1. 简介

在之前的算法课程中，我们学习了Bellman-Ford算法，它是计算最短路径的非常有效的算法之一。在该算法的基础上，我们可以进行延伸，同时算得最短路径和第二短路径。同时，我们还可以将算法运用到现实生活中，比如在多个城市之间计算最便宜的航线。首先，我们可以根据Bellman-Ford算法求得最短航线。当最便宜航线中的每一段受到毁坏，但是第二便宜的航线并未受到损害，这种情况下我们可以很快的做出选择。

我的演示大量使用了widget来接受和响应用户的操作通过使用ipweidget包，既能详细的展示Bellman-Ford算法每一个步骤，又能应用到现实的情景中解决一下现实中的问题。

1. 什么是jupyter， 什么是widget：

Jupyter：

Jupyter Notebook 的本质是一个 Web 应用程序，便于创建和共享文学化程序文档，支持实时代码，数学方程，可视化和 [markdown](https://baike.baidu.com/item/markdown/3245829)。 用途包括：数据清理和转换，数值模拟，统计建模，机器学习等等

Widgets:  
小部件是图形用户界面（[GUI](https://searchwindevelopment.techtarget.com/definition/GUI)）的元素，用于显示信息或为用户提供与[操作系统](https://whatis.techtarget.com/definition/operating-system-OS)或[应用程序](https://searchsoftwarequality.techtarget.com/definition/application)进行交互的特定方式

小部件包括图标，下拉菜单，按钮，选择框，进度指示器，开/关标记，滚动条，窗口，窗口边缘（可让您调整窗口大小），切换按钮，表单以及许多其他用于显示信息的设备以及邀请，接受和响应用户的操作。

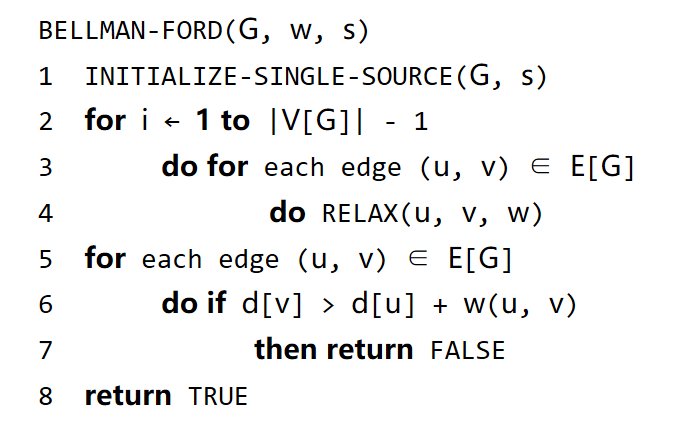
Ipwidgets:

ipywidgets 包可以实现 jupyter notebook 笔记本的交互式控件操作。它是基于网页的，并且这个控件只能再jupyter notebook中使用。用户可以对数据进行操作，并可视化数据中的更改。在使用过程中，可以使算法的过程更加的前线易懂

