**一、实验题目**

最短路径求解

**二、实验目的：**

使用plotly绘制中国地图，并实现出行路线规划

**三、实验设备与环境**

微型计算机、macOS 系列操作系统 、Pycharm系列软件

**四、实验内容**

以全国主要城市为图的顶点, 铁路连接为图的边, 距离作为加权, 设计完成一个最短路径自动查找系统. 输入为出发城市和目标城市, 输出为最短路径和距离。

**五、概要设计（思路、算法、步骤等）**

## 思路：

本次实验要求寻找两地之间的最短路径，首先想到 **Floyd 算法** 来求最短路，然后再用将最短路径在地图上画出，大致思路如下：

1. 查找全国各省市坐标点，再将全国各点之间随机连线，形成通路。最后利用图的数据结构来保存全国各省会图，并将各通路、代表各省份的点再地图上表示出来。
2. 利用 Floyd 算法计算各点之间的最短路径长度以及最短路径。
3. 将最短路径在地图上表示出来，并在控制台输出最短路径的长度。
   * 准备相关数据
   * 构建若干个trace
   * 使用Figure
   * 点击响应
   * 设计鼠标点击处理

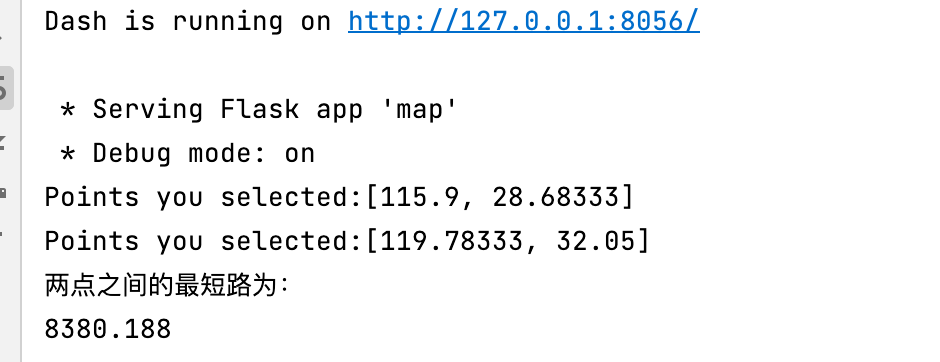
**算法：**

Floyd算法

**六、详细设计（核心代码、算法流程图等）**

*# -\*- coding: utf-8 -\*-*import dash  
from dash import dcc  
from dash import html  
import plotly.graph\_objs as go  
from dash.dependencies import Input, Output  
from random import randint  
from math import radians, cos, sin, asin, sqrt  
  
'''准备数据'''  
places = {  
 '山东': [117.000923, 36.675807],  
 '河北': [115.48333, 38.03333],  
 '吉林': [125.35000, 43.88333],  
 '黑龙江': [127.63333, 47.75000],  
 '辽宁': [123.38333, 41.80000],  
 '内蒙古': [111.670801, 41.818311],  
 '新疆': [87.68333, 43.76667],  
 '甘肃': [103.73333, 36.03333],  
 '宁夏': [106.26667, 37.46667],  
 '山西': [112.53333, 37.86667],  
 '陕西': [108.95000, 34.26667],  
 '河南': [113.65000, 34.76667],  
 '安徽': [117.283042, 31.86119],  
 '江苏': [119.78333, 32.05000],  
 '浙江': [120.20000, 30.26667],  
 '福建': [118.30000, 26.08333],  
 '广东': [113.23333, 23.16667],  
 '江西': [115.90000, 28.68333],  
 '海南': [110.35000, 20.01667],  
 '广西': [108.320004, 22.82402],  
 '贵州': [106.71667, 26.56667],  
 '湖南': [113.00000, 28.21667],  
 '湖北': [114.298572, 30.584355],  
 '四川': [104.06667, 30.66667],  
 '云南': [102.73333, 25.05000],  
 '西藏': [91.00000, 30.60000],  
 '青海': [96.75000, 36.56667],  
 '天津': [117.20000, 39.13333],  
 '上海': [121.55333, 31.20000],  
 '重庆': [106.45000, 29.56667],  
 '北京': [116.41667, 39.91667],  
 '台湾': [121.30, 25.03],  
 '香港': [114.10000, 22.20000],  
 '澳门': [113.50000, 22.20000],  
}  
logs = []  
lats = []  
names = []  
for k, v in places.items():  
 names.append(k)  
 logs.append(v[0])  
 lats.append(v[1])  
  
