《算法设计与分析》

课程实验报告

专业：计算机科学与技术

班级：2021211306

姓名：杜抒泽

学号：2021211110

第二章 分治：平面最近点对

2021211110 杜抒泽

2023 年 11 月 24 日

目录

1 实验内容

1.1 实验要求

利用 xx 省会城市 TD-LTE 网络的小区/基站数据，针对平面最近点对，验证算法正确性，观察分析算法的时间、空间复杂性变化。

- 基站数据

  参见文件“1033 个基站数据”

  - 基站标识 ENODEBID
  - 基站经度 LONGITUDE
  - 基站纬度 LATITUDE
  - 基站 k-dist 距离 K_DIST
  - 离该基站最近的 k 个基站的平均距离，或者：离该基站第 k 近的基站距离，e.g. $k = 3, 4$
- 采用平面最近点对算法，根据基站经纬度，挑选出
  - 距离非零、且最近的 2 个基站
  - 距离非零、且次最近的 2 个基站
- 说明：某些不同频点基站可能位置相同
- 要求：返回

  1) 最近/次最近的 2 个基站间距离

  2) 最近/次最近的 2 个基站点对（用基站 ENodeBID 表示）

1.2 题目需求分析

本次实验的主要内容是对于球面上的给出的一系列点集（通过经度和纬度表示），求出两两之间球面距离最小的点对和非严格次最小[1][2]的点对（以下简称最近点对和次近点对）。

形式化地，给定 n 个点的点集 $P = \{p_1, p_2, ..., p_n\}$，其中 $p_i = (x_i, y_i)$。求无序对（集合）$\{p, q\}, \{r, s\} \subset \mathcal{P}(P)$ 使得 $\forall (i, j \in \{1, 2, ..., n\} \wedge i \neq j)\big(d(p, q) \leq d\big(p_i, p_j\big)\big)$，$\forall \big(i, j \in \{1, 2, ..., n\} \wedge i \neq j \wedge \{p_i, p_j\} \neq \{p, q\}\big)\big(d(p, q) \leq d(r, s) \leq d\big(p_i, p_j\big)\big)$。其中函数

$$d\big(p_i, p_j\big) = R \arccos\big(\cos y_i \cos y_j \cos\big(x_i - x_j\big) + \sin y_i \sin y_j\big)$$

[1]“非严格”指可以等于最小距离。严格和非严格仅当存在多个点对同时距离相同且最小时有区别。求非严格答案简单许多。

[2]本题给出的数据很弱，求出的非严格次近点对的答案同时也是严格次近点对。

其中 R 为系数（地球半径）。注意公式中 x_i 和 y_i 采用弧度制，而给出的数据采用角度制，需要进行转换。

显然，直接对于球面坐标（经度和纬度）运行平面最近点对算法有系统性错误，因此将对于球面坐标投影到平面再求解。然而，将球面投影到平面上必然引入误差，因此我选择在多种投影算法下分别求解，并与不需要投影的朴素算法进行比较，以验证算法的正确性。

将球面投影到平面后，我们即可回到计算熟悉的欧几里得距离，直接使用平面最近点对算法求解。

容易发现，题给数据中只有编号、经度、纬度三列有用。

1.3 具体实验内容

具体地，我在本次实验中实现了以下内容：

- 编写数据预处理脚本，从给出的 .xls 文件中提取出基站经纬度数据并写入文本文件供后续使用。
- 采用正弦投影、墨卡托投影、米勒圆柱投影三种投影算法，将经纬度近似投影到平面上。其中正弦投影是伪圆柱投影，其余两者是圆柱投影。[3]
- 正确处理可能存在的多个点位置相同的情况。[4]
- 实现 $\mathcal{O}(n \log^2 n)$ 的平面最近点对算法，并对其进行封装，利用其求解各个投影算法下的最近和非严格次近点对。
- 编写 $\mathcal{O}(n^2)$ 的、不需投影的朴素算法，用于验证平面最近点对算法的正确性。
- 对每次算法的运行时间进行统计，分析算法的时间复杂度是否符合预期。
- 编写构建与运行脚本，提供易于使用的接口用于运行程序的各个部分。脚本在类 Unix 环境下能够工作，**但没有在 Windows 环境下测试，不保证能够运行。**若有需要请联系我或自行适配。

2 算法设计

2.1 $\mathcal{O}(n \log^2 n)$ 平面最近点对算法

基于分治思想的 $\mathcal{O}(n \log^2 n)$ 平面最近点对算法是相当经典的分治算法。其基本思想如下：

[3] 对于一个高中没学过地理的人来说搞懂这些真的老难了。
[4] 虽然最后完全没用上。

将所有点置于一个序列中。初始时按照 x 坐标为关键字升序排序（具体实现时还使用了 y 为第二关键字，不影响正确性）。设序列范围为 $[l,r)$，则取其中点 m，将序列分为 $[l,m)$ 和 $[m,r)$ 两部分。递归地求解两部分的最近点对，统计两部分的最近点对中距离最小的点对 (p,q) 以及它们的距离 d。

