

Python 程序设计大作业

世界人口查询与可视化工具 – PPPoE

2021211110 杜抒泽

2023 年 1 月 1 日

目录

1 程序设计说明	3
1.1 模块划分与基本功能	3
1.2 显式使用的第三方库	4
1.3 具体模块说明	4
1.3.1 预处理器	4
1.3.2 客户端	4
1.3.3 服务器	4
1.4 HTTP API	5
1.4.1 POST /api/total	5
1.4.2 POST /api/grid	5
1.5 其他稍微有点亮但是没那么亮的点	7
1.5.1 命令行参数解析	7
1.5.2 程序间接口统一	8
2 算法简介	9
2.1 投影算法	9
2.2 求凸包: Andrew 算法	11
2.3 查询凸包内的人口与网格边界处理	12
3 运行结果与说明	14
3.1 安装依赖、预处理、启动服务器与客户端	14
3.2 查询世界 (矩形) [17s]	16
3.3 查询新加坡 (矩形) [2s]	17
3.4 查询中国及周边 (矩形) [10s]	18
3.5 查询中国 (凸包) [10s]	19
3.6 查询山东省 (凸包) [8s]	20
4 总结	21

附录 A 附件清单	22
附录 B 源代码	22
B.1 src/main.py	22
B.2 src/client/client.py	24
B.3 src/client/requests.py	30
B.4 src/config/cli.py	31
B.5 src/config/config.py	32
B.6 src/config/project_meta.py	33
B.7 src/data_manip/data_manip.py	34
B.8 src/projection/projection.py	35
B.9 src/server/algo.py	39
B.10 src/server/server.py	43
B.11 src/util/file.py	46
B.12 src/util/log.py	47
B.13 src/util/time.py	48
B.14 requirements.txt	49

1 程序设计说明

本项目名为 PPPoE (Population Patterns Plotting over Earth)¹, 是一个基于 Python 的人口模式可视化工具。本项目的目的是为了实现一个简单的人口模式可视化工具，能够根据用户的输入，绘制出人口模式的可视化图像。

项目全程使用 Python 实现，在 Python 3.12.0 上编写、调试与测试。全部代码随附在作业中提交。若发现有任何问题请随时联系我。

1.1 模块划分与基本功能

具体业务方面，本项目实现了在给定数据上，给定一个多边形，求此点集的**凸包**并查询凸包内的人口总数，以及将其划分为网格，绘制网格上以人口数为键的热力图的功能。

具体的模块划分如下。所有的模块都是真正意义上具有 `__init__.py` 的 Python module。

- 配置模块

负责项目的基本配置，包括项目名、默认的服务器地址、端口号等。

- 工具模块

负责与项目无关的工具函数，例如日志功能在此实现。

- 数据操纵模块

负责对数据的读取、预处理、压缩等操作。此后被服务器模块继承并实现查询功能。

- 服务器模块

负责服务器的实现，包括网络响应、数据查询等功能。

- 客户端模块

负责客户端的实现，包括网络请求、用户界面、数据可视化等功能。

- 投影算法模块

负责将用户可见的经纬度坐标转换为数据矩阵下标，便于数据的查询。

¹ 借用了以太网点对点协议 Point-to-Point Protocol over Ethernet 的缩写。

1.2 显式使用的第三方库

名称	版本	备注
aiohttp	3.9.1	异步网络请求；最新版本不兼容 Python 3.11+ 故采用旧版 ²
colorlog	6.8.0	logging 模块彩色日志输出支持
matplotlib	3.8.2	热力图绘制
numpy	1.26.3	数据处理与数学运算
sanic	23.12.0	异步网络服务器

1.3 具体模块说明

1.3.1 预处理器

预处理器负责

1. 读取原始数据。
2. 将原始数据进行拼接、处理（求列前缀和）。
3. 将处理后的数据压缩并存入一个新的文件。

1.3.2 客户端

客户端负责

1. 用户交互（GUI）。
2. 将用户输入的经纬度转换为坐标。
3. 将用户请求发送给服务器。
4. 将服务器响应显示给用户。其中包含 matplotlib 画图。

底层数据对客户端完全透明。

1.3.3 服务器

服务器负责

1. 处理客户端请求。

²这个库更新得非常之快，即使是旧版也已经远超这学期前半部分的小作业时的版本了。

2. 进行几种算法，计算给定的凸包的总人口，或者将凸包划分为若干个网格，计算每个网格的人口。

地图的投影对服务器完全透明。

1.4 HTTP API

1.4.1 POST /api/total

获取给定的凸包的总人口。这个接口的性能极高，对于整个地球的查询响应在 200 ms 左右，绝大部分时间为网络通信开销。

请求体形如以下，只包含一个 geojson 格式的 Polygon 字段，但是其内容可以不为凸多边形，也可以不为闭合图形，只需是包含大于等于 3 个点的点集。

```
{  
  "geometry": {  
    "type": "Polygon",  
    "coordinates": [  
      [[0, 0], [21509, 0], [21509, 43109], [0, 43109], [0, 0]]  
    ]  
  }  
}
```

若失败，返回 HTTP 40x / 50x，响应体包含错误信息，形如以下：

```
{  
  "error": "..."  
}
```

若成功则返回 HTTP 200，响应体包含凸包上的点集（按逆时针顺序）以及总人口，形如以下：

```
{  
  "convex": [  
    [0, 0], [21509, 0], [21509, 43109], [0, 43109], [0, 0]  
  ],  
  "population": 7969335108.809966  
}
```

1.4.2 POST /api/grid

返回给定的凸包内被划分为若干个网格后，每个网格的人口，以及总人口。这个接口的性能取决于网格的大小：若分为约 15 个网格查询整个地球则可以在 700 毫秒

内响应；若分为约 1000 个，则能在 3 秒内响应。实际应用中一般都分为成千上万个网格，因此需要等待一段时间。例如，按照默认查询世界地图的 20301 个网格在我的设备上需约 17 秒。

请求体包含一个 geojson 格式的 Polygon 字段以及一个 grid_width 字段。前者与上一个接口相同，其内容可以不为凸多边形，也可以不为闭合图形，只需是包含大于等于 3 个点的点集；后者为一个正整数，表示网格的边长：

```
{  
  "geometry": {  
    "type": "Polygon",  
    "coordinates": [  
      [[0, 0], [21509, 0], [21509, 43109], [0, 43109], [0, 0]]  
    ]  
  },  
  "grid_width": 10000  
}
```

若失败，返回 HTTP 40x / 50x，响应体包含错误信息，形如以下：

```
{  
  "error": "..."  
}
```

若成功则返回 HTTP 200，响应体包含凸包内每个网格的人口，形如以下：

```
{  
  "grid": [  
    [  
      240736970.82886094,  
      463165500.04315114,  
      1901873579.2400494,  
      3899681665.412113,  
      429157.1057982445  
    ],  
    [  
      746399.9912788981,  
      404476362.5776206,  
      686712550.0107422,  
      365635194.3672638,  
      7618316.139038086  
    ],  
    [0.0, 0.0, 0.0, 0.0, 0.0]  
  ]  
}
```

```
]
}
```

1.5 其他稍微有点亮但是没那么亮的点

1.5.1 命令行参数解析

程序通过 argparse 解析命令行参数。运行效果如下图：

```
# fa_555 @ fa555-kooBcaM in ~/Documents/repos/pow on git:main ✘ .venv [20:32:00]
[$ python3 src/main.py -h
usage: main.py [-h] [-v] [-c] [-s] [-a ADDR] [-p PORT] [-l LOG]
               [--log-file LOG_FILE] [--preprocess]

A dual client / server application for tracking world population.

options:
-h, --help            show this help message and exit
-v, --version          show program's version number and exit
-c, --client           Run client
-s, --server           Run server
-a ADDR, --addr ADDR  Server address
-p PORT, --port PORT  Server port
-l LOG, --log LOG      Logging level
--log-file LOG_FILE   Preprocess data

Author: fa_555 <fa_555@foxmail.com>
```

图 1 命令行参数解析

每个参数的含义如下：

参数	默认值	备注
-h, --help	N/A	显示帮助信息
-v, --version	N/A	显示版本信息
-a, --addr	127.0.0.1 (本地回环)	客户端指定的服务器地址
-p, --port	555	客户端指定的服务器端口号
-l, --log	INFO	日志等级
--log-file	run.log	日志文件路径
-c, --client	N/A	动作之一, 启动客户端, 使用 -a 和 -p 参数

-s, --server	N/A	动作之一, 启动服务器, 使用 -p 参数
--preprocess	N/A	动作之一, 进行数据预处理

服务器会读取位于 {ASSETS_DIR}/data.npz 的数据文件, 而预处理器会读取位于 {ASSETS_DIR}/gpw-v4-population-count-rev11_2020_30_sec_asc 的原始数据文件。

1.5.2 程序间接口统一

所有模块都通过面向对象的方式进行了封装。例如, 整个主函数如下, 包含上面提到的三个动作的调用:

```
def main():
    arg_parser = Cli()
    args = arg_parser.parse_args()
    init_file()
    util.init_logging(args.log, args.log_file)

    if not any((args.preprocess, args.client, args.server)):
        logging.error('No action (client, server, preprocess) specified.')
        arg_parser.print_help()

    elif args.preprocess:
        innocent_arg(args.addr, 'addr')
        innocent_arg(args.port, 'port')
        preprocess_main()

    elif args.client:
        addr = require_arg(args.addr, 'addr', CONFIG.DEFAULT_ADDR)
        port = require_arg(args.port, 'port', CONFIG.DEFAULT_PORT)

        from client import Client
        client = Client(addr, port)
        client.run()

    elif args.server:
        innocent_arg(args.addr, 'addr')
        port = require_arg(args.port, 'port', CONFIG.DEFAULT_PORT)

        from server import Server
        server = Server(port)
        server.run()
```

不管它们是怎么搭建起来的（预处理模块通过完全自己编写的 `DataManipulator` 运行，客户端类继承了 `tk.Tk`，服务器类组合了一个 `sanic.Sanic` app 和一个 `sanic.worker.loader.AppLoader` 加载器，它们都能以类似的方式启动，实现细节对外界透明，令人安心。

2 算法简介

2.1 投影算法

确定这个数据的投影算法时非常奇怪。有显然的客观事实：

一个球体不可能被直接展开到平面。

似乎所有同学都没有理会这个客观事实，直接把经纬度当作平面坐标使用。更加离奇的是这种方法得到的结果是正确的。

实际上这个数据当然是投影得到的，而非直接展开的。观察数据源文件中存在大量 -9999 项，这些项代表了平面对应的位置不在球面上存在（投影时缺失），同时这个数据源文件的长宽比是严格的 2:1，这显然和地球的实际形状不符。

