Python程序设计-大作业

班级：2021211306

学号：2021211108

姓名：沈原灏

1 作业题目

1.1 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

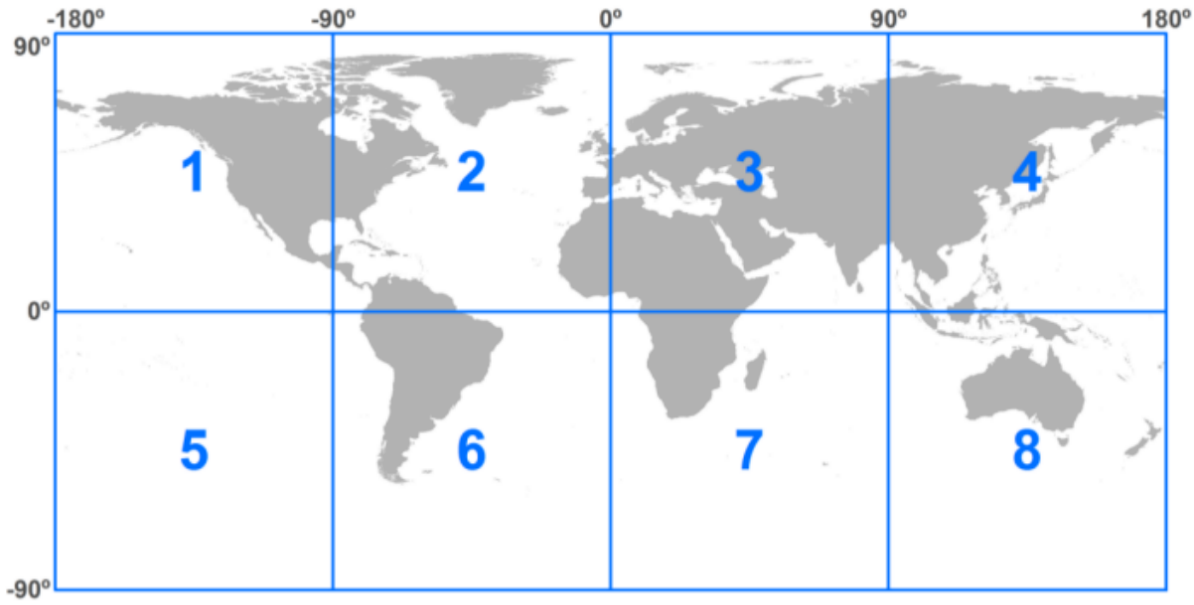


Figure 1. Tiling of 30 arc-second ASCII rasters.

这些文件分布对应地球不同经纬度的范围。

1.2 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 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）

```
{
    "type": "Polygon",
    "coordinates": [
        [[30.0, 10.0], [40.0, 40.0], [20.0, 40.0], [10.0, 20.0], [30.0, 10.0]]
    ]
}
```

1.3 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）为纬度。

2 服务端代码

预处理模块 `pre_processor.py` 如下：

```python
#  pre_processor.py
#  Copyright (c) 2023 Yuanhao Shen
import asyncio
import os
import numpy as np


async def preprocess():
    """
    预处理数据，将原始数据处理成10度*10度的block
    """
    for i in range(1, 9):
        step = 10  # 处理成10度*10度的block
        with open(f"./gpw-v4-population-count-rev11_2020_30_sec_asc/gpw_v4_population_count_rev11_2020_30_sec_{i}.asc",
                  "r") as f:
            f.readline()
            f.readline()
            stx = int(float(f.readline().split()[1]))  # 左上角经度
            sty = int(float(f.readline().split()[1])) + 90  # 左上角纬度
            is_processed = True  # 该部分中的所有block是否已经被处理
            for x_offset in range(0, 90, step):
                for y_offset in range(0, 90, step):
                    if not os.path.exists(f"./data/data_{stx + x_offset}_{sty - y_offset}.npy"):
                        is_processed = False
        print(f"Processing data on x: {stx} y: {sty}")
        if is_processed:
            continue
        data = np.genfromtxt(
```

```
            f"./gpw-v4-population-count-rev11_2020_30_sec_asc/gpw_v4_population_count_rev11_2020_30_sec_{i}.asc",
            skip_header=6)
        data[data == -9999] = np.nan  # 空数据
        for x_offset in range(0, 90, step):
            for y_offset in range(0, 90, step):
                if not os.path.exists(f"./data/data_{stx + x_offset}_{sty - y_offset}.npy"):
                    print(f"Creating data_{stx + x_offset}_{sty - y_offset}.npy")
                    np.save(f"./data/data_{stx + x_offset}_{sty - y_offset}.npy",
                            data[y_offset * 120:(y_offset + step) * 120, x_offset * 120:(x_offset + step) * 120])


if __name__ == '__main__':
    asyncio.run(preprocess())