递归后需要合并。我们不仅需要合并两部分内部的最近点对，还需要考虑跨越两部分的最近点对。我们考虑围绕分隔左右两部分的分界线（存在这样一条分界线是由于我们已经按照 x 排序）统计跨越两部分的点对的答案。对于这条分界线，将所有距离它的距离不超过 d 的点加入临时点集 T。这样做的理由很明显：如果一个点离分界线的距离都已经超过了 d，那么它与分界线另一边的点的距离必然不会小于 d。

求出点集 T 后，将其所有点按照 y 坐标升序排序。接着我们枚举 T 中的所有点。对于每个点 p，找到所有与它的纵坐标相差不超过 d 的点 q_i，计算 $d(p,q_i)$ 并更新答案。这样做的正确性同样是显然的。

但是它的复杂度为什么正确呢？枚举点集 T 内的所有点 p 和对于每个 p 的所有 q_i 不是 $O\left(n^2\right)$ 的吗？

还真不是。我们考虑对于每个点 p，它的 q_i 的数量。对于每一个点 p，考虑可能成为点 q_i 的所有点可能处于的位置：

- q_i 与分界线的距离小于 d
- q_i 与 p 的纵坐标相差不超过 d [5]

基于这两个条件，我们可以画出一个大小至多为 $2d\times 2d$ 的矩形（不含边界），q_i 一定在其中。我们将其分为 4 个 $d\times d$（不含外侧边界）的部分，容易发现每个部分中至多只能有 3 个点，它们的距离两两大于等于 d。[6] 因此，q_i 的个数是 $\mathcal{O}(1)$ 的，对于点集 T 枚举过程是 $\mathcal{O}(n)$ 而非 $\mathcal{O}\left(n^2\right)$ 的。对点集 T 的时间瓶颈在于排序，需要 $\mathcal{O}(n\log n)$ 时间。

接下来的复杂度分析可以使用 Master 定理。我们有

$$T(n)=2T\left(\frac{n}{2}\right)+\mathcal{O}(n\log n)=\mathcal{O}\left(n\log^2 n\right)$$

如果在分治求平面最近点对的同时归并排序，则时间复杂度降低到 $\mathcal{O}(n\log n)$，但实现直接采用了 C++ 标准库的 `std::sort`，因此时间复杂度为 $\mathcal{O}\left(n\log^2 n\right)$。此外，算法使用 $\mathcal{O}(n)$ 的额外空间，用于存储临时点集 T。

算法实现如下：

[5]还有一条显然的性质：q_i 和 p 不在分界线的同侧。不过这条性质在实现中并没有用。
[6]乍一看很容易发现，但严格证明起来可能没那么显然？考虑使用反证法证明。

```cpp
struct InnerAlgoRes {
  Point nearest1, nearest2;
  double min_dist;
};

using Iter = std::vector<Point>::iterator;

InnerAlgoRes _find_nearest(Iter begin, Iter end) {
  Iter::difference_type dis = std::distance(begin, end);
  if (dis == 1)
    return {*begin, *begin, INF};
  else if (dis == 2)
    return {*begin, *std::next(begin), dist(*begin, *std::next(begin))};

  Iter mid = midpoint(begin, end);
  auto res1 = _find_nearest(begin, mid);
  auto res2 = _find_nearest(mid, end);
  if (res1.min_dist > res2.min_dist)
    std::swap(res1, res2);

  auto [p1, p2, min_dist] = res1;
  std::vector<Point> t;

  mid = std::prev(mid);
  for (Iter it = begin; it ≠ end; ++it)
    if (std::abs(it→x - mid→x) ⩽ min_dist)
      t.emplace_back(*it);
  std::sort(t.begin(), t.end(), [](const Point &lhs, const Point &rhs) {
    return lhs.y < rhs.y;
  });

  for (Iter it = t.begin(); it ≠ t.end(); ++it)
    for (Iter jt = std::next(it); jt ≠ t.end() && jt→y - it→y < min_dist;
         ++jt) {
      if (double d = dist(*it, *jt); d < min_dist) {
        min_dist = d;
        p1 = *it;
        p2 = *jt;
      }
    }

  return {p1, p2, min_dist};
}
```

其中 `InnerAlgoRes` 类型用于传递算法结果（最近点对的两点与它们之间的距离）。`_find_nearest` 函数接收一个迭代器范围，返回最近点对与它们的距离。

2.2 用最近点对算法求非严格次近点对

既然我们已经有了平面最近点对算法，那么用它求出非严格次近点对就很简单了。假设我们已经求出了最近点对 (p,q)，考虑以下的过程：

1. 从原序列中删除 p 和 q。
2. 遍历新序列中的所有点 r，用 $d(p,r)$ 和 $d(q,r)$ 更新答案。这个过程是 $\mathcal{O}(n)$ 的。
3. 在新序列上运行平面最近点对算法，求出次近点对 (r,s)，用 $d(r,s)$ 更新答案。这个过程是 $\mathcal{O}\left(n\log^2 n\right)$ 的。