查阅数据的[文档](#)可知，数据采用了 Mollweide 投影。这就和数据的表现对得上了。Mollweide 投影是一种等面积伪圆柱地图投影，以角度和形状的准确性换取面积比例的准确性，同时得到的地图是一个长短轴之比恰好为 2:1 的椭圆。

然而该数据似乎在后来又进行了重新修正，因此数据维持了离心率为 $\frac{\sqrt{3}}{2}$ 的椭圆的形状，但可以直接将经纬度进行线性映射获取矩阵位置。

不管怎样，这里还是介绍一下 Mollweide 投影算法的实现，尽管它在最终版的代码中被废弃了。

假设我们有点 P ，其球面坐标为 (λ, φ) ，其中 λ 为经度， φ 为纬度。我们希望将其投影到平面上，得到其平面坐标 (x, y) 。此时可以采取以下的方式计算：

$$\begin{aligned} x &= \frac{2}{\pi}(\lambda - \lambda_0) \cos \theta \\ y &= \sin \theta \end{aligned}$$

其中 λ_0 是本初子午线的经度， θ 是中间变量，满足以下方程：

$$2\theta + \sin 2\theta = \pi \sin \varphi$$

可以看到，求 θ 并不是一个非常简单的过程。好在函数 $f(x) = x + \sin x$ 是单调的，我们可以直接通过二分在对数时间内找到 θ 的近似值。此外，我们还可以使用牛顿迭代：

$$\theta_0 = \varphi$$

$$\theta_{n+1} = \theta_n - \frac{2\theta_n + \sin 2\theta_n - \pi \sin \varphi}{2 + 2 \cos 2\theta_n}$$

迭代大约 10 次即可达到 10^{-8} 的精度。

反之，假设我们有点 P ，其平面坐标为 (x, y) ，我们希望将其投影到球面上，得到其球面坐标 (λ, φ) 。此时可以采取以下的方式计算：

$$\varphi = \arcsin \frac{2\theta + \sin 2\theta}{\pi}$$

$$\lambda = \lambda_0 + \frac{\pi x}{2 \cos \theta}$$

其中 θ 可以容易地求得：

$$\theta = \arcsin y$$

以上所有字母的取值范围分别为 $x \in [-2, 2]$, $y \in [-1, 1]$, $\lambda \in [-\pi, \pi]$, $\varphi \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ 。

绘图得到 -175° 、 -90° 、 30° 和 120° 四条经线上所有整数纬度点的采样（红色椭圆为整个投影的合法区域）：

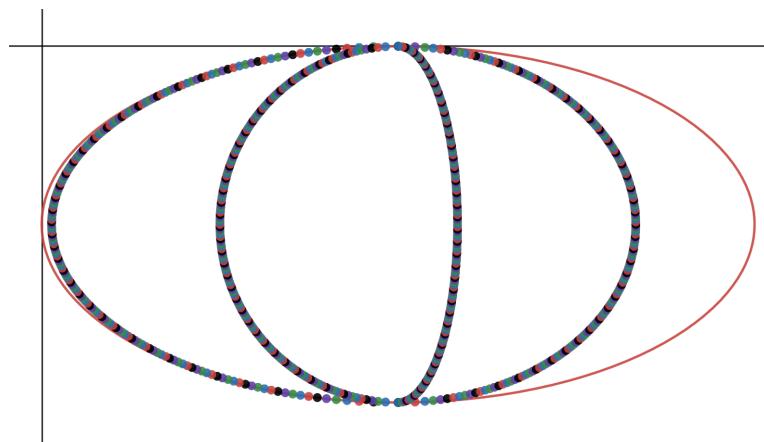


图 2 Mollweide 投影下的四条经线采样

代码实现：

```
def project(point: Point3D) → Point2D:
    def get_theta_binary(phi: float) → float:
        rhs = pi * sin(phi) # range: [-pi, pi], strictly increasing

        # then 2theta is in range [-pi, pi]
        lb = -pi
        rb = pi
        while rb - lb > 1e-8:
            mid = (lb + rb) / 2
```

```

        if mid + sin(mid) < rhs:
            lb = mid
        else:
            rb = mid

    # now 2theta approx (lb + rb) / 2
    return (lb + rb) / 4

def get_theta_newton(phi: float) → float:
    theta_ = phi
    for _ in range(10):
        tmp = 2 * theta_
        theta_ -= ((tmp + sin(tmp) - pi * sin(phi)) / (2 + 2 * cos(tmp)))
    return theta_

    if abs(point.latitude) == pi / 2:
        return Point2D.from_portion(0, point.latitude / (pi / 2))

    # 2theta + sin(2theta) = pi * sin(latitude)
    theta = get_theta_newton(point.latitude)
    x = 2 / pi * point.longitude * cos(theta)
    y = sin(theta)
    return Point2D.from_portion(x, y)

def inv_project(point: Point2D) → Point3D:
    x = point.y / CONSTANT - 2
    y = point.x / CONSTANT - 1

    theta = asin(y)
    lon = pi * x / (2 * cos(theta))
    lat = asin((2 * theta + sin(2 * theta)) / pi)
    return Point3D.from_rad(lon, lat)

```

2.2 求凸包：Andrew 算法

Andrew 算法是一种求凸包的算法，其时间复杂度为 $\mathcal{O}(n \log n)$ 。其基本思想是通过单调栈分两次求出凸包的下凸壳和上凸壳。

具体地，首先将所有点以横坐标为第一关键字，纵坐标为第二关键字排序。显然排序后最小的元素和最大的元素一定在凸包上。而且因为是凸多边形，我们如果从一个点出发逆时针走，轨迹总是“左拐”的，否则就说明这一段不在凸包上。因此我们可以用一个单调栈来维护上下凸壳。求凸壳时，一旦发现即将进栈的点 P 和栈顶的两个

点 S_1, S_2 行进的方向向右旋转（叉积小于 0），则弹出栈顶，回到上一步，继续检测，直到叉积大于 0 或者栈内仅剩一个元素为止。

实现如下：

```
def calc_convex(p: List[Point2D]) -> List[Point2D]:
    p.sort(key=lambda p: (p.x, p.y))
    used = [False] * len(p)

    stack = [0]
    # used[0] = True

    # 求下凸壳
    for i in range(1, len(p)):
        while len(stack) > 1 and cross(
            p[stack[-1]] - p[stack[-2]], p[i] - p[stack[-1]])
            ) ≤ 0:
            used[stack.pop()] = False
        stack.append(i)
        used[i] = True

    base = len(stack)
    # 求上凸壳
    for i in range(len(p) - 2, -1, -1):
        if used[i]:
            continue
        while len(stack) > base and cross(
            p[stack[-1]] - p[stack[-2]], p[i] - p[stack[-1]])
            ) ≤ 0:
            used[stack.pop()] = False
        stack.append(i)
        used[i] = True

    return [p[i] for i in stack]
```

时间复杂度为 $\mathcal{O}(n \log n)$ ，瓶颈在排序。

2.3 查询凸包内的人口与网格边界处理

感觉讲的必要性不大，似乎还没有求凸包复杂。

在数据预处理时，我们将原矩阵按每一行求前缀和。这样，我们可以在 $\mathcal{O}(1)$ 的时间内查询每行内任意区间的人口总数，同时不失去 $\mathcal{O}(1)$ 的时间内查询任意点的人口总数的能力。数据使用 numpy 压缩为 .npz 格式后不足 600MB，服务器程序启动并读入内存后占用内存约 3.92GB，空间利用效率较高。

对于每个凸包，我们枚举凸包上的每条边，并统计它与每条直线 $x = m$ 的交点。凸包的性质在这里显现：任意一条直线与凸包至多只会有两个交点，而这条线上的人口只可能是这两个交点之间的人口之和。因此我们可以在 $\mathcal{O}(n)$ 的时间内求出凸包上所有点的人口之和，其中 n 是数据矩阵较短的一边的长度。

代码实现如下：

```
def calc_whole_convex(convex, data_accessor):
    intersections = MinMaxDict()

    for i in range(0, len(convex) - 1):
        p1 = convex[i]
        p2 = convex[i + 1]
        x1 = min(p1.x, p2.x)
        y1 = min(p1.y, p2.y)
        x2 = max(p1.x, p2.x)
        y2 = max(p1.y, p2.y)
        if y1 == y2:
            for x in range(x1, x2 + 1):
                intersections.update(x, y1)
        elif x1 == x2:
            for y in range(y1, y2 + 1):
                intersections.update(x1, y)
        else:
            k = (y1 - y2) / (x1 - x2)
            b = y1 - k * x1
            for x in range(x1, x2 + 1):
                y = int(k * x + b)
                intersections.update(x, y)

    ans = 0
    for x in intersections.get_dict().keys():
        min_, max_ = intersections.get(x)
        ans += data_accessor.query(x, min_, max_)
    return ans
```

计算网格的答案时，我采取了一种比较简单的方法：对于最大、最小经度、纬度范围内的每个网格，求其四个顶点，若这四个顶点都在凸包内，则这个网格需要被统计。否则，网格的这一个元素被直接置为 -1 以标记其不在凸包内。

求一个点在不在凸包内有 $\mathcal{O}(n)$ 和 $\mathcal{O}(\log n)$ 的算法。前者只需对于凸包的每条边，计算待求点和这两端点的叉积是否同号即可。后者是对其的二分查找优化，找到凸包上第一个点 P ，使得 P 和待求点的连线与凸包上的边的叉积同号，然后再找到凸包上

第一个点 Q , 使得 Q 和待求点的连线与凸包上的边的叉积异号。若 P 和 Q 之间的距离为 0, 则待求点在凸包上; 否则, 待求点不在凸包上。

代码实现就不放了。

3 运行结果与说明

以下展示多个有实际意义的运行结果及运行时间。运行时间根据程序运行日志获得, 没有展示截图; 其在各个机器上也不尽相同, 仅作说明。

3.1 安装依赖、预处理、启动服务器与客户端

安装依赖时可以选择全局站点或者虚拟环境, 亦或者使用 conda 等其他工具。我自己还是 venv 用得最顺手。创建一个新的 venv 并激活, 使用 pip -r requirements.txt 安装依赖。