1. Bellman-Ford算法
   1. 简介：

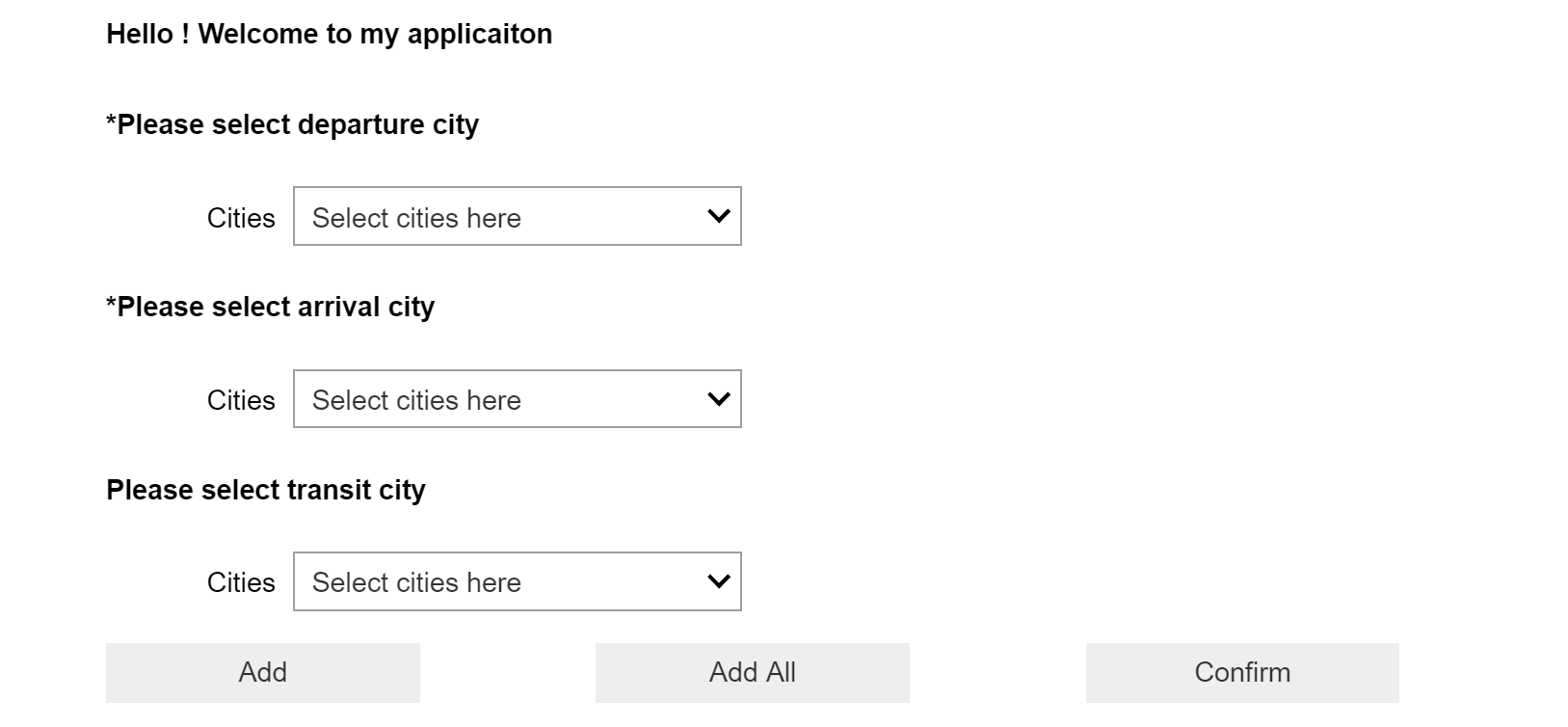
传统的Bellman-Ford算法：给定带有源s和权重函数w：E→R的加权有向图G =（V，E），Bellman-Ford算法返回一个布尔值，该布尔值指示是否存在一个负权重循环，该负权重循环可以从 资源。 如果存在这样的循环，则该算法指示不存在解决方案。 如果没有这样的循环，该算法将产生最短的路径及其权重。

该算法使用松弛，逐渐减小从源s到每个顶点v∈V的最短路径权重的估计d [v]，直到获得实际的最短路径权重δ（s，v）。 当且仅当图形不包含可从源到达的负权重循环时，算法才返回TRUE。