简单思考即可得到这个算法的正确性：它会统计原序列中任意两点，除了 (p,q) 之间的答案。而它的时间复杂度和平面最近点对算法是成线性的（只有常系数有区别），因此这个算法的时间复杂度是 $\mathcal{O}\left(n\log^2 n\right)$ 的。此外，这个算法只需要 $\mathcal{O}(1)$ 的额外空间。

```
struct AlgoRes {
  Point nearest1, nearest2;
  Point second_nearest1, second_nearest2;
};

AlgoRes find_nearest(std::vector<Point> &&points) {
  auto [nearest1, nearest2, min_dist] =
      _find_nearest(points.begin(), points.end());

  Point second_nearest1 = nearest1, second_nearest2 = nearest1;
  double second_min_dist = INF;

  for (Iter it = points.begin(); it != points.end(); ++it) {
    if (*it == nearest1 || *it == nearest2)
      continue;
    if (dist(*it, nearest1) < second_min_dist) {
      second_nearest1 = nearest1;
      second_nearest2 = *it;
      second_min_dist = dist(*it, nearest1);
    }
    if (dist(*it, nearest2) < second_min_dist) {
      second_nearest1 = nearest2;
      second_nearest2 = *it;
      second_min_dist = dist(*it, nearest2);
    }
  }

  std::erase(points, nearest1);
  std::erase(points, nearest2);
  auto [nearest3, nearest4, min_dist2] =
      _find_nearest(points.begin(), points.end());
```

```
  if (min_dist2 < second_min_dist) {
    second_nearest1 = nearest3;
    second_nearest2 = nearest4;
    second_min_dist = min_dist2;
  }

  return {nearest1, nearest2, second_nearest1, second_nearest2};
}
```

用这个算法拓展到求严格次近点对是不可行的。事实上我想不出一个低于朴素算法的求严格次近点对的算法。

2.3 求最近与次近点对的朴素算法

朴素算法没什么门道，只是对于 $\mathcal{O}(n^2)$ 个点对暴力枚举一遍统计答案：

- 每枚举一个点对，计算它们之间的距离，和当前维护的最小距离比较，若更小则将次小答案更新为当前的最小答案，将最小答案更新为当前点对。
- 否则与当前维护的次小距离距离比较，若更小则将次小答案更新为当前点对。
- 否则不更新答案。

时间复杂度是 $\mathcal{O}(n^2)$ 的，需要 $\mathcal{O}(1)$ 的额外空间。

要将这个算法改为求严格次近点对很简单，只需增加一趟遍历。第一次遍历求出距离最小的点对并记录这个距离，第二次遍历时将所有距离最小的点对排除在外即可。时间复杂度没有变化。

2.4 球面到平面的三种投影算法

将球面投影到平面是老大难问题，对于求最近点对算法尤甚。一方面投影算法多到令人眼花缭乱，另一方面这些算法或多或少都会失真，而球面距离与投影后的平面距离是无法一一对应的（这相当于找到球体的平面展开图，不可能实现）。

然而，由于题目给出的数据的经纬度跨度并不大，我选取的三种投影算法都正确地求得了最近点对和次近点对。这三种算法分别是正弦投影、墨卡托投影和米勒圆柱投影。

以下讨论中用 λ 表示经度，φ 表示纬度（均用弧度表示），，x 表示投影后的 x 坐标，y 表示投影后的 y 坐标。每种投影算法实际上是一个函数 $f:((-\pi,\pi]\times\left[-\frac{\pi}{2},\frac{\pi}{2}\right]\to\mathbb{R}^2$，也即 $(x_i,y_i)=f(\lambda_i,\varphi_i)$。

容易发现，我们其实不在意 x 和 y 的具体数值，只要 (x_i, y_i) 和 (x_j, y_j) 之间的欧几里得距离与原始 (λ_i, φ_i) 和 (λ_j, φ_j) 之间的球面距离保持相同的偏序关系[7]即可。因此我们在求最近点对时直接假定球体半径为 1，在答案统计完成之后再回到原始经纬度上计算实际的球面距离。

2.4.1 正弦投影

正弦投影[8]是伪圆柱投影，其特点为对赤道和某一条指定经线保真，对面积保真。然而其他经线的投影会发生严重拉伸。正弦投影由以下公式给出：

$$\begin{cases} x = (\lambda - \lambda_0)\cos\varphi \\ y = \varphi \end{cases}$$

其中 λ_0 是选定的保真经线。本实验的实验要求中给出了经度范围 $(102.673290, 102.895277)$，因此我们直接求其中点作为保真经线。代码实现如下：

```
inline Point sinusoidal(const Entry &e) {
  constexpr const static double CENTRAL_MERIDIAN =
      std::midpoint(102.673290, 102.895277); // 在实验要求 ppt 中给出

  double rad_lat = deg2rad(e.latitude);
  return {deg2rad(e.longitude - CENTRAL_MERIDIAN) * cos(rad_lat), rad_lat};
}
```

2.4.2 墨卡托投影

墨卡托投影[9]是一种极常见的圆柱投影，其特点为对角度保真，对面积不保真。墨卡托投影常用来绘制矩形的世界地图，地图的任一点在各种方向的长度均相等，可显示任两点间的正确方位，指出真实的经纬度，线型比例尺在图中任意一点周围都保持不变。然而墨卡托投影会使面积产生变形，赤道地区变化最小，南北两极的变形最大。墨卡托投影由以下公式给出：

