TER report

1. Introduction

In the previous algorithm course, we learned the Bellman-Ford algorithm, which is one of the most effective algorithms for calculating the shortest path. Based on the algorithm, we can extend it and calculate the shortest path and the second shortest path at the same time. What’s more, we can also apply the algorithm to real life, such as calculating the cheapest route among multiple cities. First, we can find the shortest route based on the Bellman-Ford algorithm. When a section of the cheapest route is destroyed, but the second cheapest route is not damaged, in this case we can make a choice quickly with no need for recalculating all routes again.

My demonstration uses a lot of widgets to accept and respond to user operations. By using the *ipweidget* module, it can not only show each step of the Bellman-Ford algorithm in detail, but also apply it to real-world scenarios to solve real-world problems.

1. Introduction of Jupyter and widget
	1. What is Jupyter :

The essence of Jupyter Notebook is a Web application that facilitates the creation and sharing of literary program documents, supports real-time code, mathematical equations, visualization and markdown. Uses include: data cleansing and conversion, numerical simulation, statistical modeling, machine learning, etc.

* 1. What is widget:

Widgets are elements of a graphical user interface (GUI) that are used to display information or provide users with a specific way to interact with the operating system or applications

Widgets include icons, drop-down menus, buttons, selection boxes, progress indicators, on / off markers, slider, windows, window edges (which allows you to resize the window), toggle buttons, forms, and many others Devices and invitations to accept and respond to user actions.

* 1. *Ipwidgets* module:

The *ipywidgets* module can realize the interactive control operation of jupyter notebook. It is web-based, and this control can only be used in jupyter notebooks. Users can operate on the data and visualize the changes in the data. In the process of using, it can make the process of the algorithm more front-line understandable

1. Bellman-Ford algorithm
	1. Brief introduction

Traditional Bellman-Ford algorithm:

Given a weighted directed graph G = (V, E) with source s and weight function w: E → R, the Bellman-Ford algorithm returns a Boolean value that indicates whether there is a negative weight loop. If there is such a loop, the algorithm indicates that there is no solution. If there is no such cycle, the algorithm will generate the shortest path and its weight.

The algorithm uses relaxation to gradually reduce the estimate d [v] of the shortest path weight from the source s to each vertex v∈V until the actual shortest path weight δ (s, v) is obtained. The algorithm returns TRUE if and only if the graph does not contain a negative weight loop that can be reached from the source.



* 1. Applicable conditions

i. The shortest path of a single source point: the shortest path from a certain source point s to all other vertices. This is consistent with our application scenario. We can only start from one city, but the target city can be changed.

ii. The weight of the edge can be positive or negative, but there should be no negative loop (the weight of the loop is negative). But here because we want to simulate real-life routes, and in real life, the price of air tickets is not negative, so we do not consider the case where the weight is negative

 I made some modifications on the traditional Bellman-Ford algorithm. First, combined with the application scenario, there is no negative loop in the figure, so there is no need to determine the negative loop.

When relaxing the edge, if d [v]> d [u] + w (u, v), assign the shortest path (d) to the second shortest path (d2), and update the shortest path at the same time ( d). The second short path of a node may also come from the second short path of other nodes, that is, d2 [v]> d2 [u] + w (u, v).

The source of the second short path should be clearly marked, because when printing the complete path later (to find the starting point from the end point according to the precursor, so as to obtain the complete path), it is necessary to judge that the second path comes from the shortest or second shortest path of the previous point.

1. Main functions of demo
	1. Select departure city and arrival city

In this interface, the user can select a departure city and an arrival city from the drop-down list. Users can select transit cities and add them one by one, or join all cities at once. Then click confirm to confirm. After the city is selected, all the connecting routes among the selected cities will be automatically added to the map.

* 1. Show the implementation process of bellman-Ford algorithm

The slider in the upper left corner is used to control each step of the algorithm, that is, which city, which route is currently processed, and the changes made. The opaque marker indicates the city currently being processed. The green route represents all successors of the current city. The red edge indicates the route being processed. If the shortest path and the second shortest path of the subsequent vertex are changed, then the color of the table corresponding to that row will change, and its value will be changed.

* 1. Show the shortest and second shortest path

Click to show details: the screen shows the total weight of the shortest path and the second shortest path of departure city and arrival city

Click the Show The Shortest Path button: all cities and details passed by the shortest path will be printed out, and the color of the corresponding path will turn orange in the figure

Click to display the second short path button: Same as above, the path color is blue

* 1. Change arrival city

Reselect arrival city in the drop-down list, and confirm. Bellman-Ford algorithm will be rerun for recalculating.

* 1. Delete route

This function is mainly used to simulate the cancellation of the route. When a route is cancelled, it may have an impact on the shortest route, but it has no effect on the second shortest route. In this case, the interface can be directly returned to the user for the optional path without calculation.

Click the "Delete Route" button: it will jump to the corresponding interface, where there is a checkbox list corresponding to all the routes in the figure. Remove the route you want to remove by ticking the line

If the removed route is in the first short path, but not in the second short path, then the page will prompt the user that the second short path can still be selected.



If the removed routes belong to the shortest and second shortest route, then the route needs to be recalculated.



* 1. Add route

By default, since all routes among the selected cities are automatically added to the map, the set of routes that can be added is empty.

