Python程序设计-大作业

班级：2019211308

学号：2019211539

姓名：顾天阳

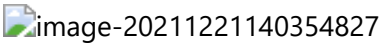
作业题目

数据

gpw-v4-population-count-rev11_2020_30_sec_asc.zip是一个全球人口分布数据压缩文件，解压后包括了8个主要的asc后缀文件，他们是全球网格化的人口分布数据文件，这些文件分别是：

- gpw-v4-population-count-rev11_2020_30_sec_1.asc
- gpw-v4-population-count-rev11_2020_30_sec_2.asc
- gpw-v4-population-count-rev11_2020_30_sec_3.asc
- gpw-v4-population-count-rev11_2020_30_sec_4.asc
- gpw-v4-population-count-rev11_2020_30_sec_5.asc
- gpw-v4-population-count-rev11_2020_30_sec_6.asc
- gpw-v4-population-count-rev11_2020_30_sec_7.asc
- gpw-v4-population-count-rev11_2020_30_sec_8.asc

这些文件分布对应地球不同经纬度的范围。



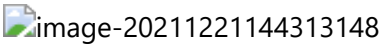
压缩文件下载网页：https://sedac.ciesin.columbia.edu/data/set/gpw-v4-population-count-rev11/data-download

服务端

压缩文件（gpw-v4-population-count-rev11_2020_30_sec_asc.zip）是一个全球人口分布数据。基于Sanic实现一个查询服务，服务包括：

- 按给定的经纬度范围查询人口总数，查询结果采用JSON格式。
- 不可以采用数据库，只允许使用文件方式存储数据。
- 可以对现有数据进行整理以便加快查询速度，尽量提高查询速度。