[7]如果一个点对的球面距离大于另一个点对，那么需要保证投影后前者的欧几里得距离也大于后者。
[8]正弦投影参见 https://en.wikipedia.org/wiki/Sinusoidal_projection。
[9]墨卡托投影参见 https://en.wikipedia.org/wiki/Mercator_projection。

$$
\begin{aligned}
x &= \lambda \\
y &= \ln \tan\left(\frac{\pi}{4} + \frac{\varphi}{2}\right) \quad \text{[chosen in implementation]} \\
&= \frac{1}{2} \ln\left(\frac{1+\sin\varphi}{1-\sin\varphi}\right) \\
&= \tanh^{-1} \sin\varphi \\
&= \sinh^{-1} \tan\varphi \\
&= \ln(\tan\varphi + \sec\varphi)
\end{aligned}
$$

实现如下：

```
inline Point mercator(const Entry &e) {
  return {
      deg2rad(e.longitude),
      log(tan(deg2rad(e.latitude) / 2 + PI / 4)),
  };
}
```

2.4.3 米勒圆柱投影

米勒圆柱投影[10]是对墨卡托投影的改进，其先将纬度缩放 $\frac{4}{5}$ 倍，将结果投影到平面上后缩放 $\frac{5}{4}$ 倍，保证对赤道保真。米勒圆柱投影有效地避免了墨卡托投影高纬度失真的问题。米勒圆柱投影由以下公式给出：

$$
\begin{aligned}
x &= \lambda \\
y &= \frac{5}{4} \ln\left(\tan\left(\frac{\pi}{4} + \frac{2\varphi}{5}\right)\right) \text{[chosen in implementation]} \\
&= \frac{5}{4} \sinh^{-1} \tan\frac{4\varphi}{5}
\end{aligned}
$$

```
inline Point miller(const Entry &e) {
  return {
      deg2rad(e.longitude),
      1.25 * log(tan(PI / 4 + 0.4 * deg2rad(e.latitude))),
  };
}
```

3 程序设计说明

[10]米勒圆柱投影参见 https://en.wikipedia.org/wiki/Miller_cylindrical_projection。

项目名称为 PNP（Planar Nearest Points）。程序在类 Unix 系统下开发与调试，Python 代码用于数据预处理和控制程序的编译、运行，在 Python 3.10.6 版本下编写；C++ 代码用于编写算法，遵守 C++20 语言标准，确保可移植性。

3.1 如何运行

我提供了构建/运行脚本 `run.py`，包含安装依赖、数据预处理、编译、运行功能，帮助信息如下图：

```
$ ./run.py -h
usage: run.py [-h] [-c] [-p] [-r] [--clean]

options:
  -h, --help        show this help message and exit
  -c, --compile     compile the C++ code
  -p, --preprocess  preprocess the data
  -r, --run         run the program
  --clean           clean the target dir
```

图 1　构建脚本帮助信息

其会从环境变量中读取 `CXX`，`CXXFLAGS` 和 `LDFLAGS` 的值，分别作为 C++ 编译器、编译选项和链接选项。如果环境变量中不存在这些变量，则使用默认值 `CXX=g++`，`CXXFLAGS="-W -Wall -Wextra -Wpedantic -O3 -std=c++20 -Isrc"` 和 `LDFLAGS=""`。

以下是构建脚本的使用示例：

```
chmod +x run.py

./run.py -p # 预处理数据
./run.py -r # 运行程序（若不存在则先编译）
./run.py --clean # 清理编译产物

./run.py -pcr # 预处理数据、编译、运行程序全套流程

CXX=g++-13 ./run.py -pcr # 我的机器上的编译器名字是 g++-13，指定环境变量
```

这个构建脚本是真的帅，我很喜欢它。其实我本来是在写 Makefile 的，但是实在难写。写烦了换用 Python，Python 太可爱了。

3.2 重复基站的处理

前文已经提到，题给数据中只有编号、经度、纬度三列有用。被抽象为 `Entry` 类型：

```
struct Entry {
  int enodebid;
```

```
  double longitude, latitude;
};
```

题目描述中提到，有可能存在两个 Entry，它们的 longitude 和 latitude 相同，只有 enodebid 不同。这种情况需要被特别处理（否则最短距离是 0）。处理数据采取如下的方式：

利用一个 std::map<std::pair<double, double>, std::vector<int>> coord2id，将经纬度映射到 enodebid 的集合。这样，我们将所有经纬度相同的点进行聚合，可以根据这个经纬度查询到所有的 enodebid。然后我们将原序列进行去重处理：

```
std::sort(entries.begin(), entries.end()); // std::unique 需要序列有序
entries.erase(std::unique(entries.begin(), entries.end()), entries.end());
```

最终，只需要拿到一个点对应的 Entry 条目即可查询到所有的 enodebid。

3.3 数据与算法接口的抽象

第二层抽象是从 Entry 到 Point 的抽象。Point 是平面点的抽象，可以用来传递给平面最近点对算法。Point 的定义如下：

```
struct Point {
  double x, y;
};
```

投影函数的签名均为 Point (*)(const Entry &)，而平面最近点对算法接收 std::vector<Point>（由于需要修改其值，不传递带 const 限定的左值引用或右值引用）。通过投影函数将 std::vector<Entry> 投影为 std::vector<Point> 后即可拷贝到平面最近点对算法中。