```

服务端 `server.py` 代码如下：

```
#  server.py
#  Copyright (c) 2023 Yuanhao Shen

import asyncio
import math
import numpy as np
from sanic import Sanic
from sanic.response import json
from sanic.log import logger
from shapely.geometry import Polygon
from shapely.errors import TopologicalError
from pre_processor import preprocess

app = Sanic("population")

@app.listener('before_server_start')
async def setup(app, loop):
    await preprocess()


async def get_block_data(block_lon, block_lat, polygon, step):
    logger.info(f"Query on block lon: {block_lon} lat: {block_lat}")
    result = []
    total_population = 0
    min_lon, min_lat, max_lon, max_lat = polygon.bounds
    block_data = np.load(f"./data/data_{block_lon}_{block_lat}.npy")
    for second_lon in range(max(block_lon * 3600, math.floor(min_lon / 30) * 30),
                            min((block_lon + step) * 3600, math.ceil(max_lon / 30) * 30), 30):
        for second_lat in range(min(block_lat * 3600, math.ceil(max_lat / 30) * 30),

```

```
                                max((block_lat - step) * 3600,
math.floor(min_lat / 30) * 30), -30):
            lon_offset = int((block_lat * 3600 - second_lat) / 30)
            lat_offset = int((second_lon - block_lon * 3600) / 30)
            cell_polygon = Polygon(
                ((second_lon, second_lat), (second_lon + 30, second_lat),
(second_lon + 30, second_lat - 30),
                 (second_lon, second_lat - 30))).intersection(polygon)
            if cell_polygon.area > 0:
                result.append((second_lon, second_lat, cell_polygon.area /
900 * block_data[lon_offset, lat_offset]))
                if not np.isnan(result[-1][2]):
                    total_population += result[-1][2]
    return result, total_population


@app.post("/data")
async def get_data(request):
    """
    data接口，接受一个多边形的坐标列表，返回该多边形内的人口数据
    """
    try:
        coordinates = request.json.get("coordinates")
        polygon = Polygon(coordinates)  # 构造多边形
        logger.info(f"Query coordinates: {coordinates}")
        min_lon, min_lat, max_lon, max_lat = polygon.bounds
        step = 10
        result = []
        total_population = 0
        task_list = []
        for block_lon in range(math.floor(min_lon / 3600 / step) * step,
math.ceil(max_lon / 3600 / step) * step, step):
            for block_lat in range(math.ceil(max_lat / 3600 / step) * step,
math.floor(min_lat / 3600 / step) * step,
                                   -step):
                task_list.append(
                    asyncio.create_task(get_block_data(block_lon, block_lat,
polygon, step)))  # 并行处理
        for task in task_list:
            result_, total_population_ = await task
            result += result_
            total_population += total_population_
        return json({"total_population": total_population, "result":
result})
    except KeyError:  # 参数错误
        return json([], status=400)
    except (ValueError, TopologicalError):  # 非多边形
        return json([], status=406)


if __name__ == '__main__':
    app.run(port=8848)
```

2.1 代码说明

2.1.1 数据预处理

我编写了 `pre_processor.py` 实现了数据预处理模块，流程如下：

1. **获取数据文件**: 遍历指定的原始人口计数数据文件（`gpw_v4_population_count_rev11_2020_30_sec_i.asc`，其中 `i` 从1到8），并对每个文件进行处理。
2. **读取并解析数据**: 每个数据文件的前两行被跳过，第三、四行的内容用于确定该数据文件对应区域的左上角经度和纬度。
3. **检查已处理数据**: 程序检查每个10度×10度区块是否已经存在对应的 `.npy` 文件，以判断该区块是否已处理。
4. **数据处理**:
    - 如果区块未处理，从原始数据文件中读取数据，使用 `np.genfromtxt` 并跳过前6行（文件头）。
    - 对于标记为 `-9999` 的数据（表示空数据），将其替换为 `np.nan`。
    - 按10度×10度的区块分割数据，并将每个区块保存为 `.npy` 文件。
5. **预处理优化**：在服务端每次启动时，都会调用数据预处理模块，由于数据量过于巨大，处理全部数据需要半小时左右的时间，而这是 `Sanic` 中 `before_server_start` 部分所无法支持的等待时间，所以我采用如下优化：
    - 在启动服务端之前单独运行一次数据预处理模块，保证服务端的启动时间不会太长。
    - 在预处理模块加入了判断，如果发现当前文件已经被预处理成了 `.npy` 文件，则直接跳过。