查询参数格式 采用GeoJSON（https://geojson.org/）的多边形（每次只需要查询一个多边形范围，只需要支持凸多边形）



客户端

针对上面的查询服务，实现一个服务查询客户端，数据获取后使用Matplotlib散点图（Scatter）进行绘制。

- 横坐标（x轴）为经度。

- 纵坐标（y轴）为维度。

服务端代码

程序源代码嵌入下方的code block中。

```
from sanic import Sanic
from sanic.response import json
from multiprocessing import Pool
import numpy as np
from shapely import geometry

app = Sanic('Population')

# 全局变量，用于存储墨卡托投影平面等分8块的人口数据
data_1, data_2, data_3, data_4 = {}, {}, {}, {}
data_5, data_6, data_7, data_8 = {}, {}, {}, {}
# 全局变量，记录人口总数
population_total = 0


@app.route('/population')
async def population(request):
    # 获取客户端传来的多边形json数据中的经纬度点对集合
    polygon = request.json['coordinates']
    # 计算人口总数，并得到返回的点集，用于客户端绘制散点图
    points = calc_population(polygon)
    # 构造返回给客户端的json数据，包括人口总数和人口点集（经度，纬度，人口数值）
    population_data = {
        'population_total': population_total,
        'population_points': points
    }
    # 以json格式回传给客户端
    return json(population_data)


def calc_population(lonLats):
    # 使用Polygon类构造多边形
    polygon = geometry.Polygon(lonLats)
    # 获取多边形边界
    lonMin, latMin, lonMax, latMax = polygon.bounds
    # 步长为30角秒，转化成角度
    step = 30 / 3600
    # 获取每一个grid的面积area
    cellArea = geometry.Polygon(
        [(0, 0), (0, step), (step, step), (step, 0)]).area
    global population_total
    population_total = 0
    points = []
    # 根据bound遍历所包含的所有grid列表
    for lon in np.arange(lonMin, lonMax, step):
        for lat in np.arange(latMin, latMax, step):
```

```
            cellLon1 = lon - lon % step
            cellLon2 = lon - lon % step + step
            cellLat1 = lat - lat % step
            cellLat2 = lat - lat % step + step
            # 构造当前grid的多边形模型
            cellPolygon = geometry.Polygon(
                [(cellLon1, cellLat1), (cellLon1, cellLat2), (cellLon2, cellLat2),
(cellLon2, cellLat1)])
            # 计算当前grid和多边形的相交面积
            area = cellPolygon.convex_hull.intersection(
                polygon.convex_hull).area
            # 如果面积不为零，则进一步获取该grid的人口数值，并根据相交面积近似获取多边形
在该grid中的人口数值
            if area > 0.0:
                p = get_population_from_file(cellLon1, cellLat1)
                # 添加到绘制点对集合
                points.append([cellLon1, cellLat1, p])
                population_total += (area / cellArea) * p
    return points


def get_population_from_file(lon, lat):
    step = 30 / 3600
    # 计算经纬度映射的序号对
    x = int((lon - (-180)) / step)
    y = int((lat - (-90)) / step)
    # 根据序号对判断该grid位于墨卡托投影平面等分8块的哪一块中
    # 如果序号对存在于人口数据的键集合中，则根据键找出人口数值，否则返回0
    if x >= 0 and x < 10800 and y >= 10800 and y < 21600:
        if (21600 - y, x) in data_1.keys():
            return data_1[(21600 - y, x)]
        else:
            return 0
    elif x >= 10800 and x < 21600 and y >= 10800 and y < 21600:
        if (21600 - y, x - 10800) in data_2.keys():
            return data_2[(21600 - y, x - 10800)]
        else:
            return 0
    elif x >= 21600 and x < 32400 and y >= 10800 and y < 21600:
        if (21600 - y, x - 21600) in data_3.keys():
            return data_3[(21600 - y, x - 21600)]
        else:
            return 0
    elif x >= 32400 and x < 43200 and y >= 10800 and y < 21600:
        if (21600 - y, x - 32400) in data_4.keys():
            return data_4[(21600 - y, x - 32400)]
        else:
            return 0
    elif x >= 0 and x < 10800 and y >= 0 and y < 10800:
        if (10800 - y, x) in data_5.keys():
            return data_5[(10800 - y, x)]
        else:
            return 0
    elif x >= 10800 and x < 21600 and y >= 0 and y < 10800:
```

```
        if (10800 - y, x - 10800) in data_6.keys():
            return data_6[(10800 - y, x - 10800)]
        else:
            return 0
    elif x >= 21600 and x < 32400 and y >= 0 and y < 10800:
        if (10800 - y, x - 21600) in data_7.keys():
            return data_7[(10800 - y, x - 21600)]
        else:
            return 0
    elif x >= 32400 and x < 43200 and y >= 0 and y < 10800:
        if (10800 - y, x - 32400) in data_8.keys():
            return data_8[(10800 - y, x - 32400)]
        else:
            return 0


def get_population_data(file):
    data = {}
    f = open(file)
    # 遍历人口数据文件中的每一行
    for line in f.readlines():
        # 对一行的数据进行拆分
        line = line.split()
        data[(int(line[0]), int(line[1]))] = float(line[2])
    return data


def concurrent_process():
    global data_1, data_2, data_3, data_4, data_5, data_6, data_7, data_8
    files = ['1.csv', '2.csv', '3.csv', '4.csv',
             '5.csv', '6.csv', '7.csv', '8.csv']
    # 引入进程池，池里最大进程数目为10
    pool = Pool(10)
    data_1, data_2, data_3, data_4, data_5, data_6, data_7, data_8 = pool.map(
        get_population_data, files)
    # 不允许更多的进程加入
    pool.close()
    # 阻塞，等待所有子进程完成
    pool.join()


if __name__ == '__main__':
    # 并行读取人口数据，加快处理速度
    concurrent_process()
    # 运行本地主机，端口号8000
    app.run(host='127.0.0.1', port=8000)
```