我们需要确定算法返回的点在原数组中的下标，以此确认其在原 Entry 条目中的经纬度。

起初，由于最近点对算法会修改传入的参数的值，我向其中传递了一个右值引用，并维护了一个 std::map<Point, int> point2id。随后我发现，既然无论如何都要花费 $\mathcal{O}(n)$ 的额外空间，而维护红黑树还需要花费 $\mathcal{O}(n \log n)$ 的时间，那我还不如直接让算法拷贝一份，然后 $\mathcal{O}(n)$ 地动态查询算法返回的点对应的下标。

4 测试、运行结果与分析说明

拿到代码直接运行运行全套流程 ./run.py -pcr，得到如下输出：

```
$ CXX=g++-13 ./run.py -pcr
xlrd not installed, installing ...
Executing: `/Library/Frameworks/Python.framework/Versions/3.10/bin/python3 -m pip install xlrd~=2.0.1` Collecting xlrd~=2.0.1
  Downloading xlrd-2.0.1-py2.py3-none-any.whl (96 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 96.5/96.5 kB 744.5 kB/s eta 0:00:00
Installing collected packages: xlrd
Successfully installed xlrd-2.0.1
Done.
Executing: `python3 ./src/preprocess.py` Done.
Executing: `rm -rf ./target` Done.
Env var CXXFLAGS not set, default to `-W -Wall -Wextra -Wpedantic -O3 -std=c++20 -Isrc`
Env var LDFLAGS not set, default to ``
Executing: `g++-13 -W -Wall -Wextra -Wpedantic -O3 -std=c++20 -Isrc -c ./src/main.cpp -o ./target/main.cpp.o ` Done.
Executing: `g++-13 -W -Wall -Wextra -Wpedantic -O3 -std=c++20 -Isrc -c ./src/reader.cpp -o ./target/reader.cpp.o ` Done.
Executing: `g++-13 -W -Wall -Wextra -Wpedantic -O3 -std=c++20 -Isrc -c ./src/algorithm.cpp -o ./target/algorithm.cpp.o ` Done.
Executing: `g++-13 -W -Wall -Wextra -Wpedantic -O3 -std=c++20 -Isrc ./target/main.cpp.o ./target/reader.cpp.o ./target/algorit
hm.cpp.o -o ./target/main ` Done.
Executing: `./target/main ./data/stations.txt` Duplicate entries detected: 568471 568849

| Brute force:
|          Nearest:  (102.741000, 25.053888) [567389] ⟷ (102.741000, 25.053940) [566803]  (5.788964829353286 m)
|   Second Nearest:  (102.791000, 25.039722) [567222] ⟷ (102.791000, 25.039790) [566784]  (7.569994141266727 m)
| Time elapsed: 13707 µs

| Sinusoidal projection:
|          Nearest:  (102.741000, 25.053888) [567389] ⟷ (102.741000, 25.053940) [566803]  (5.788964829353286 m)
|   Second Nearest:  (102.791000, 25.039722) [567222] ⟷ (102.791000, 25.039790) [566784]  (7.569994141266727 m)
| Time elapsed: 133 µs

| Mercator projection:
|          Nearest:  (102.741000, 25.053888) [567389] ⟷ (102.741000, 25.053940) [566803]  (5.788964829353286 m)
|   Second Nearest:  (102.791000, 25.039722) [567222] ⟷ (102.791000, 25.039790) [566784]  (7.569994141266727 m)
| Time elapsed: 113 µs

| Miller projection:
|          Nearest:  (102.741000, 25.053888) [567389] ⟷ (102.741000, 25.053940) [566803]  (5.788964829353286 m)
|   Second Nearest:  (102.791000, 25.039722) [567222] ⟷ (102.791000, 25.039790) [566784]  (7.569994141266727 m)
| Time elapsed: 108 µs

Done.
```

图 2　运行结果

图片中包含大量信息。

首先是安装第三方库的过程、数据预处理过程和编译过程。编译器 `CXX` 采用了我指定的编译器 `g++-13`，而 `CXXFLAGS` 和 `LDFLAGS` 采用了默认值，开启了大量警告。编译成功，没有输出任何警告信息。预处理的数据被输出到 `./data/station.txt`，被编译得到的算法程序读取。

算法程序输出共五部分。第一部分是不起眼的 `Duplicate entries detected: 568471 568849`，表示发现了坐标相同的基站 568471 和 568849。

第二至五部分是不同算法的输出。第二部分是 $\mathcal{O}(n^2)$ 朴素算法，后三部分是对于不同的投影算法运行平面最近点对算法。可以看到，四部分求得的答案完全相同：

- 最近点对是基站 567389（经纬度为 102.741000 °E, 25.053940 °N）和基站 566803（经纬度为 102.741000 °E, 25.053888 °N），球面距离约为 5.79 米。
- 次近点对是基站 567222（经纬度为 102.791000 °E, 25.039722 °N）和基站 566784（经纬度为 102.791000 °E, 25.039790 °N），球面距离约为 7.57 米。

$\mathcal{O}(n^2)$ 朴素算法的耗时为 13373 μs，而 $\mathcal{O}(n\log^2 n)$ 分治平面最近点对算法的耗时为对于三种不同投影算法的耗时分别为 161 μs, 121 μs 和 115 μs。后三者的耗时可以看作近似相同，毕竟对于能在数百微秒内完成[11]的算法，这样的差距可以看作由于 cache 问题引起的微扰。多次运行，耗时基本稳定。简单计算发现结果符合预期，甚至 $\mathcal{O}(n\log^2 n)$ 带了出乎意料地小的常系数。