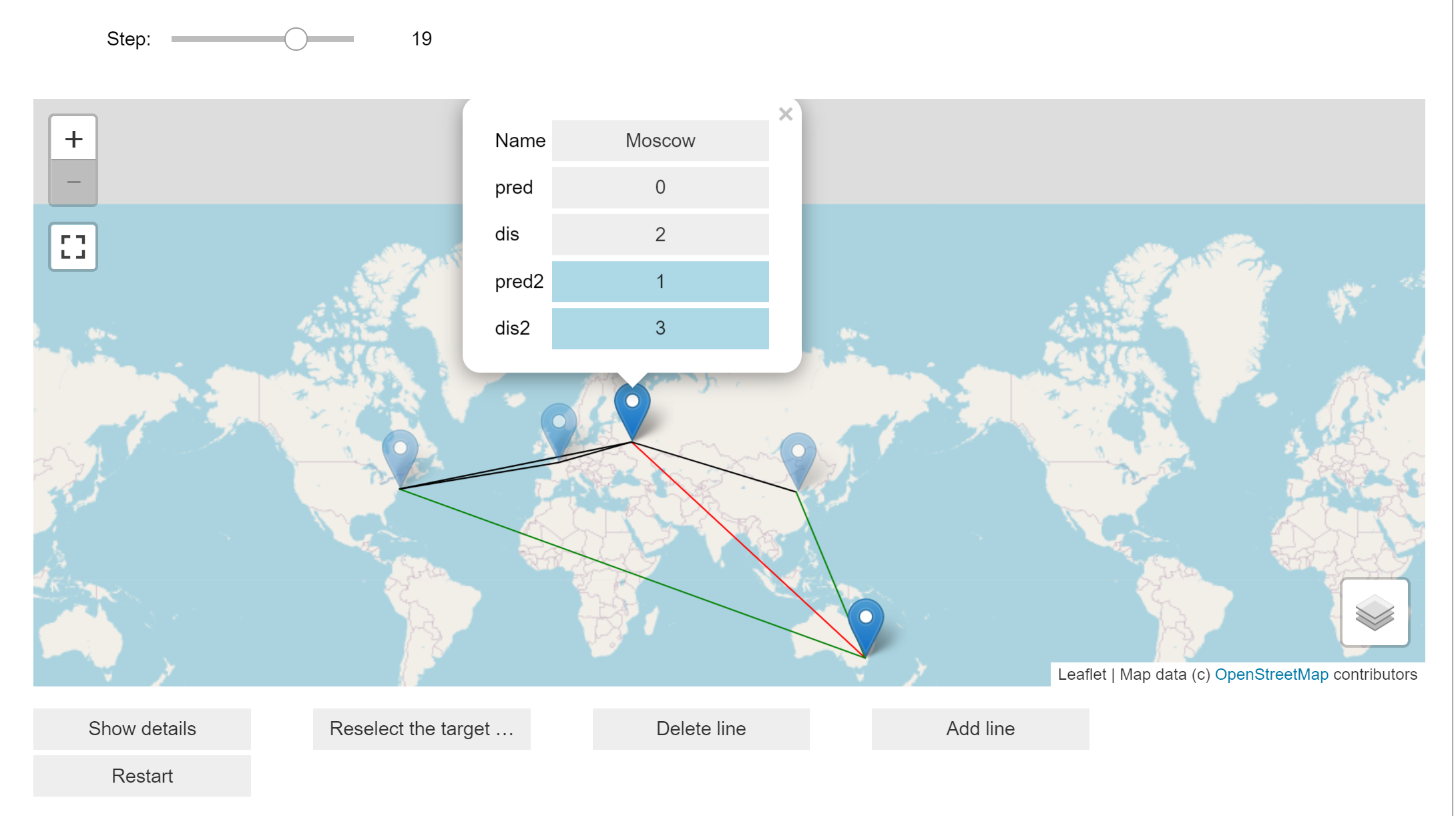
* 1. 适用条件：
     1. 单源点的最短路径：从某个源点s，到其他所有顶点的最短路径。这符合我们的应用场景，我们只能从一个城市出发，但是目标城市是可以进行变更的。
     2. 边的权值可以为正，可以为负，但是不可以存在负环（回路的权值和为负）。但是在这里因为要模拟现实生活中的航线，而现实生活中，飞机票的价格是没有负数的，所以我们不考虑权重为负的情况
  2. 我在传统的Bellman-Ford算法上面做了一些修改。首先，与应用场景相结合，图中不存在负环，因此不需要进行负环的判定。在对边进行松弛的时候，如果d[v]>d[u]+w(u,v), 将此时的最短路径(d)赋值给第二短路径(d2)，同时更新最短路径(d)的值。节点的第二短路径还可能来自于其他节点的第二短路径，即，d2[v]>d2[u]+w(u,v)。第二短路径的来源要标记清楚，因为之后在打印完整的路径的时候（要能从重点根据前驱找到起点，从而得到完整的路径），需要判断第二路径来自于前一个点的第一短路径还是第二短路径。