'''将各个地点进行编号，并放入利用字典保存'''  
rood = []  
place\_num = {} *# key = 地点 value=编号 的字典*num\_place = {} *# key = 编号 value=地点 的字典*num = 0  
for k in places.keys():  
 place\_num[k] = num  
 num\_place[num] = k  
 num = num + 1  
  
'''随机生成点之间的路线'''  
for i in range(0, num):  
 list1 = []  
 for j in range(0, num):  
 if i > j: *#创造对称矩阵* list1.append(rood[j][i])  
 else:  
 list1.append(0)  
 if i != num - 1:  
 list1[randint(i+1,num-1)]=1 *#每一个点都随机找一个点相连(若相连，矩阵置一)* rood.append(list1)  
external\_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']  
  
'''画图三部曲 '''  
'''1.构建若干个trace'''  
traces = []  
traces.append(  
 go.Scattermapbox(  
 mode="markers", *#点* text=names,  
 lon=logs,  
 lat=lats,  
 marker={'size': 10},  
 *# marker\_color='red'* )  
)  
'''增加地点之间的路径'''  
for i in range(0, num):  
 for j in range(i,num):  
 if rood[i][j] == 1: *#若两点相连，则往列表内加入一条红色连线(trace)* traces.append(  
 go.Scattermapbox(  
 mode="lines",  
 text=[num\_place[i], num\_place[j]],  
 lon=[places[num\_place[i]][0], places[num\_place[j]][0]],  
 lat=[places[num\_place[i]][1], places[num\_place[j]][1]],  
 marker={'size': 5},  
 marker\_color='red'  
 )  
 )  
'''画图三部曲 2.figure数据列表展示的样式'''  
fig = go.Figure(data=traces)  
fig.update\_layout(  
 margin={'l': 3, 't': 3, 'b': 3, 'r': 3}, *#边缘距离* mapbox={  
 'center': {'lon': places['浙江'][0], 'lat': places['浙江'][1]}, *#图中心在浙江这里* 'style': "stamen-terrain",  
 'zoom': 3 *#放大倍数* },  
)  
'''画图三部曲 3.使用dash.Dash构建app'''  
graph = dcc.Graph( *# 2* id='example-graph',  
 figure=fig *# 地图从这里传入*)  
app = dash.Dash(\_\_name\_\_, external\_stylesheets=external\_stylesheets)  
app.layout = html.Div(children=[  
 html.H1(id="title", children="中国地图"), *# 1* html.Div(id='graph',  
 children=dcc.Graph( *# 2* id='example-graph',  
 figure=fig *# 地图从这里传入* )),  
 html.Div(id="div", children="") *# 3*])  
'''根据经纬度计算两地之间的距离'''  
def geodistance(list1,list2):  
 lng1, lat1 = list1[0],list1[1]  
 lng2, lat2 = list2[0],list2[1]  
 lng1, lat1, lng2, lat2 = map(radians, [float(lng1), float(lat1), float(lng2), float(lat2)]) *# 经纬度转换成弧度* dlon = lng2-lng1  
 dlat = lat2-lat1  
 a = sin(dlat / 2) \*\* 2 + cos(lat1) \* cos(lat2) \* sin(dlon/2)\*\*2  
 distance = 2\*asin(sqrt(a)) \* 6371 \* 1000 *#地球平均半径，6371km* distance = round(distance/1000, 3)  
 return distance  
  
'''计算图中各点之间的距离'''  
distance = [] *# 表示最开始两点距离，若不可达，距离为无穷（用10000000表示）*for i in range(0, num):  
 list1 = []  
 for j in range(0, num):  
 if rood[i][j] == 1:  
 list1.append(geodistance(places[num\_place[i]],places[num\_place[j]]))  
 elif i == j:  
 list1.append(0)  
 else:  
 list1.append(10000000)  
 distance.append(list1)  
  
