE', 'VALUE']

For example, the transactions in our tiny_transaction database records show:

```
[['MA', 'CA', 'Grains', 'Truck', 47097],
 ['MA', 'ND', 'Wood', 'Truck', 12781],
 ['ND', 'MA', 'Grains', 'Truck', 12087],
 ['WY', 'ND', 'Fuels', 'Rail', 128633],
 ['ND', 'SC', 'Grains', 'Truck', 1456],
 ['SC', 'MT', 'Wood', 'Air', 41],
 ['MA', 'MA', 'Wood', 'Truck', 10606],
 ['MT', 'WY', 'Fuels', 'Truck', 584],
 ['CA', 'MA', 'Grains', 'Truck', 16463],
 ['CA', 'ND', 'Grains', 'Truck', 12462],
 ['ND', 'CA', 'Wood', 'Truck', 23],
 ['MT', 'ND', 'Grains', 'Truck', 23690],
 ['CA', 'MA', 'Wood', 'Truck', 1181],
]
```

You can also view this by running the `practice.py` file which conveniently already has code that loads and prints out the transaction entries in the tiny database.

3) Visualizing the Database </section

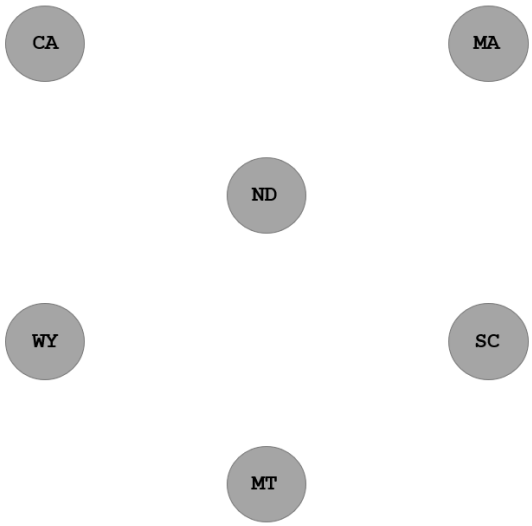
While the database contains a lot of useful information, for the purpose of these exercises we will focus only on the origin state and the destination state. Before we dive into writing code, let's think about how we can visualize[2] the trades in the database:

Check Yourself 1:

Fill in the map below to show the origin and destination of each transaction in the tiny database above:

Check Yourself 2:

Another way to visualize this map (that doesn't involve exact geography) is by representing the states as circles (nodes) and drawing pointed arrows (edges) to represent the direction of the transaction. Fill in the missing lines in the diagram below:



3.1) Programming Exercise Instructions

The following sections will contain programming exercises and related concept questions. For each programming exericse, we recommend the following approach:

1. Read the function docstring.
2. Answer the concept questions by hand (you should be able to answer these without writing code!)
3. Write the function in the provided `practice.py` file.
4. Once your code in `practice.py` passes the test case associated with the function in the `test.py` file, paste it into the code submission box. Submitting a fully correct solution to the code submission box will unlock the ability to view the staff solutions.

If you have any questions about the code solutions or want to verify your answers to the concept questions, we highly encourage you to check with your peers and/or ask a staff member!

Transforming the Data

Now that we have thought about different ways to visually represent the database, how can we use python to simplify our database to just represent the origin and destination of the various transactions?

3.2) Transformation 1: list of lists

```
def transform_list_pairs(database):
    """
```

```
    Transforms the database into a more concise format that only includes
    transactions where the origin is different from the destination.

    Parameters:
        * database (list) : a list where each element is a list of the
form
            [
             origin_state (str),
             destination_state (str),
             item (str),
             transportation (str),
             value (int)
            ]

    Returns:
        A list where each element is a list of the form
            [origin_state (str), destination_state (str)].
    """
    pass
```

Check Yourself 3:

A. What should the length of the list returned by `transform_list_pairs(tiny_database)` be?

B. What should `transform_list_pairs(tiny_database)` return?

C. Paste your definition of transform_list_pairs below: (click in the box, use ctrl+a to highlight text, ctrl+v to paste)

```
1 def transform_list_pairs(database):
2     pass # Your code here
```

3.3) Transformation 2: set of tuples

```
def transform_set_pairs(database):
    """
    Transforms the database into a more concise set that only includes
    transactions where the origin is different from the destination.

    Parameters:
        * database (list) : a list where each element is a list of the
```

```
form
            [
             origin_state (str),
             destination_state (str),
             item (str),
             transportation (str),
             value (int),
            ]

    Returns:
        A set where each element is a tuple of the form
            (origin_state (str), destination_state (str)).
    """
    pass
```

Check Yourself 4:

A. What should the length of the set returned by `transform_set_pairs(tiny_database)` be?

B. What would `transform_set_pairs(tiny_database)` return?

C. Paste your definition of `transform_set_pairs` below: (click in the box, use ctrl+a to highlight text, ctrl+v to paste)

```
1 def transform_set_pairs(database):
2     pass # Your code here
```