2.1.2 查询处理

1. **接收多边形数据**: 通过 `/data` 端点接收客户端发送的包含多边形坐标的请求。
2. **构建多边形对象**: 使用 `shapely` 库的 `Polygon` 类根据坐标构建多边形对象。
3. **计算多边形边界**: 使用 `polygon.bounds` 获取多边形的边界（最小经度、最小纬度、最大经度、最大纬度）。
4. **遍历相关区块**: 根据边界遍历所有可能与多边形相交的10度×10度的区块。
5. **并行处理区块**: 对每个区块异步执行 `get_block_data` 函数以并行处理。
6. **处理区块内的数据**:
    - 加载区块内的人口数据。
    - 遍历区块内每30秒（1/120度）的单元格，判断其是否与多边形相交。
    - 计算相交单元格的人口数据，将重合的面积比整个cell面积的权重乘这个cell中的人口数（即实际在多边形中的人口数） 加入总人口中。

2.1.3 日志

代码中我使用了 `Sanic` 的日志系统的 `logger.info` 添加了一些日志信息，效果如下：

```
[2023-12-17 17:20:52 +0800] [5392] [INFO] Sanic v23.6.0
[2023-12-17 17:20:52 +0800] [5392] [INFO] Goin' Fast @ http://127.0.0.1:8848
[2023-12-17 17:20:52 +0800] [5392] [INFO] mode: production, single worker
[2023-12-17 17:20:52 +0800] [5392] [INFO] server: sanic, HTTP/1.1
[2023-12-17 17:20:52 +0800] [5392] [INFO] python: 3.11.2
[2023-12-17 17:20:52 +0800] [5392] [INFO] platform: Windows-10-10.0.19045-SP0
[2023-12-17 17:20:52 +0800] [5392] [INFO] packages: sanic-routing==23.6.0
[2023-12-17 17:20:53 +0800] [6856] [INFO] Starting worker [6856]
[2023-12-17 17:21:01 +0800] [6856] [INFO] Query prarms: [[416307.69230769225, 118738.63636363635], [423692.3076923075, 118738.63636363635], [425538.4615384615, 108613.63636363635], [418153.846153846, 106772.7272727273]]
[2023-12-17 17:21:01 +0800] [6856] [INFO] Query on block x: 110 y: 40
[2023-12-17 17:21:08 +0800] [6856] [INFO] Query on block x: 110 y: 30
```

3 客户端代码

```python
import tkinter as tk
from tkinter import messagebox
import requests
import numpy as np
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature
from PIL import Image, ImageTk
from os import path

path = path.dirname(__file__)


class Coordinate:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return f"[{self.x},{self.y}]"

    @property
    def x_deg(self):
        return self.x // 3600

    @property
    def x_min(self):
        return self.x % 3600 // 60

    @property
    def x_sec(self):
        return f'{self.x % 60:.2f}'

    @property
```