1. Demo的主要功能
   1. 选择出发城市和目标城市：



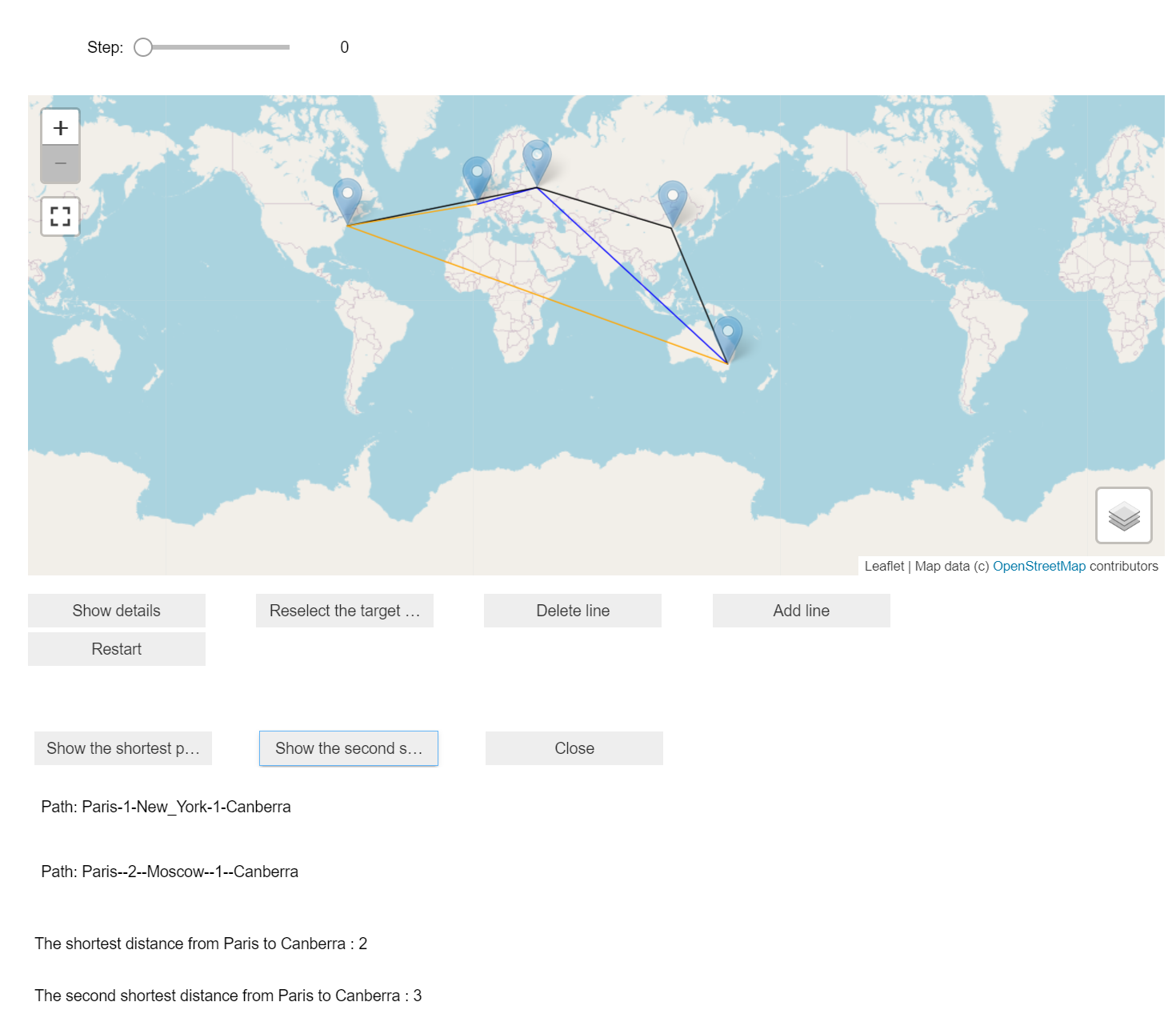
在这个界面，用户可以从下拉列表中选择一个出发城市和一个到达城市。中转城市用户可以选择城市然后逐个添加，或者一次性全部加入所有城市。之后点击confirm确认。当城市选定以后，所有被选择城市之间相连的航线将被自动添加到图中。

* 1. 展示bellman-Ford算法的实现过程



左上角的slider用来控制算法的每一个步骤，即，当前处理到哪一个城市，哪一条航线，以及做出的改变。不透明的marker表示当前处理的城市。绿色的边表示当前城市的所有的后继。红色的边表示正在处理的航线。如果后继顶点的最短路径和第二短路径有所改变，那么它对应那行的表格的颜色会改变，并且改变其数值。