将提供的数据文件解包放入资源目录下 (默认是 ./assets), 运行 python3 src/main.py --preprocess 即可进行预处理。预处理的结果会被保存在 ./assets/data.npz 中。

```
# fa_555  fa555-kooBcaM in ~/Documents/repos/pow on git:main x .venv [21:02:07] C:130
$ python3 src/main.py --preprocess
21:02:11 INFO Logging service initialised.
21:02:11 INFO Preprocessing data ...
21:02:11 INFO Loading file gpw_v4_population_count_rev11_2020_30_sec_1.asc ...
21:02:16 INFO File gpw_v4_population_count_rev11_2020_30_sec_1.asc loaded.
21:02:16 INFO Loading file gpw_v4_population_count_rev11_2020_30_sec_2.asc ...
21:02:20 INFO File gpw_v4_population_count_rev11_2020_30_sec_2.asc loaded.
21:02:20 INFO Loading file gpw_v4_population_count_rev11_2020_30_sec_3.asc ...
21:02:25 INFO File gpw_v4_population_count_rev11_2020_30_sec_3.asc loaded.
21:02:25 INFO Loading file gpw_v4_population_count_rev11_2020_30_sec_4.asc ...
21:02:30 INFO File gpw_v4_population_count_rev11_2020_30_sec_4.asc loaded.
21:02:30 INFO Loading file gpw_v4_population_count_rev11_2020_30_sec_5.asc ...
21:02:35 INFO File gpw_v4_population_count_rev11_2020_30_sec_5.asc loaded.
21:02:35 INFO Loading file gpw_v4_population_count_rev11_2020_30_sec_6.asc ...
21:02:39 INFO File gpw_v4_population_count_rev11_2020_30_sec_6.asc loaded.
21:02:39 INFO Loading file gpw_v4_population_count_rev11_2020_30_sec_7.asc ...
21:02:44 INFO File gpw_v4_population_count_rev11_2020_30_sec_7.asc loaded.
21:02:44 INFO Loading file gpw_v4_population_count_rev11_2020_30_sec_8.asc ...
21:02:49 INFO File gpw_v4_population_count_rev11_2020_30_sec_8.asc loaded.
21:02:49 INFO Concatenating data ...
21:02:51 INFO Data concatenated.
21:02:51 INFO Transforming data ...
21:03:00 INFO Sum: 7969432977.363524
21:03:00 INFO Saving preprocessed data ...
21:03:31 INFO Data saved.
21:03:31 INFO Data preprocessed in 80.244 s.
```

图 3 预处理

运行 `python3 src/main.py --server` 即可启动服务器。服务器默认监听 555 端口，可以通过 `python3 src/main.py --server --port 1234` 指定端口号。服务器启动后会读取预处理后的数据文件，占用内存稳定在约 3.92GB。

```
$ python3 src/main.py -s
21:05:58 INFO Logging service initialised.
21:05:58 INFO No `port` specified, default to 555
21:05:58 INFO Loading preprocessed data ...
[2024-01-05 21:06:05 +0800] [96256] [INFO]
Sanic v23.12.0
Goin' Fast @ http://0.0.0.0:555
Build Fast. Run Fast.
```

21:06:05 INFO

```
Sanic v23.12.0
Goin' Fast @ http://0.0.0.0:555
Build Fast. Run Fast.
```

[2024-01-05 21:06:06 +0800] [96256] [DEBUG] Creating multiprocessing context using 'spawn'
21:06:06 DEBUG Creating multiprocessing context using 'spawn'
[2024-01-05 21:06:06 +0800] [96256] [DEBUG] Starting a process: Sanic-Server-0-0
21:06:06 DEBUG Starting a process: Sanic-Server-0-0
[2024-01-05 21:06:06 +0800] [96256] [DEBUG] Starting a process: Sanic-Reloader-0
21:06:06 DEBUG Starting a process: Sanic-Reloader-0
[2024-01-05 21:06:14 +0800] [96275] [DEBUG] Process ack: Sanic-Server-0-0 [96275]
DEBUG:sanic.root:Process ack: Sanic-Server-0-0 [96275]
[2024-01-05 21:06:14 +0800] [96275] [INFO] Starting worker [96275]
INFO:sanic.server:Starting worker [96275]

图 4 启动服务器³

运行 `python3 src/main.py --client` 即可启动客户端。客户端默认连接到本地回环的 555 端口，可以通过 `python3 src/main.py --client --addr a.b.c.d --port`

³Sanic 的日志和我自己的日志服务用了同一个模块，它们在打架！

5555 指定服务器地址和端口号。客户端启动后不会读取本地的任何文件，只有 GUI 在消耗资源。

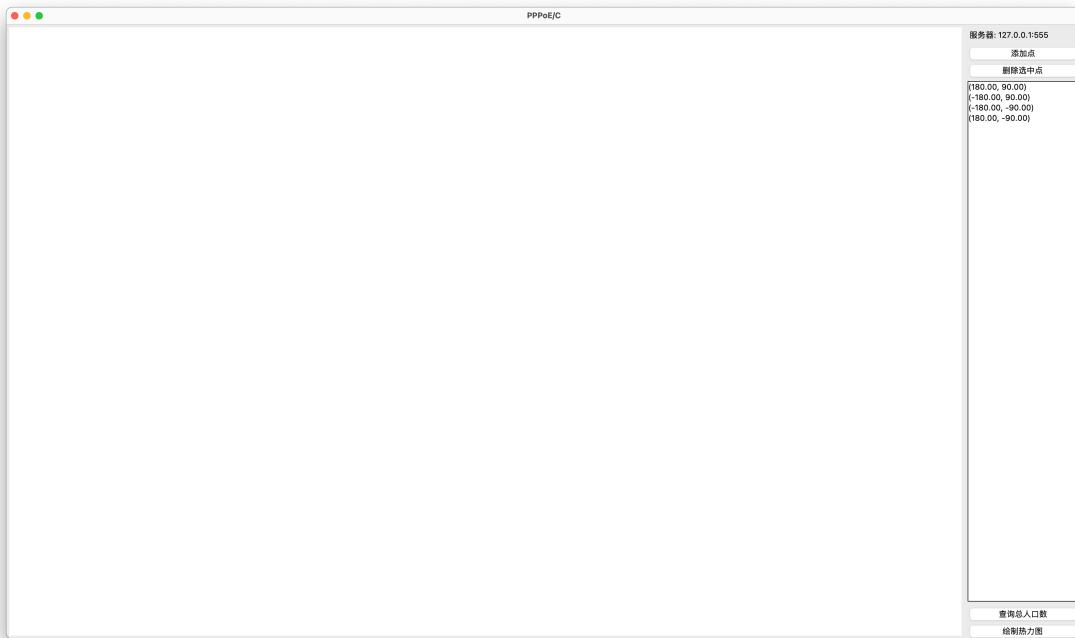


图 5 启动客户端

可以看到程序的主体界面现在是空白的。右侧有四个按钮和一个列表，可以进行添加、删除坐标点的操作，或者向服务器查询。这些功能在此不过多演示。右侧的点列表可以选择若干个预设，只需要在 `client/client.py` 中 `Client.mock_data` 函数中增删注释即可。图中展示的是全世界地图查询的预设。注意，展示的热力图中人口数经过取对数，只展示相对数量。

3.2 查询世界（矩形） [17s]

坡子地少人多，地理特征简单，恰巧今年暑假在 NUS 呆了一阵子，跟坡子还是有点熟的，所以拿它举个例子。

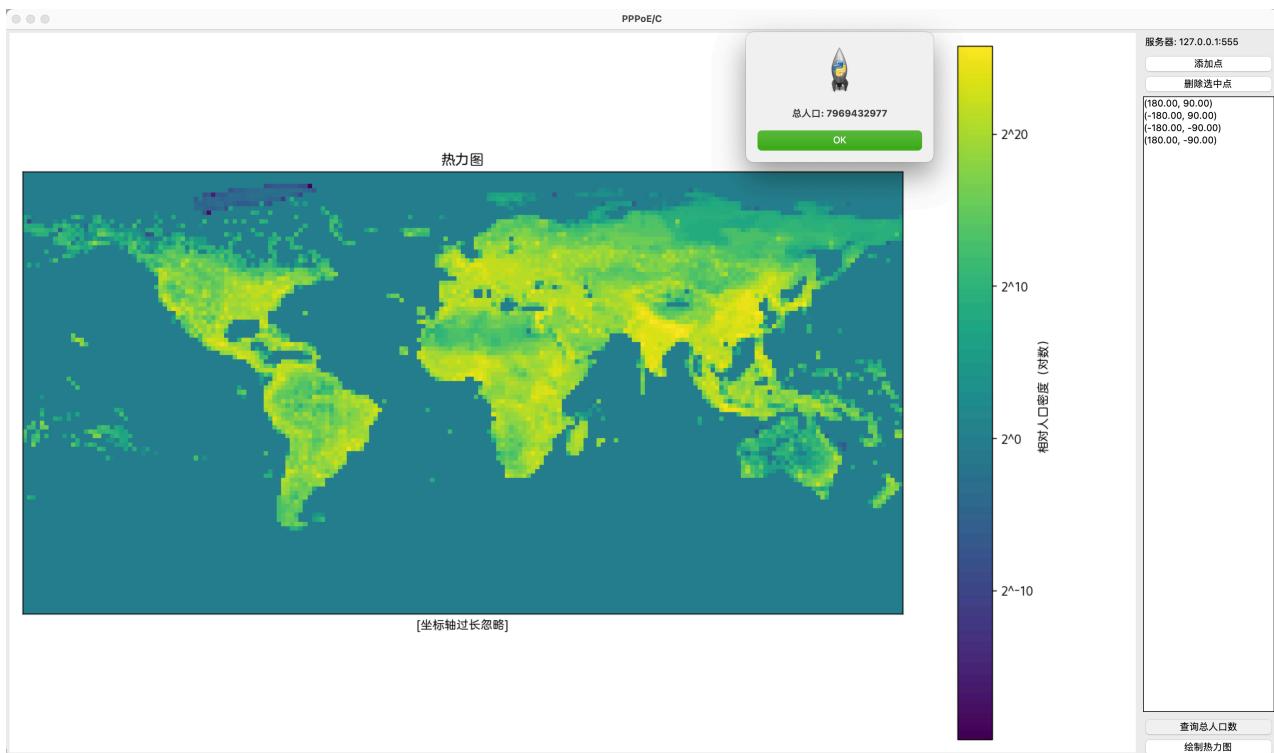


图 6 查询全世界

这个例子耗时 17s。总人口数约 79.7 亿，符合预期。

3.3 查询新加坡（矩形） [2s]