However, after the user deletes the route, the deleted route will be added to the addable route. Tick the route you want to add and confirm. The system will automatically complete the relevant calculations

* 1. Restart

Click the restart button. The program will automatically jump to the first interface. The user can now add and delete cities.

1. Implementation process
	1. Select departure city and arrival city

Define sets of all cities and all routes.

The arrival city and departure city will be stored in the dictionary *c\_arrival* and *c\_departure* respectively

Three matrices are used to express the relationship among cities: the first matrix represents the weight of routes between different cities (m1), and the second matrix represents the routes between cities (m2)

Since the routes between cities are always coming and going, and the flight prices are the same, so all three matrices are symmetrical.

After selecting the arrival city, departure city and transit city, the above two matrices are cut according to the ids of all selected cities. Thus, two sub-matrices are obtained. After that, all the algorithms are based on these two sub-matrices

* 1. Mark all cities and routes on the map:

Create a marker list based on the selected city. Inside, each city is a marker, initializing its location.

According to the selected city, Create a new class (Point) for each city separately and complete the initialization of its id, name, predecessor, shortest path, etc. and then create a city list (city\_list), which is consist of class Points created above.

And according to the properties in the Point class, create a table by using le code

marker.popup = tab1

to set the content of the popup when click on the popup

According to the selected route, create a polyline list, which stores all routes.

* 1. Run the algorithm

In the operation of the algorithm, use *ite* to mark how many times of calculations have been performed. This will be used for the creation of sliders to facilitate the use of sliders to control the execution of each step of the viewing algorithm.

Through the function ineract\_slider () to achieve the processing when the slider is updated. In the function interact\_slider (), the opacity of the marker being processed is updated, and the table in its popup is updated

In order to complete the interactive function, there are several methods:

1. Interact

The Interact function automatically creates UI controls. It is the easiest way to use IPython widgets.

For example: create a slider interactive control: interact (f, x = 10)

Create interactive controls for checkbox: interact (f, x = True)

2. Interactive

The Interactive function returns an instance of the widget without displaying the widget. We can use the display function to control whether the widget is displayed

3. @interact:

Using function annotations to specify interact controls for a function is now deprecated and will be removed in a future version of ipywidgets. (# 1292 )

4. Interact\_output:

It does not automatically create widget. You can create an instance of the widget and pass it to interactive\_output as a parameter. You can also control how the widget is laid out. It's more powerful

But when the program has multiple interfaces, it is necessary to display or close some interfaces to achieve interface switching, this time is not suitable to use *interact*, because we cannot get the slider reference, so we cannot close it. so, I used *interactive\_output*

* 1. Delete route

In the sub-matrix (m2) for recording routes, find the row and column (i, j) of the route to be deleted, and then record the route m2 [i] [j] and its corresponding weight in the dictionary *ll\_delete*. Then update these two matrices: m1 [i] [j] =NULL, m2 [i] [j] = INFINI

* 1. Add route

Describe a checkbox list for all values in the dictionary *ll\_delete*. For the selected routes to be added, change the values of the corresponding rows and columns in the two matrices in the same way as above

1. Comparison of different modules: networkx, bqplot, basemap
	1. *Networkx:*

Not only can *networkx* create a variety of directed graphs, undirected graphs and multiple graphs, it also has built-in many standard graph theory algorithms.

And using *networkx* can also store the network in standardized and non-standardized data formats, generate a variety of random networks and classic networks, analyze network structure, establish network models, design new network algorithms, and perform network drawing, etc.

But when the modul *networkx* and *ipywidget* are combined, every time the slider is used to update the interface, there will be a very obvious stuck. This obviously does not meet our needs, so I tried module *bqplot* .

* 1. Bqplt:

Compared to *networkx, bqplot* is more flexible.

*Bqplot* has a widget *Map*, which can combine graphs and maps(graphs can be displayed above maps). And the map can be zoomed in and out smoothly with the mouse. You can select different countries and respond very flexibly.

*B*ut it is impossible to synchronize the position in the map and the map. In other words, when we zoom in on the map, the graph is fixed.

* 1. Basemap:

It is an extension of matplotlib, so it has all the functions to create data visualizations, and adds geographic projection and some data sets so that coastlines, countries, etc. can be drawn directly from the library. And this module can also create 3D maps.

Using this module can draw points and lines on the map, but the drawing method is too rigid, and each update also takes longer time, so after a few attempts to give up this module.

Compared with several other modules, Ipyleaflet is the best choice to meet the combination of graphs and maps, and to ensure smooth process. Each module has its own distinctive and prominent role, here is just a brief exploration of the graph and map parts. I just chose the module that best suits my idea

1. Difficulties:

The structure of stored data is not so uniform and clear. Although some classes have been created, the readability of the code is still not high. It needs more improvement.

1. Improvement

1. the latitude and longitude of the cities in the collection are manually determined. I hope that in the next version, the user can click on the map, and then the program can automatically obtain its location. This may be possible through the function:

feature\_layer.on\_click (interact\_click) and geoJSON format data to achieve.

2. There is no such properity that can change the shape of polyling in *ipyleaflet*, so now all the routes in the figure are without arrows and bidirectional by default. One-way arrows and route weights can be achieved by adding icons. This can make the whole picture more intuitive.

1. Conclusion

this is my first time that I have used jupyter. Through this TER,I explored multiple modules for drawing and maps. In a step-by-step attempt to achieve my ideas.

1. Reference