* 1. 显示最短路径和第二短路径



点击显示细节：屏幕显示城市A-B的最短路径和第二短路径的总权重

点击显示最短路径按钮：最短路径所经过的所有城市以及细节会被打印出来，对应路径的颜色会在图中变成橙色

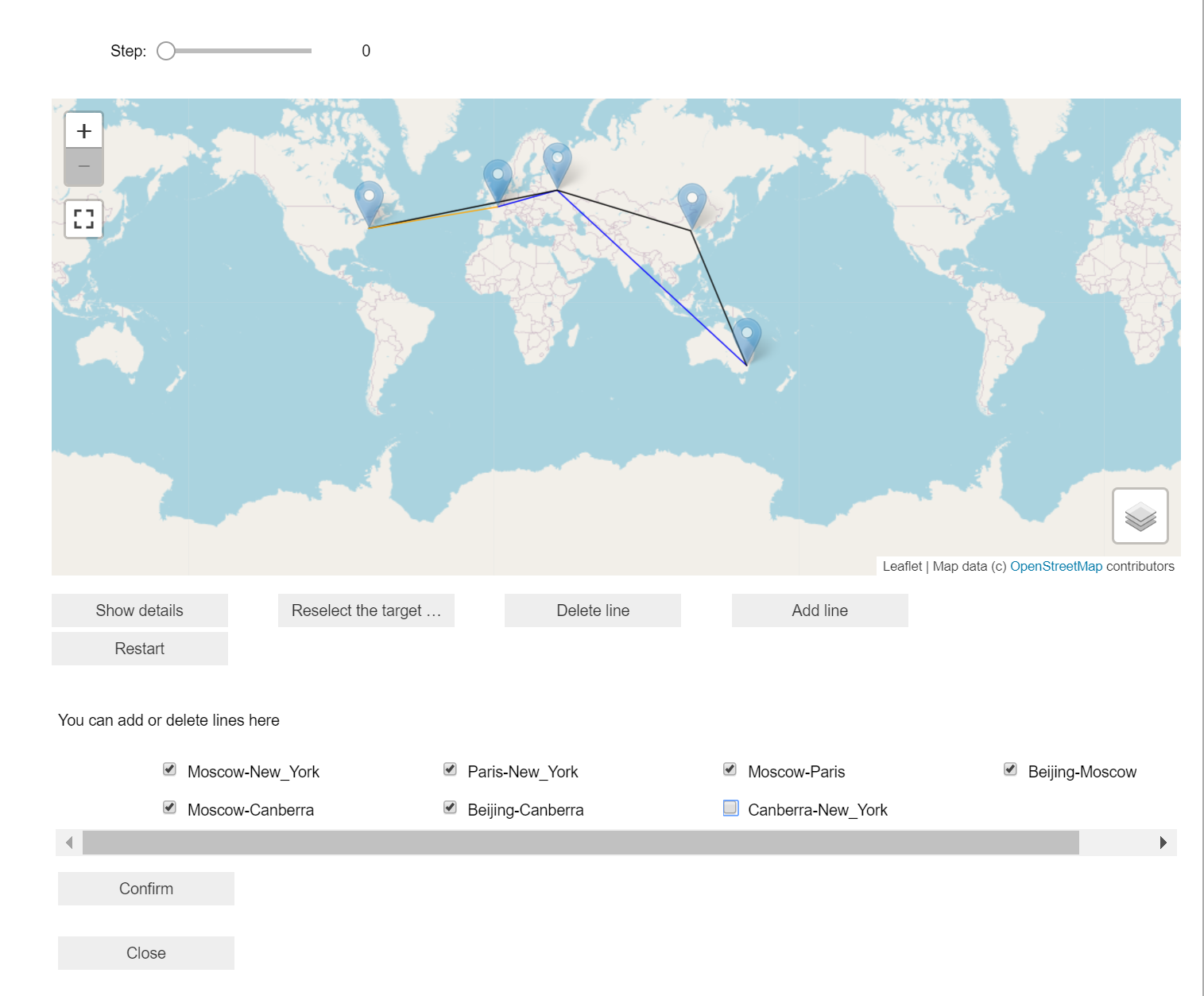
点击显示第二短路径按钮：同上，路径颜色为蓝色

* 1. 更改目标城市



在下拉菜单中重新选择目标城市，并确认。

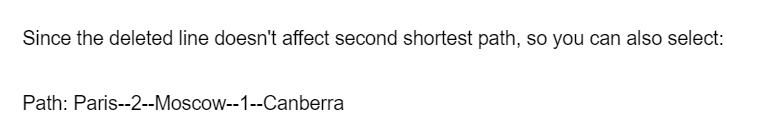
* 1. 删除航线



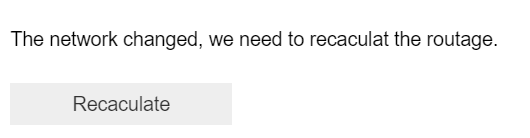
这个功能主要用来模拟航线的取消。当某条航线取消以后，可能对最短路径产生影响，但是对第二短路径并没有影响。在这种情况下，界面可不再经过计算直接返回给用户可选路径。

点击“删除航线”按钮：将跳转到对应界面，那里有所有图中的航线对应的checkbox列表。通过勾选线来去掉想去掉的线

如果去掉的航线在第一短路径，但是没有在第二短路径里面，那么页面将提示用户仍旧可以选择第二短路径。



如果去掉的航线们属于最短和第二短路径，那么需要对路径进行重新的计算。



* 1. 添加航线

在默认情况下，由于被选中城市之间的所有航线都自动加到了图中，所以可以添加的航线的集合是空的。但是当用户在删除航线以后，这些被删除的航线会被添加到可添加的航线中。勾选想要添加的航线，然后确认。系统将自动完成相关计算

* 1. 重启