5 总结

本次实验到此完全结束。我在本次实验中实现了平面最近点对算法，采用了三种不同的投影算法外加朴素算法，验证了算法的正确性，并对算法的耗时进行了分析。在这次实验中，我将分治算法在解决实际问题的背景中进行了应用，加深了对分治算法的理解，还学习到了如何将抽象的算法概念应用于具体的数据处理。实验仍有以下改进方向：

- 优化算法，在分治的同时进行归并排序，将时间复杂度将至 $\mathcal{O}(n\log n)$。
- 在更大的数据量下进行测试，验证算法面对大数据集（点数更多，经纬度范围更大）时的表现（如时间复杂度和投影的正确性）。同时尝试更多投影算法，选择最合适的投影算法。
- 尝试使用更多算法解决，如空间最近点对（简单查阅资料知似乎也能在 $\mathcal{O}(n\log n)$ 的时间内解决）。而给定切面，三维距离和球面距离可以一一对应。
- 修改数据预处理脚本，使之输出到更标准的数据格式，如 .csv 格式。
- 增加跨平台支持，使之能够在 Windows 环境下运行。

[11]相当于每秒能完整运行近万次。

6 附录 1 - 附件清单

```
PNP
├── run.py
└── src
    ├── algorithm.cpp
    ├── algorithm.h
    ├── data_entry.h
    ├── main.cpp
    ├── preprocess.py
    ├── reader.cpp
    └── reader.h
```

7 附录 2 - 源代码

7.1 src/main.cpp

```cpp
#include "algorithm.h"
#include "data_entry.h"
#include "reader.h"

#include <algorithm>
#include <chrono>
#include <format>
#include <iostream>
#include <map>
#include <numeric>
#include <vector>

std::string to_string(
    const Entry &e,
    const std::map<std::pair<double, double>, std::vector<int>> &coord2id
) {
  // assert e{lat, lon} to be exist in coord2id, or let it crash
  std::string ret;
  ret.reserve(32);
  ret += std::format("({:.6f}, {:.6f}) [", e.longitude, e.latitude);
  for (int id : coord2id.find({e.longitude, e.latitude})->second)
    ret += std::format("{} ", id);
  ret.pop_back();
  ret += ']';
  return ret;
}
```

```
int main(int argc, char **argv) {
  std::vector<Entry> entries = read_data(argc, argv);

  std::map<std::pair<double, double>, std::vector<int>> coord2id;
  for (const Entry &e : entries)
    coord2id[{e.longitude, e.latitude}].emplace_back(e.enodebid);
  for (const auto &[p, v] : coord2id) {
    if (v.size() > 1) {
      std::cout << "Duplicate entries detected: ";
      for (int e : v)
        std::cout << e << ' ';
      std::cout << std::endl;
    }
  }
  std::cout << std::endl;

  std::sort(entries.begin(), entries.end());
  entries.erase(std::unique(entries.begin(), entries.end()), entries.end());

  find_nearest_brute_force(entries);

  for (const auto &projection : {sinusoidal, mercator, miller}) {
    std::chrono::high_resolution_clock::time_point start =
        std::chrono::high_resolution_clock::now();

    std::vector<Point> points;
    auto point_to_index = [&points](const Point &p) {
      return std::distance(
          points.begin(), std::find(points.begin(), points.end(), p)
      );
    };

    for (const Entry &e : entries)
      points.emplace_back(projection(e));

    AlgoRes ret = find_nearest(points);
    Entry e1 = entries[point_to_index(ret.nearest1)];
    Entry e2 = entries[point_to_index(ret.nearest2)];
    Entry e3 = entries[point_to_index(ret.second_nearest1)];
    Entry e4 = entries[point_to_index(ret.second_nearest2)];

    std::chrono::high_resolution_clock::time_point end =
        std::chrono::high_resolution_clock::now();
```

```
    if (projection == sinusoidal)
      std::cout << "| Sinusoidal projection:\n";
    else if (projection == mercator)
      std::cout << "| Mercator projection:\n";
    else if (projection == miller)
      std::cout << "| Miller projection:\n";

    std::cout << std::format(
        "|          Nearest:  {} ⟷ {}  ({} m)\n|   Second Nearest:  {} ⟷ "
        "{}  ({} m)\n| Time elapsed: {} μs\n",
        to_string(e1, coord2id), to_string(e2, coord2id), real_dist(e1, e2),
        to_string(e3, coord2id), to_string(e4, coord2id), real_dist(e3, e4),
        std::chrono::duration_cast<std::chrono::microseconds>(end - start)
            .count()
    );
    std::cout << std::endl;
  }
}
```

7.2 src/algorithm.h

```
#pragma once

#ifndef ALGORITHM_H
#  define ALGORITHM_H

#  include "data_entry.h"

#  include <vector>

template<typename I>
  requires std::random_access_iterator<I>
inline I midpoint(I l, I r) {
  return l + std::distance(l, r) / 2;
}

struct AlgoRes {
  Point nearest1, nearest2;
  Point second_nearest1, second_nearest2;
};

struct InnerAlgoRes {
```

```
  Point nearest1, nearest2;
  double min_dist;
};

void find_nearest_brute_force(const std::vector<Entry> &entries);

AlgoRes find_nearest(std::vector<Point> points);

#endif // !ALGORITHM_H
```