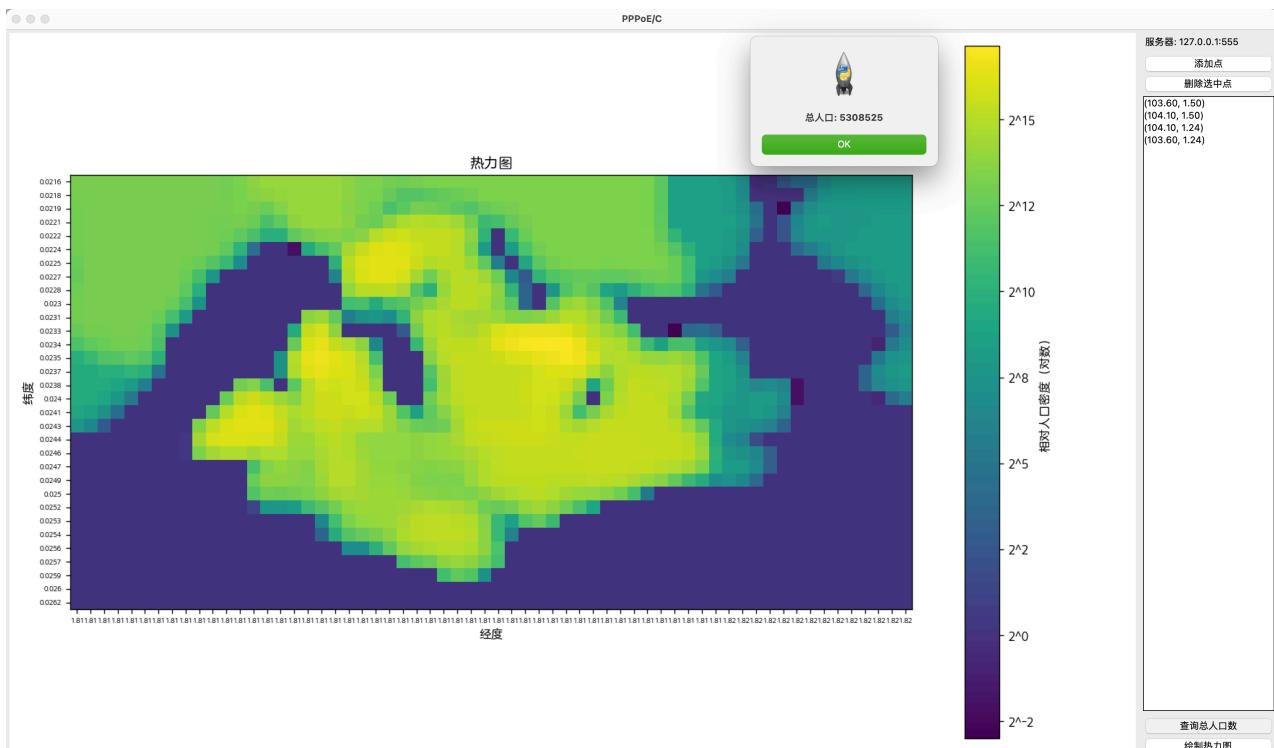


图 7 查询新加坡

我们来具体分析这个结果。首先找到一张新加坡地图：

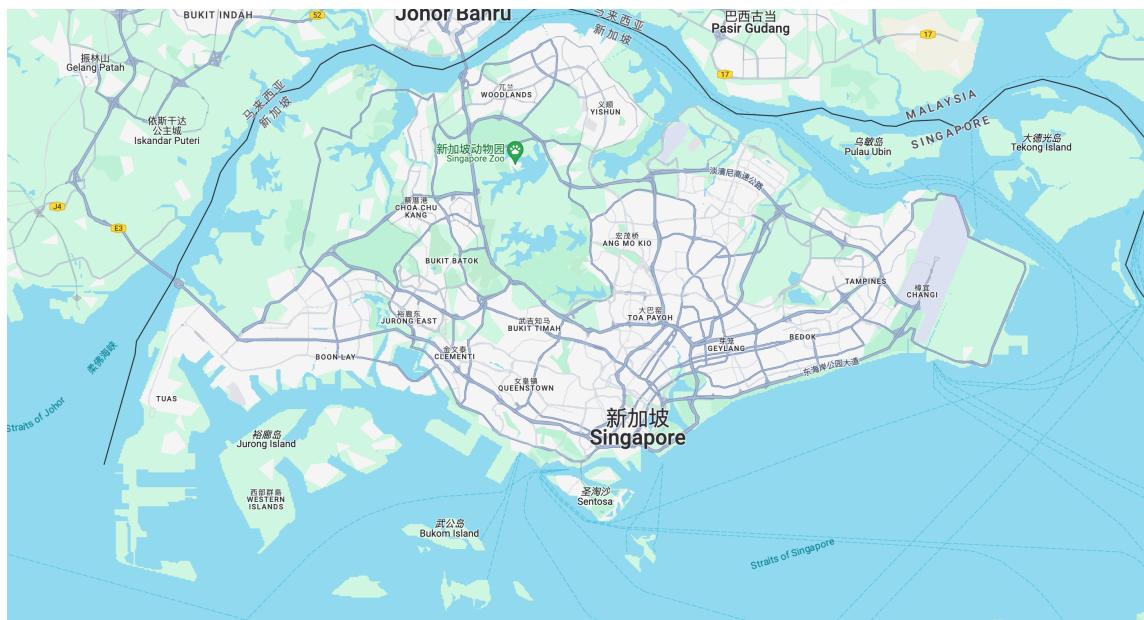


图 8 新加坡地图

与上图的形状是比较接近的。热力图中左下角无人口处是一些小岛，左上角是水库和自然公园等。最右侧有人口但人数较少的是樟宜机场，而偏右侧的无人口区域是巴耶利峇自然保护区（含有沼泽和军用机场）。这些都是符合预期的。

再搜索一下新加坡的总人口：

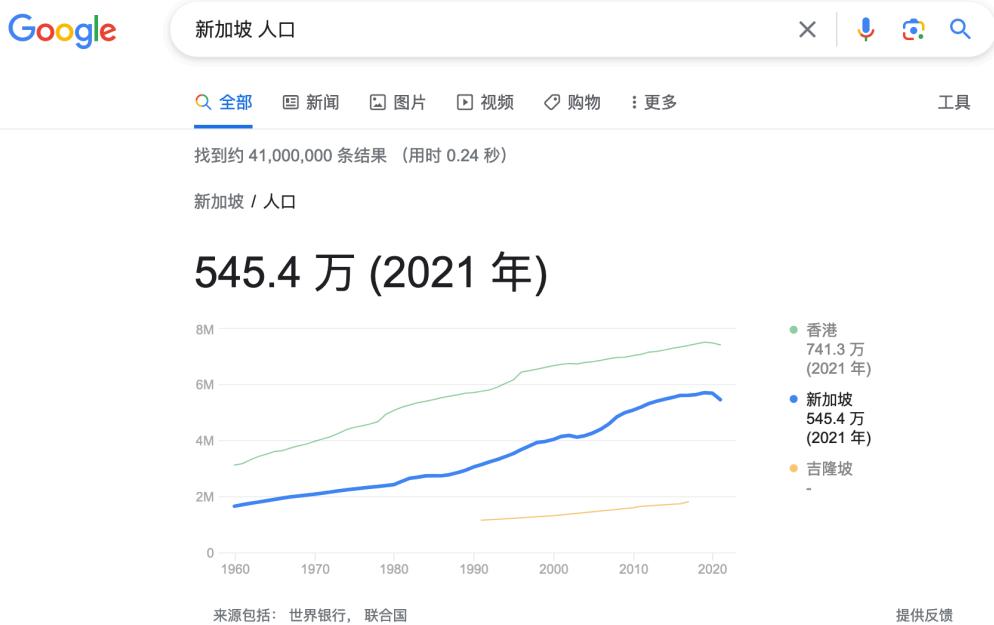


图 9 新加坡总人口

这与图中展示的 530 万人是比较接近的。

3.4 查询中国及周边（矩形） [10s]

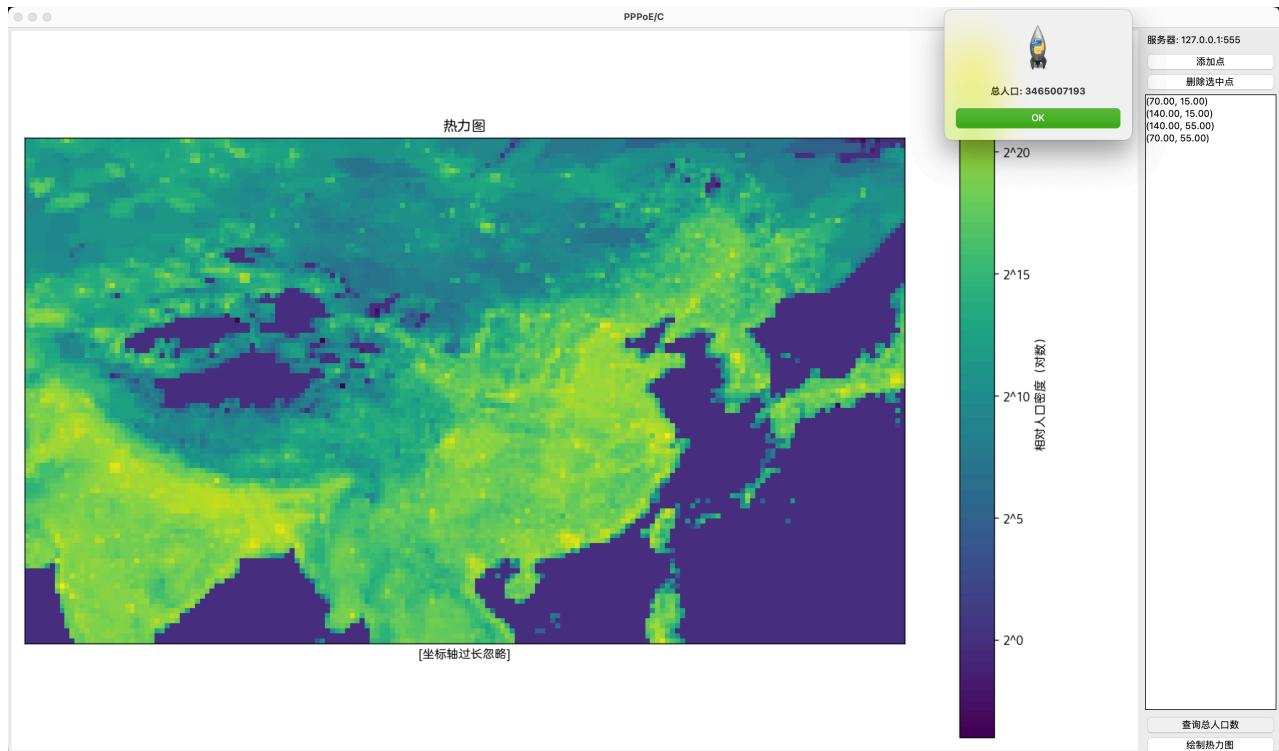


图 10 查询中国及周边

这张图包含了中国的领土全境（不含南海诸岛）以及周围的国家（含印度恒河平原）。可以看到，东部沿海地区人口密度较高，长三角和珠三角尤甚；西部人口密度较低，尤其是新疆和西藏出现了大片无人区。印度的恒河平原也是世界上人口密度最高的地区之一。这些都与预期相符。