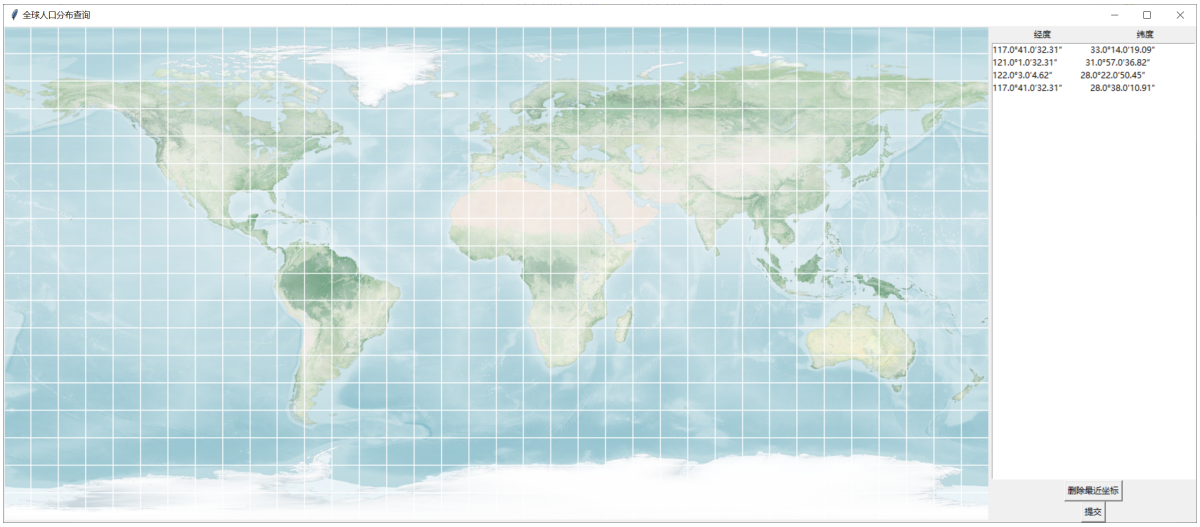
```
    def y_deg(self):
        return self.y // 3600

    @property
    def y_min(self):
        return self.y % 3600 // 60

    @property
    def y_sec(self):
        return f'{self.y % 60:.2f}'


class ClientApp:
    def __init__(self, root):
        self.root = root
        self.root.title("全球人口分布查询")
        self.coordinate_list = []
        self.setup_ui()

    def setup_ui(self):
        # 设置图像显示区域
        self.canvas = tk.Canvas(self.root, width=1400, height=700)
        self.canvas.pack(side="left")

        self.load_world_map()

        # 设置列表显示区域
        self.list_frame = tk.Frame(self.root)
        self.list_frame.pack(side="right", fill="both", expand=True)

        header_frame = tk.Frame(self.list_frame)
        header_frame.pack(side="top", fill="x")

        # 添加列名
        label_longitude = tk.Label(header_frame, text="经度", width=20)
        label_longitude.pack(side="left")

        label_latitude = tk.Label(header_frame, text="纬度", width=20)
        label_latitude.pack(side="left")

        self.listbox = tk.Listbox(self.list_frame)
        self.listbox.pack(side="top", fill="both", expand=True)

        self.submit_button = tk.Button(self.list_frame, text="提交",
command=self.submit)
        self.submit_button.pack(side="bottom")

        # 删除按钮
        self.remove_button = tk.Button(self.list_frame, text="删除最近坐标",
command=self.remove_last_point)
        self.remove_button.pack(side="bottom")

        self.canvas.bind("<Button-1>", self.on_canvas_click)

    def load_world_map(self):
        try:
```

```
            # 加载图片并调整大小
            self.map_image = Image.open('world_map_gridded.png')
            self.map_image = self.map_image.resize((1400, 700),
Image.Resampling.LANCZOS)  # 修改这里
            self.map_photo = ImageTk.PhotoImage(self.map_image)

            # 绘制图片
            self.canvas.create_image(0, 0, anchor="nw",
image=self.map_photo)
        except IOError:
            print("无法加载地图图片")

    def on_canvas_click(self, event):
        x = event.x / self.canvas.winfo_width() * 1296000 - 648000
        y = -(event.y / self.canvas.winfo_height() * 648000 - 324000)
        self.add_point(x, y)

    def add_point(self, x, y):
        self.coordinate_list.append(Coordinate(x, y))
        self.update_listbox()

    def remove_last_point(self):
        if self.coordinate_list:
            self.coordinate_list.pop()
            self.update_listbox()

    def update_listbox(self):
        self.listbox.delete(0, tk.END)
        space = ' ' * 10
        for coord in self.coordinate_list:
            self.listbox.insert(tk.END,
                                f"{coord.x_deg}°{coord.x_min}′{coord.x_sec}
"" + space + f"{coord.y_deg}°{coord.y_min}′{coord.y_sec}"")

    def submit(self):
        if not self.coordinate_list:
            messagebox.showinfo("提示", "列表为空，请先添加坐标点！")
            return

        # 准备请求数据
        coordinates = [[coord.x, coord.y] for coord in
self.coordinate_list]
        payload = {
            "type": "Polygon",
            "coordinates": coordinates
        }

        try:
            r = requests.post("http://127.0.0.1:8848/data", json=payload)
            if r.status_code != 200:
                messagebox.showerror("错误", f"服务器返回错误:
{r.status_code}")
                return

            # 解析返回的数据
            response_data = r.json()
```