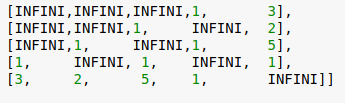
点击重新开始按钮。程序将会自动跳转到第一个界面。用户此时可以对城市进行添加和删除。

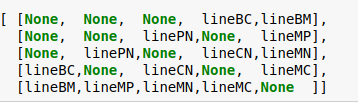
1. 实现过程（矩阵分割）
   1. 选择出发城市和目标城市：

分别定义所有的城市和所有的线。

达到城市和出发城市将分别被存储在字典c\_arrival和c\_departure

用三个矩阵来表示他们之间的关系：第二个矩阵表示不同城市之间航线的权重(m1)，第三个矩阵表示城市间的航线“都是谁“(m2)

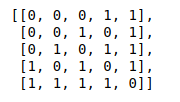




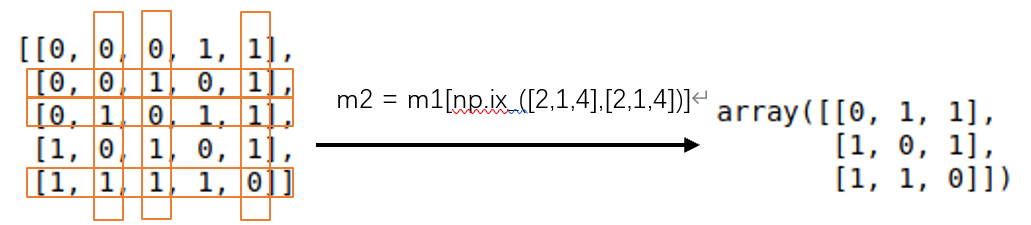
由于城市之间的航线都是有来必有往，且航班价格一样，所以三个矩阵都是对称的。

当选择了目标城市，出发城市以及中转城市之后，根据所有被选择城市的id来对上面的三个矩阵进行切割。从而获得三个子矩阵。之后所有的算法都是根据这三个子矩阵来做的

加图：

m2 = m1[np.ix\_([2,1,4],[2,1,4])]



* 1. 在图上标记所有的城市和航线：

根据被选择的城市，创建一个marker list。在里面，每一个城市就是一个marker，初始化它的位置。

根据被选择的城市，创建一个城市组成的list(city\_list)。分别对每一个城市new一个类（Point），并完成对其id，name，前驱，最短路径等的初始化。