由于包含了中国全境和恒河平原，这张图的总人口数超过了 30 亿，符合预期。

3.5 查询中国（凸包） [10s]

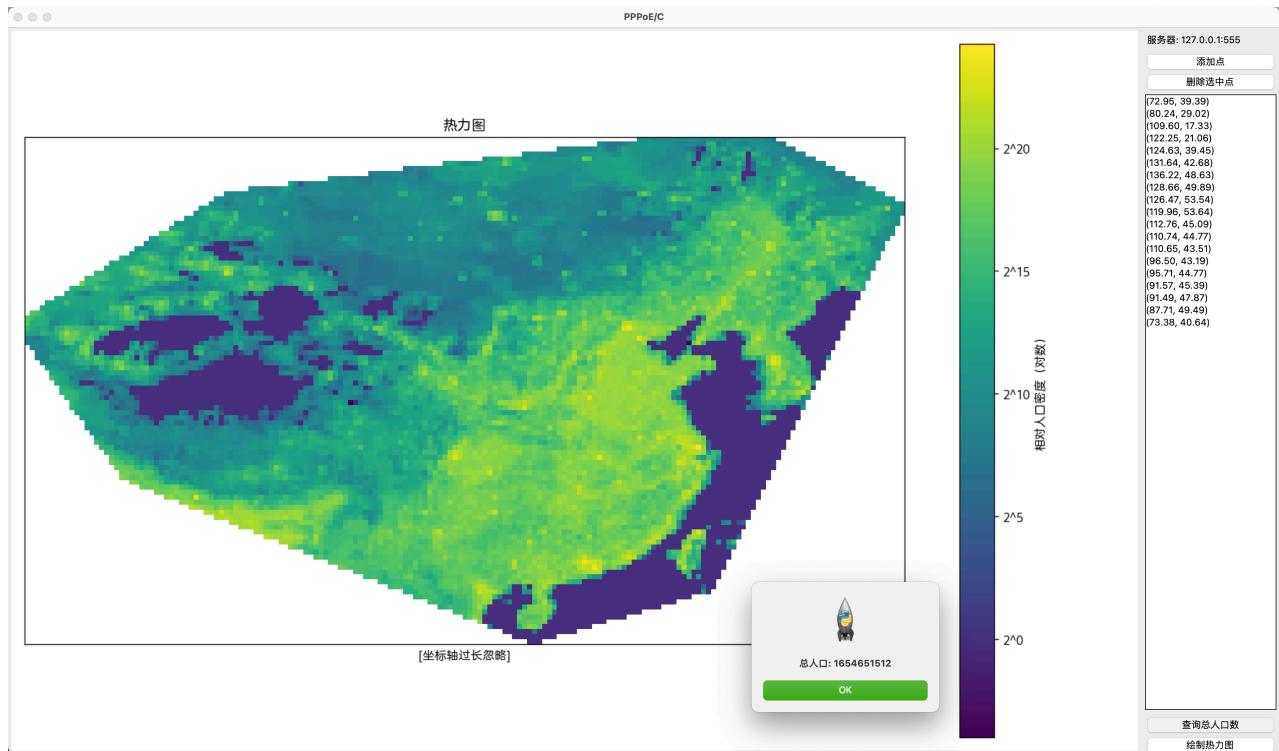


图 11 查询中国

用一个凸包括出中国全境，包含了尼泊尔、外蒙古和朝鲜半岛，这是没有办法的事情。查询到总人口为 16.5 亿，略多于中国人口，符合预期。

3.6 查询山东省（凸包） [8s]

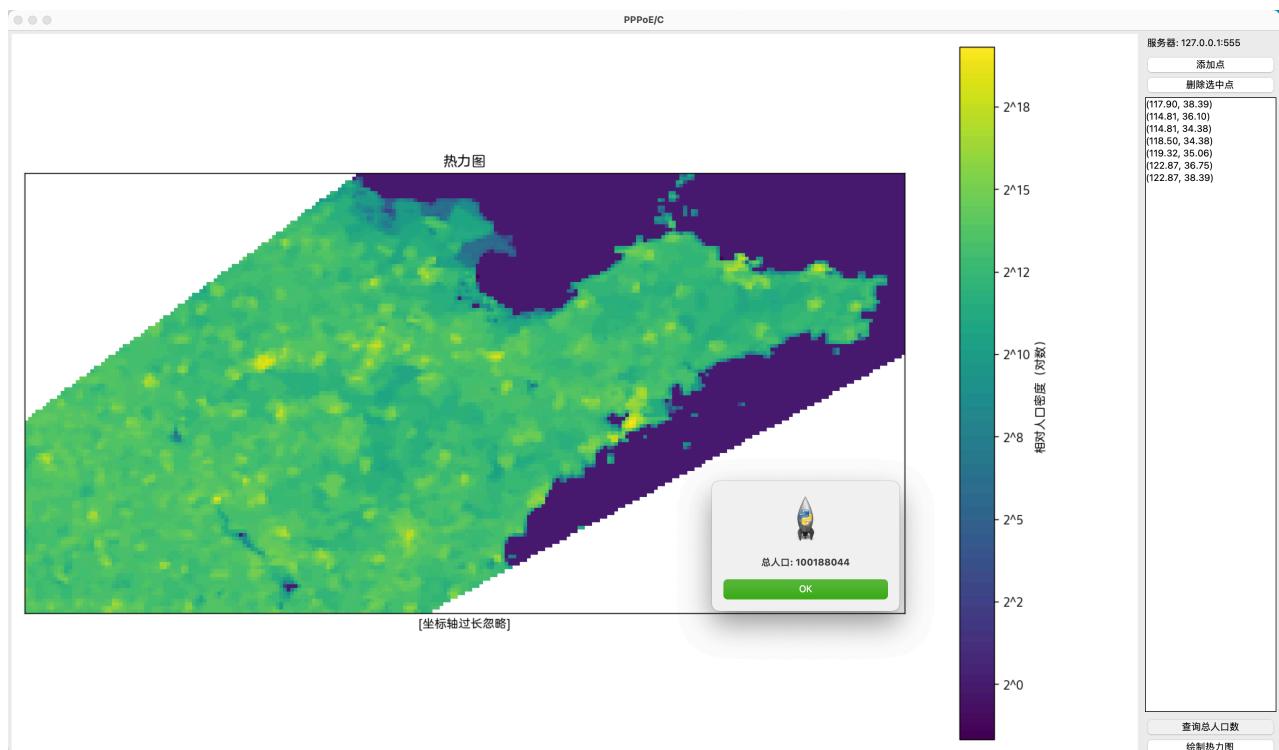


图 12 查询山东省

山东来的孩子迫不及待要查查自己家。

可以看到，山东省的人口分布是比较均匀的。左侧的一个大黄点和右侧沿海的一个包围住一片水域的大黄点分别是济南和青岛（被青岛包围住的水域是胶州湾）。鲁南的一部分没有人口的区域是微山湖，微山湖东北的一个稍大的黄点是临沂市。作临沂市和微山湖的中垂线向西北方向，略微偏东侧的一片浅绿色平原是我家。

查阅资料知山东省 2020 年总人口为 1.01 亿⁴，和图中的查询数据（1.001 亿）也是比较接近的。

4 总结

本次大作业至此结束。项目实现了一个简单的人口模式可视化工具，能够根据网络上的数据和绘制出用户需要的地区的人口密度可视化图像（）热力图。项目采用了 Python 作为开发语言，使用了 aiohttp、matplotlib、sanic 等第三方库，实现了一对简单的 HTTP 服务器和客户端。本项目未来如有后续，仍可有以下改进方向：

- 继续优化算法，提高服务器查询性能。感觉现在查整个世界地图分成 20000+ 个区域需要 17s 还是稍微有点慢⁵。
- 优化客户端的用户体验。目前客户端的用户体验还是比较差的，例如没有实现进度条、丰富的提示信息等用户友好界面。
- 支持更多通用的数据格式。目前项目的数据预处理器只能处理特定的一种数据格式。
- 接入更多数据源或 API，例如接入地图 API，在热力图下方叠一层地图，可以更加直观地将人口密度和行政区划、地理位置联系起来。

⁴参见[第七次全国人口普查公报](#)。

⁵慢吗？

附录 A 附件清单

```
PPPoE
└── requirements.txt
└── src
    ├── client
    │   ├── __init__.py
    │   ├── client.py
    │   └── requests.py
    ├── config
    │   ├── __init__.py
    │   ├── cli.py
    │   ├── config.py
    │   └── project_meta.py
    ├── data_manip
    │   ├── __init__.py
    │   └── data_manip.py
    ├── main.py
    ├── projection
    │   ├── __init__.py
    │   ├── projection.py
    │   └── projection_test.py
    ├── server
    │   ├── __init__.py
    │   ├── algo.py
    │   └── server.py
    └── util
        ├── __init__.py
        ├── file.py
        ├── log.py
        └── time.py
```

附录 B 源代码

下面这些都是排版工具用代码生成的。谁闲着没事粘贴这些。

B.1 src/main.py

```
# -*- coding: utf-8 -*-

from config import Cli, CONFIG
import util

import logging
```

```
def init_file():
    """
    Initialize files.

    :return: None
    """
    util.try_mkdir(CONFIG.INSTANCE_DIR)

def preprocess_main():
    logging.info('Preprocessing data ...')
    start = util.get_cur_time_ms()

    from data_manip import DataManipulator
    dm = DataManipulator()
    dm.load_raw()
    dm.process()
    dm.save()

    end = util.get_cur_time_ms()
    logging.info(f'Data preprocessed in {(end - start) / 1000} s.')

def require_arg(arg, name, default):
    if arg is None:
        logging.info(f'No `{name}` specified, default to {default}')
        return default
    return arg

def innocent_arg(arg, name):
    if arg is not None:
        logging.warning(f'`{name}` specified but ignored.')

def main():
    arg_parser = Cli()
    args = arg_parser.parse_args()
    init_file()
    util.init_logging(args.log, args.log_file)

    if not any((args.preprocess, args.client, args.server)):
        logging.error('No action (client, server, preprocess) specified.')
```

```
arg_parser.print_help()

elif args.preprocess:
    innocent_arg(args.addr, 'addr')
    innocent_arg(args.port, 'port')
    preprocess_main()

elif args.client:
    addr = require_arg(args.addr, 'addr', CONFIG.DEFAULT_ADDR)
    port = require_arg(args.port, 'port', CONFIG.DEFAULT_PORT)

    from client import Client
    client = Client(addr, port)
    client.run()

elif args.server:
    innocent_arg(args.addr, 'addr')
    port = require_arg(args.port, 'port', CONFIG.DEFAULT_PORT)

    from server import Server
    server = Server(port)
    server.run()

if __name__ == '__main__':
    main()
```