代码说明

需求实现

原始文本数据预处理

- 首先通过numpy中的loadtxt方法读取原始asc文件，并设置skiprows为6，表示跳过文件起始6行；将人口数据存入data对象中。
- 将墨卡托投影平面等分成8块，对每一块中的每一个经纬度点对进行编号，点对之间的点差为30角秒，由此可以将每一块投影平面的经度和纬度分别进行10800等分，即平面的左上角坐标为（0，0），右下角坐标为（10799，10799）。
- 遍历data文件，将数值为-9999的数据抛弃，其余数据以（坐标x，坐标y，人口值）存入新的csv文件中。
- 预处理代码如下：

```
import numpy as np

def save_2_csv(original_file, target_file):
    data = np.loadtxt(original_file, skiprows=6)
    f = open(target_file, 'a')
    for i in range(10800):
        for j in range(10800):
            if data[i][j] > 0:
                f.write('%i %i %.8f\n' % (i, j, data[i][j]))

if __name__ == '__main__':
    for i in range(1, 9):
        original_file = str(i) + '.asc'
        target_file = str(i) + '.csv'
        save_2_csv(original_file, target_file)
```

得到多边形坐标的边界bound

- 引入python地理处理包shapely，它针对笛卡尔平面对几何对象进行操作和分析；这里我们使用shapely中的geometry对象。
- 对处理过的经纬度点对数组，通过geometry上的Polygon类，获得多边形坐标的边界bound，即最大最小经度、最大最小纬度。
- 相关代码如下：

```
from shapely import geometry

polygon = geometry.Polygon(lonLats)
lonMin, latMin, lonMax, latMax = polygon.bounds
```

根据bound遍历出包含的所有grid列表

- 通过两层for循环，第一层从最小经度以30角秒步长遍历至最大经度，第二层从最小纬度以30角秒步长遍历至最大纬度。

- 针对遍历中的每一个经纬度点对，寻找其所在的以30角秒划分的grid区域，并通过取余等操作找出该grid的标识坐标；进而传入该grid的坐标为其构造Polygon类。
- 相关代码如下：

```
for lon in np.arange(lonMin, lonMax, step):
    for lat in np.arange(latMin, latMax, step):
        cellLon1 = lon - lon % step
        cellLon2 = lon - lon % step + step
        cellLat1 = lat - lat % step
        cellLat2 = lat - lat % step + step
        cellPolygon = geometry.Polygon([(cellLon1, cellLat1), (cellLon1,
cellLat2), (cellLon2, cellLat2), (cellLon2, cellLat1)])
```

根据grid列表获得多边形所含人口总数

- 获取人口数值主要通过get_population_from_file(lon, lat)函数，该函数首先根据lon（经度）和lat（纬度）两个参数，判断grid属于墨卡托投影平面等分8块中的哪一块，据此将经纬度点对映射成序号对，用于在人口大数据中检索。
- 获取每一个grid的人口数值后，根据该grid与多边形的相交面积近似得出多边形在该grid内的人口，对所有此项求即得多边形所含人口总数。
- 相关代码如下：

```
def get_population_from_file(lon, lat):
    step = 30 / 3600
    x = int((lon - (-180)) / step)
    y = int((lat - (-90)) / step)
    if x >= 0 and x < 10800 and y >= 10800 and y < 21600:
        if (21600 - y, x) in data_1.keys():
            return data_1[(21600 - y, x)]
        else:
            return 0
    elif x >= 10800 and x < 21600 and y >= 10800 and y < 21600:
        if (21600 - y, x - 10800) in data_2.keys():
            return data_2[(21600 - y, x - 10800)]
        else:
            return 0
    elif x >= 21600 and x < 32400 and y >= 10800 and y < 21600:
        if (21600 - y, x - 21600) in data_3.keys():
            return data_3[(21600 - y, x - 21600)]
        else:
            return 0
    elif x >= 32400 and x < 43200 and y >= 10800 and y < 21600:
        if (21600 - y, x - 32400) in data_4.keys():
            return data_4[(21600 - y, x - 32400)]
        else:
            return 0
    elif x >= 0 and x < 10800 and y >= 0 and y < 10800:
        if (10800 - y, x) in data_5.keys():
            return data_5[(10800 - y, x)]
```

```
        else:
            return 0
    elif x >= 10800 and x < 21600 and y >= 0 and y < 10800:
        if (10800 - y, x - 10800) in data_6.keys():
            return data_6[(10800 - y, x - 10800)]
        else:
            return 0
    elif x >= 21600 and x < 32400 and y >= 0 and y < 10800:
        if (10800 - y, x - 21600) in data_7.keys():
            return data_7[(10800 - y, x - 21600)]
        else:
            return 0
    elif x >= 32400 and x < 43200 and y >= 0 and y < 10800:
        if (10800 - y, x - 32400) in data_8.keys():
            return data_8[(10800 - y, x - 32400)]
        else:
            return 0
```