7.3 src/algorithm.cpp

```
#include "algorithm.h"

#include "data_entry.h"

#include <algorithm>
#include <chrono>
#include <format>
#include <iostream>
#include <numeric>
#include <vector>

constexpr const double INF = 1e10;

void find_nearest_brute_force(const std::vector<Entry> &entries) {
  std::chrono::high_resolution_clock::time_point start =
      std::chrono::high_resolution_clock::now();

  double min_dist = INF, second_min_dist = INF;
  Entry nearest1{}, nearest2{}, second_nearest1{}, second_nearest2{};
  for (std::vector<Entry>::size_type i = 0; i < entries.size(); ++i)
    for (std::vector<Entry>::size_type j = i + 1; j < entries.size(); ++j) {
      double d = real_dist(entries[i], entries[j]);
      if (d < min_dist) {
        second_min_dist = min_dist;
        second_nearest1 = nearest1;
        second_nearest2 = nearest2;
        min_dist = d;
        nearest1 = entries[i];
        nearest2 = entries[j];
      } else if (d < second_min_dist) {
        second_min_dist = d;
```

```
        second_nearest1 = entries[i];
        second_nearest2 = entries[j];
      }
    }

  std::chrono::high_resolution_clock::time_point end =
      std::chrono::high_resolution_clock::now();
  std::cout << std::format(
      "| Brute force:\n|          Nearest:  ({:.6f}, {:.6f}) [{}] ⟷ ({:.6f},
"
      "{:.6f}) [{}]  ({} m)\n|   Second Nearest:  ({:.6f}, {:.6f}) [{}] ⟷ "
      "({:.6f}, {:.6f}) [{}]  ({} m)\n| Time elapsed: {} μs\n",
      nearest1.longitude, nearest1.latitude, nearest1.enodebid,
      nearest2.longitude, nearest2.latitude, nearest2.enodebid,
      real_dist(nearest1, nearest2), second_nearest1.longitude,
      second_nearest1.latitude, second_nearest1.enodebid,
      second_nearest2.longitude, second_nearest2.latitude,
      second_nearest2.enodebid, real_dist(second_nearest1, second_nearest2),
     std::chrono::duration_cast<std::chrono::microseconds>(end - start).count()
  );
  std::cout << std::endl;
}

using Iter = std::vector<Point>::iterator;

InnerAlgoRes _find_nearest(Iter begin, Iter end) {
  Iter::difference_type dis = std::distance(begin, end);
  if (dis == 1)
    return {*begin, *begin, INF};
  else if (dis == 2)
    return {*begin, *std::next(begin), dist(*begin, *std::next(begin))};

  Iter mid = midpoint(begin, end);
  auto res1 = _find_nearest(begin, mid);
  auto res2 = _find_nearest(mid, end);
  if (res1.min_dist > res2.min_dist)
    std::swap(res1, res2);

  auto [p1, p2, min_dist] = res1;
  std::vector<Point> t;

  mid = std::prev(mid);
  for (Iter it = begin; it ≠ end; ++it)
    if (std::abs(it→x - mid→x) ⩽ min_dist)
```

```
      t.emplace_back(*it);
  std::sort(t.begin(), t.end(), [](const Point &lhs, const Point &rhs) {
    return lhs.y < rhs.y;
  });

  for (Iter it = t.begin(); it != t.end(); ++it)
    for (Iter jt = std::next(it); jt != t.end() && jt->y - it->y < min_dist;
         ++jt) {
      if (double d = dist(*it, *jt); d < min_dist) {
        min_dist = d;
        p1 = *it;
        p2 = *jt;
      }
    }

  return {p1, p2, min_dist};
}

AlgoRes find_nearest(std::vector<Point> points) {
  auto [nearest1, nearest2, min_dist] =
      _find_nearest(points.begin(), points.end());

  Point second_nearest1 = nearest1, second_nearest2 = nearest1;
  double second_min_dist = INF;

  std::erase(points, nearest1);
  std::erase(points, nearest2);

  for (Iter it = points.begin(); it != points.end(); ++it) {
    if (dist(*it, nearest1) < second_min_dist) {
      second_nearest1 = nearest1;
      second_nearest2 = *it;
      second_min_dist = dist(*it, nearest1);
    }
    if (dist(*it, nearest2) < second_min_dist) {
      second_nearest1 = nearest2;
      second_nearest2 = *it;
      second_min_dist = dist(*it, nearest2);
    }
  }

  auto [nearest3, nearest4, min_dist2] =
      _find_nearest(points.begin(), points.end());
  if (min_dist2 < second_min_dist) {
```

```
    second_nearest1 = nearest3;
    second_nearest2 = nearest4;
    second_min_dist = min_dist2;
  }

  return {nearest1, nearest2, second_nearest1, second_nearest2};
}
```