B.2 src/client/client.py

```
# -*- coding: utf-8 -*-
import threading

from .requests import *
from config.project_meta import __project__
from projection import Point2D, Point3D, project

from tkinter import messagebox, simpledialog
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

import numpy as np
import tkinter as tk
import matplotlib.pyplot as plt
import logging
```

```
class AddPointDialog(simpledialog.Dialog):
    def __init__(self, parent, title=None):
        self.result = None
        self.latitude_entry = None
        self.longitude_entry = None
        self.latitude = None
        self.longitude = None
        super().__init__(parent, title)

    def body(self, master):
        tk.Label(master, text='经度 (-180 ~ 180): ').grid(row=0)
        tk.Label(master, text='纬度 (-90 ~ 90): ').grid(row=1)

        self.longitude_entry = tk.Entry(master)
        self.latitude_entry = tk.Entry(master)

        self.longitude_entry.grid(row=0, column=1)
        self.latitude_entry.grid(row=1, column=1)

        return self.longitude_entry

    def apply(self):
        try:
            longitude = float(self.longitude_entry.get())
            latitude = float(self.latitude_entry.get())
            self.result = Point3D.from_deg(longitude, latitude)
        except ValueError as e:
            messagebox.showerror('出错了!', f'输入不合法! \n{e}')
            self.result = None

class Client(tk.Tk):
    def __init__(self, addr, port):
        super().__init__()

        self.request_grid_button = None
        self.request_total_button = None
        self.point_listbox = None
        self.delete_point_button = None
        self.add_point_button = None
        self.plot = None
        self.server_label = None
```

```
self.server = (addr, port)
self.title(f'{__project__}/C')
self.points = []
self.setup_ui()
self.mock_data()

self.loop = asyncio.new_event_loop()
threading.Thread(target=self.run_loop, args=(self.loop,))

def run_loop(self, loop):
    asyncio.set_event_loop(loop)
    loop.run_forever()

def mock_data(self):
    mock_coord = (
        # 世界
        # (180, 90), (-180, 90), (-180, -90), (180, -90),

        # 山东
        (117.90, 38.39), (114.81, 36.10), (114.81, 34.38),
        (118.50, 34.38), (119.32, 35.06), (122.87, 36.75),
        (122.87, 38.39),

        # 中国 (粗糙)
        # (70, 15), (140, 15), (140, 55), (70, 55),

        # 新加坡
        # (103.60, 1.50), (104.10, 1.50), (104.10, 1.24), (103.60, 1.24),

        # 中国 (精细)
        # (72.95, 39.39), (80.24, 29.02), (109.60, 17.33), (122.25, 21.06),
        # (124.63, 39.45), (131.64, 42.68), (136.22, 48.63),
        # (128.66, 49.89), (126.47, 53.54), (119.96, 53.64),
        # (112.76, 45.09), (110.74, 44.77), (110.65, 43.51), (96.50, 43.19),
        # (95.71, 44.77), (91.57, 45.39), (91.49, 47.87), (87.71, 49.49),
        # (73.38, 40.64),
    )
    for coord in mock_coord:
        self.points.append(Point3D.from_deg(coord[0], coord[1]))
        self.point_listbox.insert(tk.END, str(self.points[-1]))
        self.point_listbox.see(tk.END)
```

```
def setup_ui(self):
    self.server_label = tk.Label(
        self, text=f'服务器: {self.server[0]}:{self.server[1]}')
    self.server_label.grid(
        row=0, column=1, padx=5, pady=5, columnspan=2, sticky='w')
)

self.plot = FigureCanvasTkAgg(
    plt.Figure(figsize=(13, 8), dpi=120), self)
self.plot.get_tk_widget().grid(
    row=0, column=0, rowspan=6, padx=5, pady=5, sticky='nsew')

self.add_point_button = tk.Button(
    self, text='添加点', command=self.add_coordinate)
self.add_point_button.grid(
    row=1, column=1, padx=5, sticky='ew')

self.delete_point_button = tk.Button(
    self, text='删除选中点', command=self.delete_selected)
self.delete_point_button.grid(
    row=2, column=1, padx=5, sticky='ew')

self.point_listbox = tk.Listbox(self, height=50)
self.point_listbox.grid(
    row=3, column=1, padx=5, pady=5, sticky='nsew')

self.request_total_button = tk.Button(
    self, text='查询总人口数',
    command=lambda: self.loop.run_until_complete(self.request_total()))
self.request_total_button.grid(
    row=4, column=1, padx=5, sticky='ew')

self.request_grid_button = tk.Button(
```

```
        self, text='绘制热力图',
        command=lambda: self.loop.run_until_complete(self.request_grid())
    )
    self.request_grid_button.grid(
        row=5, column=1, padx=5, sticky='ew'
)

def add_coordinate(self):
    dialog = AddPointDialog(self, '添加坐标点')
    if dialog.result is not None:
        logging.info(f'Point added: {dialog.result}')
        self.points.append(dialog.result)
        self.point_listbox.insert(tk.END, str(dialog.result))
        self.point_listbox.see(tk.END)

def delete_selected(self):
    try:
        index = self.point_listbox.curselection()[0]
        self.point_listbox.delete(index)
        self.points.pop(index)
    except IndexError:
        messagebox.showerror('出错了!', '没有选中坐标点。')

async def request_total(self):
    geojson = encap_geojson([project(p) for p in self.points])
    logging.info(f'Requesting total population: {geojson}')
    url = f'{self.server[0]}:{self.server[1]}/api/total'
    response = await request(url, {'geometry': geojson})

    logging.info(f'Response: {response}')
    if 'error' in response:
        messagebox.showerror('出错了!', response['error'])
    else:
        messagebox.showinfo(
            '总人口',
            f'总人口: {int(response["population"])}'
        )

async def request_grid(self):
    points2d = [project(p) for p in self.points]
    max_x = max(p.x for p in points2d)
    min_x = min(p.x for p in points2d)
    max_y = max(p.y for p in points2d)
```

```
min_y = min(p.y for p in points2d)
grid_count = min(100, max(max_x - min_x, max_y - min_y) // 3)
width = max(1, (max_x - min_x) // grid_count)

geojson = encap_geojson([project(p) for p in self.points])
logging.info(f'Requesting grid population: {geojson}')
url = f'{self.server[0]}:{self.server[1]}/api/grid'
response = await request(url, {
    'geometry': geojson,
    'grid_width': width
})

logging.info(f'Response: {response}')
if 'error' in response:
    messagebox.showerror('出错了!', response['error'])
else:
    self.plot_data(response['grid'])

def plot_data(self, data):
    max_lat = max(p.latitude for p in self.points)
    min_lat = min(p.latitude for p in self.points)
    max_lon = max(p.longitude for p in self.points)
    min_lon = min(p.longitude for p in self.points)
    data = np.array(data)
    data = np.where(data == -1, np.nan, data)
    data = np.log2(data, out=np.zeros_like(data), where=(data != 0))

    # draw heatmap
    self.plot.figure.clear()
    ax = self.plot.figure.subplots()
    ax.set_title('热力图')
    ax.invert_yaxis()

    if data.shape[0] <= 90 and data.shape[1] <= 90:
        ax.set_xlabel('经度')
        ax.set_ylabel('纬度')
        ax.set_xticks(range(data.shape[1]))
        ax.set_yticks(range(data.shape[0]))

        lon_sticks = np.linspace(min_lon, max_lon, data.shape[1])
        lat_sticks = np.linspace(min_lat, max_lat, data.shape[0])
        ax.set_xticklabels([f'{x:.3g}' for x in lon_sticks])
        ax.set_yticklabels([f'{x:.3g}' for x in lat_sticks])
```

```

        ax.tick_params(axis='both', labelsize=6)
    else:
        ax.set_xlabel('[坐标轴过长忽略]')
        ax.set_xticks([])
        ax.set_yticks([])

    im = ax.imshow(data, interpolation='nearest', cmap='viridis')
    self.plot.figure.colorbar(im, format="2^{x:.0f}",
                             label='相对人口密度 (对数)')

    plt.xlim((min_lon, max_lon))
    plt.ylim((min_lat, max_lat))
    self.plot.figure.tight_layout()
    self.plot.draw()

    def run(self):
        logging.info('Loading font ...')
        plt.rcParams['font.sans-serif'] = ['HYZhengYuan'] # 在其他环境中可能需要
修改字体
        plt.rcParams['axes.unicode_minus'] = False
        logging.info('Starting GUI ...')
        self.mainloop()

```

B.3 src/client/requests.py

```

# -*- coding: utf-8 -*-

from projection import Point2D

from typing import Any, Dict, List

import aiohttp
import asyncio
import json


def encapsulate_geojson(points2d: List[Point2D]) -> Dict[str, Any]:
    """
    Encapsulate points in GeoJSON format.

    :param points2d: list of Point3D objects.
    :return: GeoJSON string.