处理grid与多边形相交的部分

- 通过调用Polygon类中的intersection，找出当前grid与多边形的相交面积area；如果面积不为零，进一步找出该grid的人口数值，通过相交面积近似得出包含于多边形内的人口。
- 相关代码如下：

```
area = cellPolygon.convex_hull.intersection(
    polygon.convex_hull).area
if area > 0.0:
    p = get_population_from_file(cellLon1, cellLat1)
    population_total += (area / cellArea) * p
```

客户端代码

客户端代码嵌入下发的code block中。

```
import aiohttp
import asyncio
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.basemap import Basemap

# 存储不同人口密集度的点集，包括<1, 1-5, 5-25, 25-250, 250-1000, >1000(persons)
population_points_1 = []
population_points_2 = []
population_points_3 = []
population_points_4 = []
population_points_5 = []
population_points_6 = []
```

```
async def main(url, data):
    # 设置超时时间
    timeout = aiohttp.ClientTimeout(total=1000)
    # 客户端会话
    async with aiohttp.ClientSession() as session:
        async with session.get(url=url, json=data, timeout=timeout) as response:
            # 阻塞等待服务端消息
            population_data = await response.json()
            # 输出人口总数
            print(population_data['population_total'])
            # print(f'content-type={response.headers["content-type"]}')
            # 将人口点集按照密集度分类
            get_population_classification(population_data['population_points'])
            # 绘制人口散点图
            draw_population_scatter()


def draw_population_scatter():
    # 设置图片大小
    plt.figure(figsize=(10, 8))
    # 绘制基础地图，可以设置经纬度的范围
    m = Basemap(llcrnrlat=-90, urcrnrlat=90, llcrnrlon=-180, urcrnrlon=180)
    # 绘制海岸线
    m.drawcoastlines()
    # 绘制边界
    m.drawmapboundary(fill_color='white')
    # 填充大陆
    m.fillcontinents(color='none', lake_color='white')

    # 绘制经纬度点的分割
    parallels = np.arange(-90., 90., 10.)
    m.drawparallels(parallels, labels=[False, True, True, False])
    meridians = np.arange(-180., 180., 20.)
    m.drawmeridians(meridians, labels=[True, False, False, True])

    # 绘制多边形边界
    polygon = np.array([(-52.3, 45.6), (-40.5, 47.8), (-30.6, 35.1), (-52.3,
45.6)])
    m.plot(polygon[:, 0], polygon[:, 1], c='k', lw=1)

    # 使用np.array初始化人口点集
    points_1 = np.array(population_points_1)
    points_2 = np.array(population_points_2)
    points_3 = np.array(population_points_3)
    points_4 = np.array(population_points_4)
    points_5 = np.array(population_points_5)
    points_6 = np.array(population_points_6)

    # 绘制散点图
    if len(points_1) > 0:
        p1 = m.scatter(
            points_1[:, 0], points_1[:, 1], marker='o', s=0.01, c='#fefde1',
label='<1')
    if len(points_2) > 0:
```

```
        p2 = m.scatter(
            points_2[:, 0], points_2[:, 1], marker='o', s=0.01, c='#adcab4',
label='1-5')
    if len(points_3) > 0:
        p3 = m.scatter(
            points_3[:, 0], points_3[:, 1], marker='o', s=0.01, c='#509894',
label='5-25')
    if len(points_4) > 0:
        p4 = m.scatter(
            points_4[:, 0], points_4[:, 1], marker='o', s=0.01, c='#227295',
label='25-250')
    if len(points_5) > 0:
        p5 = m.scatter(
            points_5[:, 0], points_5[:, 1], marker='o', s=0.01, c='#2b458f',
label='250-1000')
    if len(points_6) > 0:
        p6 = m.scatter(
            points_6[:, 0], points_6[:, 1], marker='o', s=0.01, c='#030064',
label='>1,000(persons)')

    # 添加图例
    plt.legend(loc='lower left', markerscale=100)
    plt.show()


def get_population_classification(population_points):
    # 通过人口数值给当前点分类，包括<1, 1-5, 5-25, 25-250, 250-1000, >1000(persons)
    for p in population_points:
        if p[2] >= 0 and p[2] < 1:
            population_points_1.append(p)
        elif p[2] >= 1 and p[2] < 5:
            population_points_2.append(p)
        elif p[2] >= 5 and p[2] < 25:
            population_points_3.append(p)
        elif p[2] >= 25 and p[2] < 250:
            population_points_4.append(p)
        elif p[2] >= 250 and p[2] < 1000:
            population_points_5.append(p)
        else:
            population_points_6.append(p)


if __name__ == '__main__':
    url = 'http://127.0.0.1:8000/population'
    # 构造GeoJSON的多边形数据
    data = {
        'type': 'Polygon',
        'coordinates': [
            # 这里是全美数据，可自行修改
            (-127, 50), (-73, 50), (-73, 25), (-127, 25), (-127, 50)
        ]
    }
    asyncio.run(main(url, data))
```