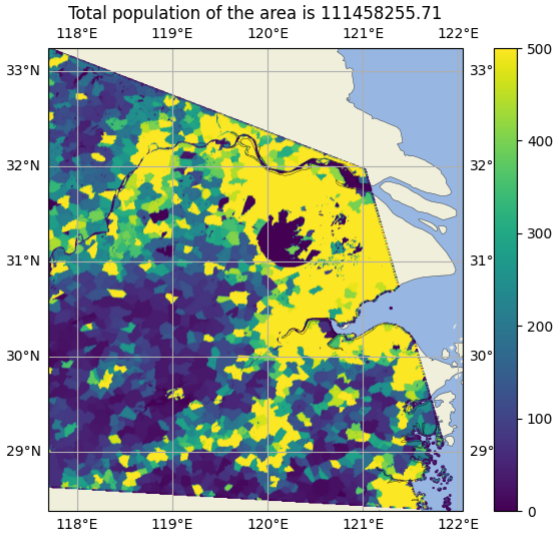
```
            data = np.array(response_data.get("res")).transpose((1, 0))
            total_population = response_data.get("total")

            # 绘制结果
            self.plot_result(data, total_population)
        except requests.RequestException as e:
            messagebox.showerror("错误", f"请求失败: {e}")

    def plot_result(self, data, total_population):
        extent = [np.min(data[0]) / 3600, np.max(data[0]) / 3600,
                  np.min(data[1]) / 3600, np.max(data[1]) / 3600]
        fig = plt.figure(figsize=(8, 6))
        ax = fig.add_subplot(111, projection=ccrs.PlateCarree())
        ax.set_extent(extent, crs=ccrs.PlateCarree())
        ax.add_feature(cfeature.LAND.with_scale('10m'))  # 图背景的陆地标识
        ax.add_feature(cfeature.COASTLINE.with_scale('10m'), lw=0.25)  # 图背景的海岸线标识
        ax.add_feature(cfeature.OCEAN.with_scale('10m'))  # 图背景的海洋标识
        ax.gridlines(draw_labels=True, dms=True, x_inline=False, y_inline=False)
        im = ax.scatter(data[0] / 3600, data[1] / 3600, s=0.5, c=data[2], cmap='viridis', vmin=0, vmax=500)
        fig.colorbar(im, ax=ax)
        ax.title.set_text(f"Total population of the area is {total_population:.2f}")
        plt.show()


if __name__ == "__main__":
    root = tk.Tk()
    app = ClientApp(root)
    root.mainloop()

```

3.1 代码说明

客户端采用了 `tkinter` 编写 GUI，界面如下：



上图的查询结果如下：



3.1.1 鼠标位置获取

- 左半部分是一个canvas画布，背景图片采用了 NASA 官网的世界地图，并进行了地图与画布坐标的映射与校准。经度（`x`）的范围是从西经180度（-648000秒）到东经180度（648000秒），纬度（`y`）的范围是从南纬90度（-324000秒）到北纬90度（324000秒）。
- 当用户在地图上点击时，`on_canvas_click` 方法被触发，`event.x` 和 `event.y` 获取点击位置的像素坐标。
- 这些像素坐标被转换为经纬度坐标，基于画布的宽度和高度以及地图的实际范围进行缩放。

3.1.2 坐标列表

- 右半部分展示鼠标已经点击的坐标列表，每个坐标通过 `Coordinate` 类来表示。
- `Coordinate` 类用于表示一个地理坐标点，包括秒数形式的 `x`（经度）和 `y`（纬度）以及它们的度、分、秒形式。
- 当用户点击画布时，通过 `add_point` 方法添加一个 `Coordinate` 对象到 `coordinate_list`。
- `remove_last_point` 方法允许用户删除最近添加的坐标点。
- `update_listbox` 方法更新 `Listbox` 控件，显示当前所有坐标点的度、分、秒格式。

3.1.3 绘制图像

- 当用户点击“提交”按钮时，`submit` 方法被触发。
- 该方法从 `coordinate_list` 构造多边形的顶点坐标列表，并发送到服务器。如果服务器响应状态码为200（成功），则处理返回的数据，其中包括人口数据和地理坐标。
- 使用 `matplotlib` 和 `cartopy` 根据返回的数据绘制人口分布图。图表显示了在所选区域内人口分布的热力图。
- 图表的颜色映射（cmap）设置为 'viridis'，并且人口密度数据的范围被设置为0到500，我认为这有助于提升数据的可视化效果。
- `plot_result` 方法使用 `cartopy` 的 `PlateCarree` 投影绘制地图，并添加陆地、海洋和海岸线的特征。图表还包括一个颜色条，显示人口密度的范围。