```

```
"""
    return {
        'type': 'Polygon',
        'coordinates': [[(p.x, p.y) for p in points2d]]
    }

async def request(url, data):
    if not url.startswith('http://'):
        url = 'http://' + url
    async with aiohttp.ClientSession() as session:
        async with session.post(url, json=data) as response:
            return await response.json()
```

B.4 src/config/cli.py

```
# -*- coding: utf-8 -*-

"""
Cli arguments parser for the project.
"""

from .project_meta import *
from .config import CONFIG

from argparse import ArgumentParser


class Cli(ArgumentParser):
    def __init__(self):
        super().__init__(
            description=__description__,
            epilog=f'Author: {__author__}',
        )
        self.add_argument(
            '-v', '--version',
            action='version',
            version=f'{__project__} {__version__}',
        )
        self.add_argument(
            '-c', '--client',
            action='store_true',
            help='Run client',
```

```
)  
    self.add_argument(  
        '-s', '--server',  
        action='store_true',  
        help='Run server',  
    )  
    self.add_argument(  
        '-a', '--addr',  
        type=str,  
        help='Server address',  
    )  
    self.add_argument(  
        '-p', '--port',  
        type=int,  
        help='Server port',  
    )  
    self.add_argument(  
        '-l', '--log',  
        type=str,  
        default=CONFIG.DEFAULT_LOG_LEVEL,  
        help='Logging level',  
    )  
    self.add_argument(  
        '--log-file',  
        type=str,  
        default=CONFIG.DEFAULT_LOG_FILE,  
    )  
    self.add_argument(  
        '--preprocess',  
        action='store_true',  
        help='Preprocess data',  
    )  
  
def parse(self):  
    return self.parse_args()
```

B.5 src/config/config.py

```
# -*- coding: utf-8 -*-  
  
"""  
Configurations for the project.  
"""
```

```
from dataclasses import dataclass

@dataclass
class Config:
    INSTANCE_DIR: str = 'instance'
    ASSETS_DIR: str = 'assets'
    DEFAULT_LOG_LEVEL: str = 'INFO'
    DEFAULT_LOG_FILE: str = 'run.log'
    DEFAULT_ADDR: str = '127.0.0.1'
    DEFAULT_PORT: int = 555

    def __getitem__(self, item):
        return getattr(self, item)

    def __setitem__(self, key, value):
        return setattr(self, key, value)

    def __delitem__(self, key):
        return delattr(self, key)

CONFIG = Config()
```

B.6 src/config/project_meta.py

```
# -*- coding: utf-8 -*-

"""
Meta information about the project.
"""

__all__ = [
    '__project__',
    '__version__',
    '__repo__',
    '__author__',
    '__description__',
]

__project__ = "PPPoE"
__version__ = '0.1.0'
```

```
__repo__ = 'https://'  
__author__ = 'fa_555 <fa_555@foxmail.com>'  
__description__ = \  
    'A dual client / server application for tracking world population.'
```

B.7 src/data_manip/data_manip.py

```
# -*- coding: utf-8 -*-  
  
from config import CONFIG  
  
import logging  
import numpy as np  
import os  
  
  
class DataManipulator:  
    DATASET = 'gpw-v4-population-count-rev11_2020_30_sec_asc'  
    FOLDER = os.path.join(CONFIG.ASSETS_DIR, DATASET)  
    FILE_PREFIX = 'gpw_v4_population_count_rev11_2020_30_sec_'  
  
    def __init__(self):  
        self._data_raw = []  
        self._data_processed = None  
  
    def load_raw(self):  
        for i in range(8):  
            file_name = f'{self.FILE_PREFIX}{i + 1}.asc'  
            logging.info(f'Loading file {file_name} ... ')  
            file = os.path.join(self.FOLDER, file_name)  
            data = np.loadtxt(file, skiprows=6, dtype=np.float32)  
            self._data_raw.append(data)  
            logging.info(f'File {file_name} loaded.')  
  
    def process(self):  
        layout = (  
            (0, 0), (0, 1), (0, 2), (0, 3),  
            (1, 0), (1, 1), (1, 2), (1, 3),  
        )  
        logging.info('Concatenating data ... ')  
        self._data_processed = np.zeros((21600, 43200), dtype=np.float32)  
        for i, data in enumerate(self._data_raw):  
            x, y = layout[i]
```

```
x *= 10800
y *= 10800
self._data_processed[x:x + 10800, y:y + 10800] = data

del self._data_raw
import gc
gc.collect()

logging.info('Data concatenated.')

logging.info('Transforming data ... ')
self._data_processed = np.where(
    self._data_processed == -9999, 0, self._data_processed
)
self._data_processed = np.cumsum(self._data_processed, axis=1)

s = 0
for i in range(21600):
    s += self._data_processed[i, -1]
logging.info(f'Sum: {s}')

def save(self):
    logging.info('Saving preprocessed data ... ')
    file = os.path.join(CONFIG.INSTANCE_DIR, 'data.npz')
    np.savez_compressed(file, data=self._data_processed)
    logging.info('Data saved.')

def load(self):
    logging.info('Loading preprocessed data ... ')
    file = os.path.join(CONFIG.INSTANCE_DIR, 'data.npz')
    self._data_processed = np.load(file)['data']
```

B.8 src/projection/projection.py

```
# -*- coding: utf-8 -*-

"""
Projection module.

Using Mollweide projection to project the world map.
"""

import logging
from dataclasses import dataclass
```

```
from math import pi, sin, cos, asin, sqrt

import numpy as np

CONSTANT = 10800
VERTICAL = CONSTANT * 2
HORIZONTAL = CONSTANT * 4

@dataclass
class Point2D:
    """
    Point in 2D space.
    """
    x: int # range: [0, 10800 * 2)
    y: int # range: [0, 10800 * 4)

    def __hash__(self):
        return self.x * HORIZONTAL + self.y

    @classmethod
    def from_portion(cls, x: float, y: float):
        """
        Create a Point2D object from portion.

        :param x: x portion, range: [-2, 2].
        :param y: y portion, range: [-1, 1].
        :return: Point2D object.
        """
        logging.debug(f'Point2D from portion x: {x}, y: {y}.')
        if x < -2 or x > 2:
            raise ValueError('x must be in [-2, 2]')
        if y < -1 or y > 1:
            raise ValueError('y must be in [-1, 1]')

        # y: [-1, 1] → [0, 2] → [0, 10800 * 2)
        # x: [-2, 2] → [0, 4] → [0, 10800 * 4)
        x, y = y, x
        x = int((-x + 1) * CONSTANT)
        y = int((y + 2) * CONSTANT)
        if x == VERTICAL:
            x -= 1
        if y == HORIZONTAL:
            y -= 1
```

```
        return cls(x, y)

    def __sub__(self, other):
        return Point2D(self.x - other.x, self.y - other.y)

@dataclass
class Point3D:
    """
    Point in 3D space. Latitude and longitude are in radians.
    """

    longitude: float # range: [-pi, pi]
    latitude: float # range: [-pi / 2, pi / 2]

    def __str__(self):
        return f'({np.rad2deg(self.longitude):.2f}, '\
               f'{np.rad2deg(self.latitude):.2f})'

    @classmethod
    def from_rad(cls, longitude: float, latitude: float):
        """
        Create a Point3D object from radians.

        :param longitude: Longitude in radians.
        :param latitude: Latitude in radians.
        :return: Point3D object.
        """

        logging.debug(
            f'Point3D from rad longitude: {longitude}, latitude: {latitude}.'
        )
        if longitude < -pi or longitude > pi:
            raise ValueError('longitude must be in [-pi, pi]')
        if latitude < -pi / 2 or latitude > pi / 2:
            raise ValueError('latitude must be in [-pi / 2, pi / 2]')

        return cls(longitude, latitude)

    @classmethod
    def from_deg(cls, longitude: float, latitude: float):
        """
        Create a Point3D object from degrees.

        :param longitude: Longitude in degrees.
        :param latitude: Latitude in degrees.
        """
```

```
:return: Point3D object.  
"""  
  
logging.debug(  
    f'Point3D from deg longitude: {longitude}, latitude: {latitude}.'  
)  
# if longitude < -180 or longitude > 180:  
#     raise ValueError('longitude must be in [-180, 180]')  
# if latitude < -90 or latitude > 90:  
#     raise ValueError('latitude must be in [-90, 90]')  
return cls.from_rad(np.deg2rad(longitude), np.deg2rad(latitude))  
  
  
def project_(point: Point3D) → Point2D:  
    """  
    Project a point on the world map.  
  
    :param point: Point3D object.  
    :return: Point2D object.  
    """  
  
def get_theta_binary(phi: float) → float:  
    """  
    Get theta, where 2theta + sin(2theta) = pi * sin(phi).  
    :param phi:  
    :return: desired theta  
    """  
  
    rhs = pi * sin(phi) # range: [-pi, pi], strictly increasing  
  
    # then 2theta is in range [-pi, pi]  
    lb = -pi  
    rb = pi  
    while rb - lb > 1e-8:  
        mid = (lb + rb) / 2  
        if mid + sin(mid) < rhs:  
            lb = mid  
        else:  
            rb = mid  
  
    # now 2theta approx (lb + rb) / 2  
    return (lb + rb) / 4  
  
def get_theta_newton(phi: float) → float:  
    """  
    Get theta, where 2theta + sin(2theta) = pi * sin(phi).
```

```
:param phi:  
:return: desired theta  
"""  
  
    theta_ = phi  
    for _ in range(10):  
        tmp = 2 * theta_  
        theta_ -= ((tmp + sin(tmp) - pi * sin(phi)) / (2 + 2 * cos(tmp)))  
    return theta_  
  
if abs(point.latitude) == pi / 2:  
    return Point2D.from_portion(0, point.latitude / (pi / 2))  
  
# 2theta + sin(2theta) = pi * sin(latitude)  
theta = get_theta_newton(point.latitude)  
x = 2 / pi * point.longitude * cos(theta)  
y = sin(theta)  
return Point2D.from_portion(x, y)  
  
  
def project(point: Point3D) → Point2D:  
    # map [-pi, pi] * [-pi / 2, pi / 2] to [-2, 2] * [-1, 1] linearly  
    x = point.longitude / pi * 2  
    y = point.latitude / pi * 2  
    return Point2D.from_portion(x, y)  
  
  
def inv_project(point: Point2D) → Point3D:  
    """  
    Un-project a point on the world map.  
  
    :param point: Point2D object.  
    :return: Point3D object.  
    """  
    x = point.y / CONSTANT - 2  
    y = point.x / CONSTANT - 1  
  
    theta = asin(y)  
    lon = pi * x / (2 * cos(theta))  
    lat = asin((2 * theta + sin(2 * theta)) / pi)  
    return Point3D.from_rad(lon, lat)
```