3.4) Transformation 3: dictionary of lists

```
def transform_dict_list(database):
    """
    Transforms the database into a dictionary mapping origin_states to a
    list of destination states. Only includes transactions where the
origin
    is different from the destination (note destinations may be
repeated.)
    Only includes origin states with at least one destination.

    Parameters:
        * database (list) : a list where each element is a list of the
form
```

```
            [
             origin_state (str),
             destination_state (str),
             item (str),
             transportation (str),
             value (int),
            ]

    Returns:
        A dictionary with keys that are origin_states and values that are
a
        list of strings of all destination states.
            {origin_state (str) : [destination_state (str), ...]}
    """
    pass
```

Check Yourself 5:

A. How many keys are in the dictionary that is returned by `transform_dict_list(tiny_database)`?

B. What should `transform_dict_list(tiny_database)` return?

C. Paste your definition of `transform_dict_list` below: (click in the box, use ctrl+a to highlight text, ctrl+v to paste)

```
1 def transform_dict_list(database):
2     pass # Your code here
```

3.5) Transformation 4: dictionary of sets

```
def transform_dict_set(database):
    """
    Transforms the database into a dictionary mapping origin_states to a
    list of destination states. Only includes transactions where the
origin
    is different from the destination. Only includes origin states with
at least
    one destination.

    Parameters:
        * database (list) : a list where each element is a list of the
```

```
form...

    Returns:
        A dictionary with keys that are origin_states and values that are
a
        set of strings of all destination states.
            {origin_state (str) : {destination_state (str), ...}}
    """
    pass
```

Check Yourself 6:

A. What would `transform_dict_set(tiny_database)` return?

B. Paste your definition of `transform_dict_set` below: (click in the box, use ctrl+a to highlight text, ctrl+v to paste)

```
1 def transform_dict_set(database):
2     pass # Your code here
```

3.6) Transformation Concept Questions

List of Transformation Functions (for reference):

1. `transform_list_pairs`
2. `transform_set_pairs`
3. `transform_dict_list`
4. `transform_dict_set`

Check Yourself 7:

A. Which transformation function(s) would return a data structure that would allow you to draw the arrows in the original visualization exercises without additional information?

B. Which transformation function(s) would allow you to determine from any database *exactly* how many transactions occurred between any two different states like California and Massachusetts? Why?

C. Which transformation function returns a data structure that would likely be the fastest at checking whether Kentucky ever shipped goods to Hawaii? Why?

D. Which transformation function returns a data structure that would likely be the slowest at checking whether Kentucky ever shipped goods to Hawaii? Why?

4) Trade Relationships

4.1) One-way Dictionary

```
def oneway_relations_dict(database):
    """
    Create dictionary representing only the oneway trade relationships
    in the database. A oneway trade relationship is defined as a pair of
    states where state A shipped products to state B, but state B never
    shipped anything to state A. Only states with at least one oneway
trade
    relationship should be included.

    Parameters:
        * database (list) : a list where each element is a list of the
form...

    Returns:
        A dictionary with keys that are origin_states and values that are
a
        set of strings of all oneway destination states.
            {origin_state (str) : {destination_state (str), ...}}
    """
    pass
```

Check Yourself 8:

A. Draw the oneway relations in the tiny database using the nodes / edges method used in question Check Yourself 2.

B. How many keys should be in the dictionary that `oneway_relations_dict(tiny_database)` returns?

C. What should `oneway_relations_dict(tiny_database)` return?

D. What function could be used to efficiently determine whether two states like MT and WY have a oneway trade relationship?

E. Paste your definition of `oneway_relations_dict` (and any helper functions) below: (click in the box, use ctrl+a to highlight text, ctrl+v to paste)

```
def oneway_relations_dict(database):
    pass # Your code here
```

4.2) One-way Trade Routes

```
def oneway_loop(database, state):
    """
    Finds a path representing the smallest number of states with oneway
    trade relationships that form a loop starting and ending with the
given state.

    Parameters:
        * database (list) : a list where each element is a list of the
form...
        * state (str) : the desired start and end state

    Returns:
        A list of state strings representing the path, or None if there
is no such path.

    Notes:
        * In the tiny DB, the shortest oneway loop for ND would be [ND,
SC, MT, ND]
          (note [ND, SC, MT, WY, ND] is a valid oneway loop but not the
shortest loop.)
        * In the tiny DB, there is no oneway loop for MA so this function
should return None.
          (MA->MA is not a valid path and [MA, ND, MA] represents a trade
alliance,
          not a oneway path.)
    """
    pass
```

Paste your definition of `oneway_loop` (and any helper functions) below: (click in the box, use ctrl+a to highlight text, ctrl+v to paste)

```
def oneway_loop(database, state):
    pass # your code here
```

Footnotes

[1] The databases we use in this assignment are a processed subset of the 2012 Commodity Flow Survey Dataset published by the U.S. Census Bureau.

[2] This image is in the public domain.