并且根据Point这个类里面的属性，创建一个表格. 通过命令marker.popup = tab1

来使点开以后popup的内容为表格tab1

根据被选择的航线，创建一个polyline的list，里面存储了所有的航线。

* 1. 运行算法：

在算法的运行中，用ite来标记一共进行了多少次的计算。这将用于slider的创建，方便用slider来控制查看算法每一步的执行。

通过h桉树ineract\_slider（）来实现slider更新的时候的处理

在函数interact\_slider()中，正在处理的marker的不透明度被更新，它的popup中的表格被更新

为了完成交互功能，有几种方法：

1. Interact

Interact函数会自动创建UI控件，他是使用IPython小部件的最简单的方法。

比如：创建slider的交互控件： interact(f, x=10)

创建checkbox的交互控件： interact(f, x=True)

1. Interactive

Interactive函数返回一个widget的实例，而不会显示widget。这是我们可以使用display函数控制是否显示widget

1. @interact：使用功能注释来指定交互控件的功能在7.0以后的版本中已经被淘汰Using function annotations to specify interact controls for a function is now deprecated and will be removed in a future version of ipywidgets. (#1292)
2. Interact\_output:

不会自动小部件。你可以创建一个小部件的实例，然后以参数的方式将其传递给interactive\_output. 还可以控制小部件如何布局。它的功能更强大

但是当程序拥有多个界面时，需要显示或者关闭某些界面来实现界面切换，这个时候不适合用interact，因为我们无法得到slider的引用，从而无法关闭它，所以我使用了interactive\_output

* 1. 删除航线

在记录航线的子矩阵(m3)中，找到要删除的航线所在的行和列(i,j)，然后将航线m2[i][j]以及其对应的权重记录在字典ll\_delete里面。然后将m[i][j]改为NULL。M1[i][j]=0, m2[i][j] = INFINI

* 1. 添加航线

针对字典ll\_delete里面的所有值描绘出一个checkbox list。对于勾选的要添加的航线，与上同样的方法，更改三个矩阵中对应行和列的值

1. 不同module的比较：networkx, bqplot, basemap

Networkx：

networkx不但可以创建各种有向图，无向图和多重图，它还内置了许多标准的图论算法。

并且利用networkx还可以以标准化和非标准化的数据格式存储网络、

生成多种随机网络和经典网络、分析网络结构、建立网络模型、设计新的网络算法、

进行网络绘制等。

但是当modul networkx和ipywidget相结合的时候，每次利用slider更新界面，会有非常明显的卡顿。这显然是不符合我们的需求的，所以我又对module bqplot进行了尝试。

Bqplt:

比较于networkx，bqplot更灵活。

bqplotli内拥有控件Map，能够结合图和地图。并且地图可以用鼠标平滑的进行放大缩小。可以选中不同的国家，做出响应等都是很灵活的。

bqplt 可以和basemap结合使用，但是在无法同步在图中和地图中的位置。也就是说，当我们放大地图的时候，用bqplot画出来的图是固定不动的。

Basemap:

它是matplotlib的扩展程序，因此具有创建数据可视化的所有功能，并添加了地理投影和一些数据集，以便可以直接从库中绘制海岸线，国家等。并且可以创建3D地图

使用这个module可以在地图上绘制点和线，但是绘制的方式过于死板，并且每次更新也需要更久的时间，所以在经过一些尝试以后放弃了这个module

相比于其他的几个module，Ipyleaflet是满足图和地图的结合，并且能保证过程流畅的最好的选择。每个module都有自己与众不同且突出的作用，这里只是对图和地图的部分进行了浅显的探索。

1. 遇到的困难

存储数据的结构不是那么的统一和清晰。虽然创建了一些类，但是代码的可读性仍旧不高。需要更多的改善。

1. 改进和提高
2. 现在城市的集合是手动确定的城市的经纬度。我希望在下一个版本中，用户可以点击地图，然后程序可以自动获取其位置。这可能可以通过函数：

feature\_layer.on\_click（interact\_click）和geoJSON格式的数据来实现。

1. Ipyleaflet自带的polyline并没有控制其形状的属性，所以现在图中所有的航线都是没有箭头，默认双向的。单向箭头和航线权重是可以根据通过添加图标来实现的。这样可以使整个图更直观
2. 总结：

通过这次的ter，我第一使用了jupyter，并且探索了多个用来画图和地图的module。在一步步的尝试中实现了自己的构想。

1. 引用

Marker:少打了一个r

矩阵那里两个矩阵可以合并