B.9 src/server/algo.py

```
import logging

from data_manip import DataManipulator
from projection import Point2D

from typing import List


class MinMax:
    def __init__(self, arg):
        self.min = arg
        self.max = arg

    def update(self, arg):
        self.min = min(self.min, arg)
        self.max = max(self.max, arg)

    def get(self):
        return self.min, self.max


class MinMaxDict:
    def __init__(self):
        self._dict = {}

    def update(self, key, value):
        if key not in self._dict:
            self._dict[key] = MinMax(value)
        else:
            self._dict[key].update(value)

    def get_dict(self):
        return self._dict

    def get(self, key):
        return self._dict[key].get()


def cross(lhs: Point2D, rhs: Point2D) → float:
    """
    Calculate the cross product of two vectors.

    :param lhs: Vector lhs.
    :param rhs: Vector rhs.
    """
```

```
:return: Cross product of lhs and rhs.  
"""  
    return lhs.x * rhs.y - lhs.y * rhs.x  
  
  
def calc_convex(p: List[Point2D]) → List[Point2D]:  
    """  
        Calculate the convex hull of a list of points.  
  
        :param p: List of points.  
        :return: List of points consisted of the convex hull. In counter-clockwise  
                order.  
    """  
  
    p.sort(key=lambda p: (p.x, p.y))  
    used = [False] * len(p)  
  
    stack = [0]  
    # used[0] = True  
  
    # 求下凸壳  
    for i in range(1, len(p)):  
        while len(stack) > 1 and cross(  
            p[stack[-1]] - p[stack[-2]], p[i] - p[stack[-1]])  
        ) ≤ 0:  
            used[stack.pop()] = False  
        stack.append(i)  
        used[i] = True  
  
    base = len(stack)  
    # 求上凸壳  
    for i in range(len(p) - 2, -1, -1):  
        if used[i]:  
            continue  
        while len(stack) > base and cross(  
            p[stack[-1]] - p[stack[-2]], p[i] - p[stack[-1]])  
        ) ≤ 0:  
            used[stack.pop()] = False  
        stack.append(i)  
        used[i] = True  
  
    return [p[i] for i in stack]  
  
  
def point_in_convex_linear(point: Point2D, convex: List[Point2D]) → bool:
```

```
"""
Check if a point is in a convex.

:param point: Point to check.
:param convex: Convex to check, in counter-clockwise order.
:return: True if the point is in the convex, False otherwise.
"""

for i in range(len(convex) - 1):
    if cross(convex[i] - point, convex[(i + 1) % len(convex)] - point) < 0:
        return False
return True


def point_in_convex(point: Point2D, convex: List[Point2D]) → bool:
"""
Check if a point is in a convex.

:param point: Point to check.
:param convex: Convex to check, in counter-clockwise order.
:return: True if the point is in the convex, False otherwise.
"""

lb = 0
rb = len(convex) - 1
while rb - lb > 1:
    mid = (lb + rb) // 2
    if cross(convex[mid] - convex[0], point - convex[0]) ≥ 0:
        lb = mid
    else:
        rb = mid

return cross(convex[lb] - point, convex[rb] - point) ≥ 0


class DataAccessor(DataManipulator):
    def __init__(self):
        super().__init__()
        self.load()

    def query_all_sum(self):
        return self._data_processed[21509, 43109]

    def query(self, row, starting_col, ending_col):
        if row ≥ 21600:
            return 0
```

```
if ending_col >= 43200:
    ending_col = 43109

    ret = self._data_processed[row, ending_col] - (
        0
        if starting_col == 0 else
        self._data_processed[row, starting_col - 1]
    )
return ret

def calc_whole_convex(convex, data_accessor):
    intersections = MinMaxDict()

    for i in range(0, len(convex) - 1):
        p1 = convex[i]
        p2 = convex[i + 1]
        x1 = min(p1.x, p2.x)
        y1 = min(p1.y, p2.y)
        x2 = max(p1.x, p2.x)
        y2 = max(p1.y, p2.y)
        if y1 == y2:
            for x in range(x1, x2 + 1):
                intersections.update(x, y1)
        elif x1 == x2:
            for y in range(y1, y2 + 1):
                intersections.update(x1, y)
        else:
            k = (y1 - y2) / (x1 - x2)
            b = y1 - k * x1
            for x in range(x1, x2 + 1):
                y = int(k * x + b)
                intersections.update(x, y)

    ans = 0
    for x in intersections.get_dict().keys():
        min_, max_ = intersections.get(x)
        ans += data_accessor.query(x, min_, max_)
    return ans
```

B.10 src/server/server.py

```
from .algo import *
from config.project_meta import __project__

from sanic import Sanic, response
from sanic.worker.loader import AppLoader
from sanic.log import logger
from functools import partial

import asyncio

def create_app():
    app = Sanic(f'{__project__}s')
    app.ctx.data_accessor = DataAccessor()

    @app.get('/ping')
    async def ping_handler(request):
        arg: str = request.args.get('arg', 'ping')
        return response.text(
            arg.replace('i', 'o').replace('I', 'O')
        )

    @app.post('/api/total')
    async def total_handler(request):
        logger.log(logging.INFO, f'/api/total: {request.json}')
        try:
            points = request.json.get('geometry')['coordinates'][0]
            if len(points) < 3:
                raise Exception('Too few points.')

            convex = calc_convex(list(Point2D(p[0], p[1]) for p in points))

            loop = asyncio.get_event_loop()
            ans = await loop.run_in_executor(
                None, calc_whole_convex, convex, app.ctx.data_accessor
            )

            logger.debug(f'convex: {convex}, ans: {ans}')
            return response.json({
                'convex': [[p.x, p.y] for p in convex],
                'population': ans
            })

        except Exception as e:
```

```
logger.fatal(f'/api/total: {e.__repr__()}', exc_info=True)
    return response.json({
        'error': str(e)
    })

@app.post('/api/grid')
async def grid_handler(request):
    logger.log(logging.INFO, f'/api/grid: {request.json}')
    try:
        points = request.json.get('geometry')['coordinates'][0]
        width = request.json.get('grid_width')

        if len(points) < 3:
            raise Exception('Too few points.')
        if width < 1:
            raise Exception('Grid width must be positive.')

        min_x = min(p[0] for p in points)
        max_x = max(p[0] for p in points)
        min_y = min(p[1] for p in points)
        max_y = max(p[1] for p in points)

        convex = calc_convex(list(Point2D(p[0], p[1]) for p in points))

        cache = {}
        ans_mat = []
        for x in range(min_x, max_x + 1, width):
            ans_mat.append([])
            for y in range(min_y, max_y + 1, width):
                square = [
                    Point2D(x, y), Point2D(x + width, y),
                    Point2D(x, y + width), Point2D(x + width, y + width)
                ]

                valid = False
                for p in square:
                    if p not in cache:
                        cache[p] = await (asyncio.get_event_loop()
                                         .run_in_executor(
                                             None, point_in_convex, p, convex))
                if cache[p]:
                    valid = True
                    break
            if valid:
                break
        ans_mat[x][y] = valid
    except Exception as e:
        logger.error(f'Error processing grid request: {e}')
        return response.json({'error': str(e)})
    return response.json({'grid': ans_mat})
```

```
if not valid:
    ans_mat[-1].append(-1)
    continue

    ans = 0
    conv = calc_convex(square)
    loop = asyncio.get_event_loop()
    ans += await loop.run_in_executor(
        None, calc_whole_convex,
        conv, app.ctx.data_accessor
    )
    ans_mat[-1].append(ans)

return response.json({
    'grid': ans_mat,
})

except Exception as e:
    logger.fatal(f'/api/grid: {e.__repr__()}', exc_info=True)
    return response.json({
        'error': str(e)
})

return app

class Server:
    def __init__(self, port: int):
        self.loader = AppLoader(factory=partial(create_app))
        self.app = self.loader.load()
        self.app.prepare(host='0.0.0.0', port=port, dev=True)

    def run(self):
        Sanic.serve(primary=self.app, app_loader=self.loader)
```

B.11 src/util/file.py

```
# -*- coding: utf-8 -*-

import os

def try_mkdir(path: str):
```

```
"""
Try to make a directory.

:param path: Path of the directory.
:return: None
"""

try:
    os.mkdir(path)
except FileExistsError:
    pass


def try_rm(path: str):
    """
    Try to remove a file or directory.

    :param path: Path of the file or directory.
    :return: None
    """

    try:
        if os.path.isfile(path):
            os.remove(path)
        elif os.path.isdir(path):
            os.rmdir(path)
    except FileNotFoundError:
        pass
```

B.12 src/util/log.py

```
# -*- coding: utf-8 -*-

from config import CONFIG

import logging
import colorlog
import os

from typing import Optional


def init_logging(level: str, log_file: str):
    """
    Initialize logging system.

    :param level: Logging level (e.g., 'INFO', 'WARNING')
    :param log_file: Path to the log file
    """
    # Set up logging configuration
    logging.basicConfig(
        filename=log_file,
        level=level,
        format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
        style='-'
```

```
:param level: Logging level.
:param log_file: Logging file.
:return: None
"""

log_file = os.path.join(CONFIG.INSTANCE_DIR, log_file)
numeric_level: Optional[int] = getattr(logging, level.upper(), None)

if not isinstance(numeric_level, int):
    raise ValueError(f'Invalid log level: {numeric_level}')

logging.basicConfig(
    level=numeric_level,
    format='%(asctime)s %(levelname)s %(message)s',
    datefmt='%H:%M:%S',
    handlers=[
        logging.FileHandler(log_file),
        # logging.StreamHandler()
    ]
)

stream_handler = logging.StreamHandler()
stream_handler.setFormatter(colorlog.ColoredFormatter(
    '%(log_color)s%(asctime)s %(levelname)s %(message)s',
    datefmt='%H:%M:%S',
    log_colors={
        'DEBUG': 'cyan',
        'INFO': 'purple',
        'WARNING': 'yellow',
        'ERROR': 'red',
        'CRITICAL': 'red,bg_white',
    }
))
logging.getLogger().addHandler(stream_handler)

logging.info('Logging service initialised.')
```

B.13 src/util/time.py

```
# -*- coding: utf-8 -*-

import time
```

```
def get_cur_time_ms():
    """
    Get current time in milliseconds.

    :return: Current time in milliseconds.
    """
    return int(round(time.time() * 1000))
```

B.14 requirements.txt

```
aiofiles==23.2.1
aiohttp==3.9.1
aiosignal==1.3.1
asyncio==3.4.3
attrs==23.2.0
colorlog==6.8.0
contourpy==1.2.0
cycler==0.12.1
fonttools==4.47.0
frozenlist==1.4.1
html5tagger==1.3.0
http tools==0.6.1
idna==3.6
kiwisolver==1.4.5
matplotlib==3.8.2
multidict==6.0.4
numpy==1.26.3
packaging==23.2
pillow==10.2.0
pyparsing==3.1.1
python-dateutil==2.8.2
sanic==23.12.0
sanic-routing==23.12.0
setuptools==69.0.3
six==1.16.0
tracerite==1.1.1
typing_extensions==4.9.0
ujson==5.9.0
uvloop==0.19.0
websockets==12.0
yarl==1.9.4
```

[点此](#)回到附录以前。