代码说明

程序说明

- 数据采用GeoJson格式，每次查询一个多边形范围，例如：

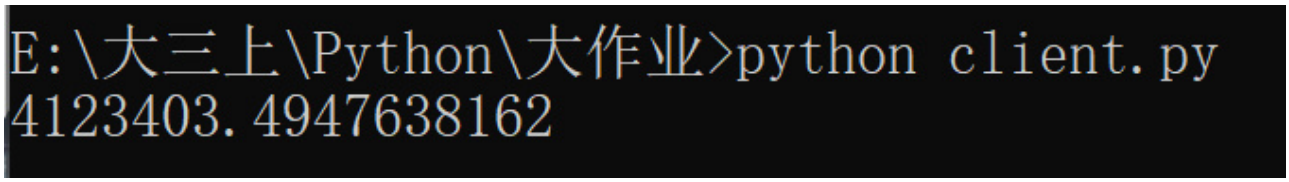
```
data = {
    'type': 'Polygon',
    'coordinates': [
        # 这里是全美数据，可自行修改
        (-127, 50), (-73, 50), (-73, 25), (-127, 25), (-127, 50)
    ]
}
```

- 主程序基于aiohttp进行编写，调用了aiohttp.ClientSession()和session.get()等函数。
- 待服务端返回人口总数及结果点集时，程序首先会输出人口总数，并对人口点集进行分类，包括<1, 1-5, 5-25, 25-250, 250-1000, >1000(persons)
- 绘制散点图部分基于matplotlib.pyplot和Basemap，其中，basemap基于GEOS的地图二维数据，其底图数据库与GMT相同，封装了大量常用的地图投影、坐标转换功能，利用简洁的Python语法支持绘出多种多样的地理地图。