'''利用Floyd算法来计算最短路径'''  
path=[]  
for i in range(0, num): *# 初始化全-1 的path矩阵* list1 = []  
 for j in range(0, num):  
 list1.append(-1)  
 path.append(list1)  
for k in range(0, num):  
 for i in range(0, num):  
 for j in range(0, num):  
 if distance[i][j] > distance[i][k] + distance[k][j]:  
 distance[i][j] = distance[i][k] + distance[k][j]  
 path[i][j] = k  
'''得到 distanc矩阵表示两点最短距离和path矩阵表示最短距离的路径'''  
  
'''根据path列表来寻找最短路'''  
trace1 = [] *# 存放最短路径的列表，连线用黄色表示*def printroad(x,y,path):  
 place1 = num\_place[x]  
 place2 = num\_place[y]  
 if path[x][y] == -1:  
 trace1.append(  
 go.Scattermapbox( *# 路径的样式* mode="lines",  
 text=[place1, place2],  
 lon=[places[place1][0], places[place2][0]],  
 lat=[places[place1][1], places[place2][1]],  
 marker={'size': 5},  
 marker\_color = 'yellow'  
 )  
 )  
 return  
 else:  
 mid = path[x][y]  
 printroad(x, mid, path)  
 printroad(mid, y, path)  
  
*# 实现最短路径算法*def solve\_shortest\_path():  
 print("两点之间的最短路为：") *# 读出两点之间的最短路* print(distance[place\_num[points\_name[0]]][place\_num[points\_name[1]]])  
 global trace1  
 trace1 = []  
 printroad(place\_num[points\_name[0]], place\_num[points\_name[1]], path) *# 获得两点之间最短路径* return None  
  
  
two\_points = [] *# 需要选择两个点*points\_name = []  
SELECTED = ""  
  
def selectPoint(point, double\_points, text):  
 if len(double\_points) == 1 and double\_points[0] == point:  
 return  
 else:  
 double\_points.append(point)  
 points\_name.append(text)  
 print("Points you selected:{}".format(point))  
  
'''dash地图创建好后，点击响应'''  
@app.callback(  
 [Output('div', 'children'), Output('graph', 'children')],  
 [Input('example-graph', 'clickData')])  
  
def display\_click\_data(clickData):  
 global two\_points, SELECTED,points\_name,graph  
 if clickData:  
 point\_dict = clickData['points'][0]  
 lon = point\_dict['lon']  
 lat = point\_dict['lat']  
 text = point\_dict['text']  
 selectPoint([lon, lat], two\_points, text)  
 SELECTED += "{}:({},{}) ".format(text, lon, lat)  
 MSG = "您选择了" + SELECTED  
  
 if len(two\_points) == 2:  
 solve\_shortest\_path() *# 最短路算法实现函数* two\_points = []  
 points\_name = []  
 SELECTED = ""  
  
 global trace1  
 fig = go.Figure(data=traces + trace1)  
 fig.update\_layout(  
 margin={'l': 3, 't': 3, 'b': 3, 'r': 3},  
 mapbox={  
 'center': {'lon': places['浙江'][0], 'lat': places['浙江'][1]},  
 'style': "stamen-terrain",  
 'zoom': 3  
 },  
 )  
 graph = dcc.Graph( *# 2* id='example-graph',  
 figure=fig *# 地图从这里传入* )  
 return MSG, graph  
 else:  
 return "请选择两个点", graph  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run\_server(debug=True, port=8056)

**七、测试结果及分析：**



**八、总结**

通过本次实验，我使用了Plotly的Dash库绘制了中国地图，并在其框架上运用Floyd算法计算了地图上随机生成路径的最短路径。这次实验使我对Dash库有了一定的了解，并且通过实际操作掌握了最短路径算法。

在这次实验中，我使用Python语言完成了最短路径算法的实现。之前在学习Floyd算法时只是在理论上有了解，并没有亲自用计算机实现过。通过这次实验，我深入地理解了Floyd算法。

然而，对我来说，本次实验最大的挑战并不在于Floyd算法，而是在最后的最短路径可视化呈现上。由于对Plotly Dash工具并不熟悉，在更新地图和可视化路径时，我遇到了很大的困难。我对如何使用Plotly Dash进行地图更新问题进行了多次尝试，最终才解决了这个问题。

总的来说，本次实验给我带来了丰富的收获。我不仅更加深入地理解了Floyd算法，还初步了解了Plotly Dash工具。我希望今后能够进一步利用这些知识和工具，进行更多有意义的实践。这次实验为我打开了新的可能性，让我在算法和可视化方面都有了进一步的提升。