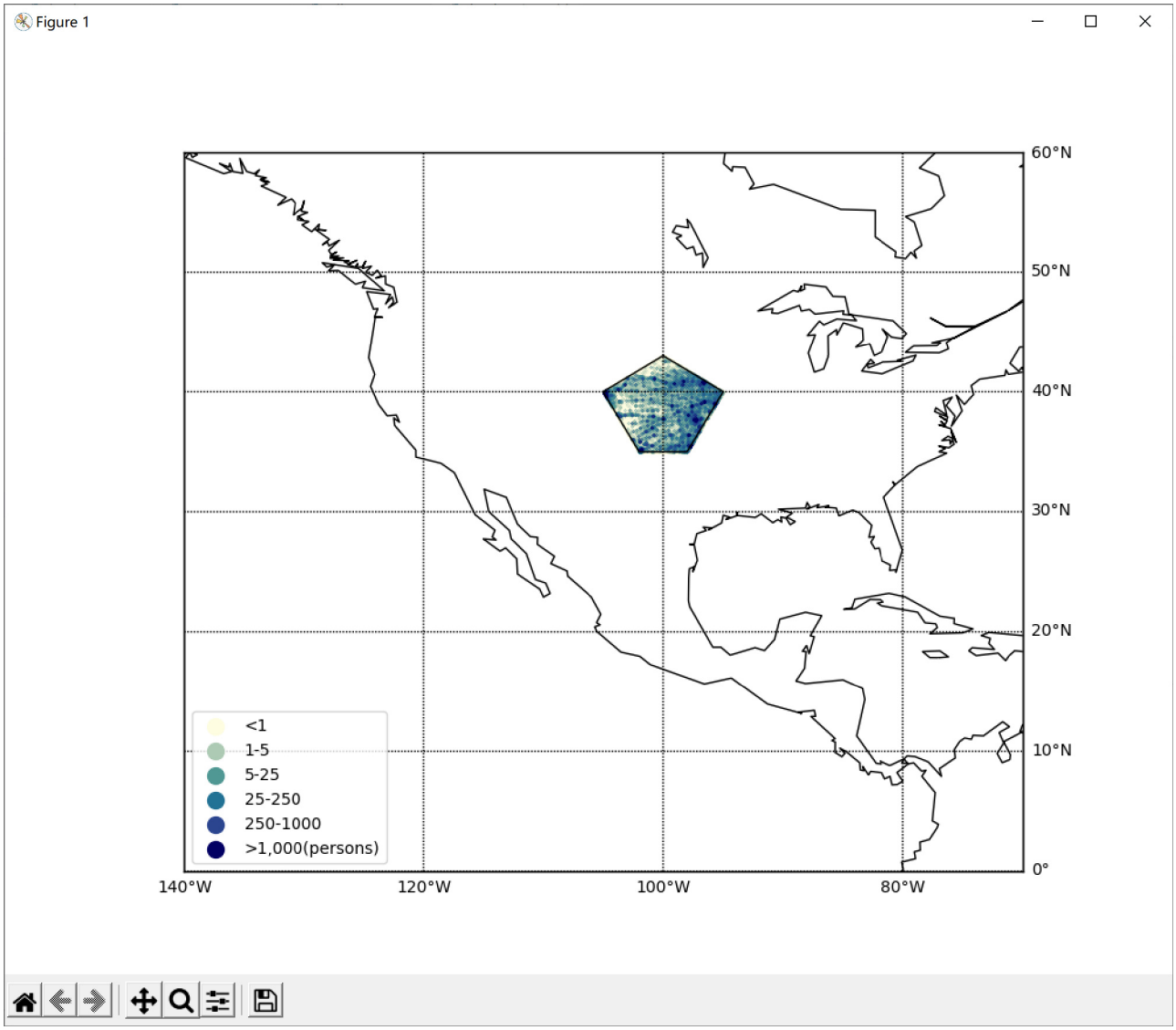
效果图

样例一

- 经纬度点：(-100, 43), (-95, 40), (-98, 35), (-102, 35), (-105, 40), (-100, 43)
- 人口总数及散点图：

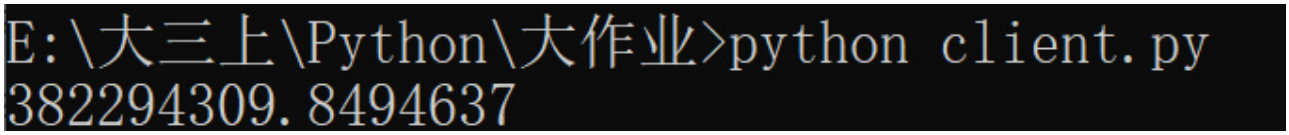


- 可以看到，此多边形范围内约有400万人。



样例二

- 经纬度点（近乎全美国）：(-127, 50), (-73, 50), (-73, 25), (-127, 25), (-127, 50)
- 人口总数及散点图：



- 可以看到，此范围约有3.8亿人，去除加拿大和墨西哥区域内人数，近似美国总人口，较好吻合！