7.4 src/data_entry.h

```
#pragma once

#ifndef DATA_H
#  define DATA_H

#  include <cmath>
#  include <numbers>
#  include <numeric>
#  include <tuple>

constexpr const static double PI = std::numbers::pi;
constexpr const static double EARTH_RADIUS = 6378137.; // m

struct Point {
  double x, y;

  bool operator<(const Point &rhs) const {
    return std::tie(x, y) < std::tie(rhs.x, rhs.y);
  }

  bool operator==(const Point &rhs) const {
    return std::tie(x, y) == std::tie(rhs.x, rhs.y);
  }
};

inline double dist(const Point &p1, const Point &p2) {
  return std::hypot(p1.x - p2.x, p1.y - p2.y);
}

inline double deg2rad(double deg) {
  return deg * PI / 180;
}
```

```cpp
struct Entry {
  int enodebid;
  double longitude, latitude;

  bool operator<(const Entry &rhs) const {
    return std::tie(latitude, longitude) <
           std::tie(rhs.latitude, rhs.longitude);
  }

  bool operator==(const Entry &rhs) const {
    return std::tie(latitude, longitude) ==
           std::tie(rhs.latitude, rhs.longitude);
  }
};

inline double real_dist(const Entry &e1, const Entry &e2) {
  double rad_lon1 = deg2rad(e1.longitude);
  double rad_lon2 = deg2rad(e2.longitude);
  double rad_lat1 = deg2rad(e1.latitude);
  double rad_lat2 = deg2rad(e2.latitude);
  double s = acos(
      cos(rad_lat1) * cos(rad_lat2) * cos(rad_lon1 - rad_lon2) +
      sin(rad_lat1) * sin(rad_lat2)
  );
  s *= EARTH_RADIUS;
  return s;
}

inline Point sinusoidal(const Entry &e) {
  constexpr const static double CENTRAL_MERIDIAN =
      std::midpoint(102.673290, 102.895277); // 在实验要求 ppt 中给出

  double rad_lat = deg2rad(e.latitude);
  return {deg2rad(e.longitude - CENTRAL_MERIDIAN) * cos(rad_lat), rad_lat};
}

inline Point mercator(const Entry &e) {
  return {
      deg2rad(e.longitude),
      log(tan(deg2rad(e.latitude) / 2 + PI / 4)),
  };
}

inline Point miller(const Entry &e) {
```

```
  return {
      deg2rad(e.longitude),
      1.25 * log(tan(PI / 4 + 0.4 * deg2rad(e.latitude))),
  };
}

#endif // !DATA_H
```

7.5 src/reader.h

```
#pragma once

#ifndef READER_H
#  define READER_H

#  include "data_entry.h"

#  include <vector>

std::vector<Entry> read_data(int argc, char **argv);

#endif // !READER_H
```

7.6 src/reader.cpp

```
#include "reader.h"

#include "data_entry.h"

#include <fstream>
#include <iostream>
#include <sstream>
#include <vector>
#include <cmath>

std::vector<Entry> read_data(int argc, char **argv) {
  if (argc != 2) {
    std::cout << "Usage: " << argv[0] << " <filename>" << std::endl;
    exit(EXIT_FAILURE);
  }

  std::ifstream file(argv[1]);
  if (!file.is_open()) {
```

```
    std::cerr << "Error opening file: " << argv[1] << std::endl;
    std::cerr << "Try `python3 run.py -p` to generate data file." << std::endl;
    exit(EXIT_FAILURE);
  }

  std::vector<Entry> entries;
  for (std::string line; std::getline(file, line);) {
    std::istringstream iss(line);
    Entry e;
    if (!(iss >> e.enodebid >> e.longitude >> e.latitude)) {
      std::cerr << "Error reading line: " << line << std::endl;
      continue;
    }
    entries.emplace_back(e);
  }
  return entries;
}
```

7.7 src/preprocess.py

```
#! /usr/bin/env python3
# -*- coding: utf-8 -*-


from typing import List

import dataclasses
import xlrd


@dataclasses.dataclass
class Entry:
    enodebid: int
    longitude: str
    latitude: str


def read_data(path: str) -> List[Entry]:
    data: List[Entry] = []
    sheet = xlrd.open_workbook(path).sheet_by_index(0)
    for i in range(1, sheet.nrows):
        row = sheet.row_values(i)
        data.append(Entry(int(row[0]), row[1], row[2]))
```

```
    return data


def write_data(path: str, data: List[Entry]) → None:
    with open(path, 'w') as f:
        for entry in data:
            f.write(f'{entry.enodebid}\t{entry.longitude}\t{entry.latitude}\n')


def main() → None:
    data = read_data('./data/02-1 1033 个基站数据.xls')
    write_data('./data/stations.txt', data)


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

7.8 run.py

```
#! /usr/bin/env python3
# -*- coding: utf-8 -*-

from dataclasses import dataclass
from typing import List

import argparse
import os
import subprocess
import sys


SRC_DIR = './src'
TARGET_DIR = './target'
DATA_DIR = './data'

SRC_FILES = [
    'main.cpp',
    'reader.cpp',
    'algorithm.cpp',
]


class Argument:
```

```
    @dataclass
    class Default:
        CXX: str = 'g++'
        CXXFLAGS: str = '-W -Wall -Wextra -Wpedantic -O3 -std=c++20 -Isrc'
        LDFLAGS: str = ''

    def __init__(self):
        default = Argument.Default()
        for key in ('CXX', 'CXXFLAGS', 'LDFLAGS'):
            self.__setattr__(key, os.getenv(key))
            if self.__getattribute__(key) is None:
                print(
                                    f'Env var {key} not set, default to
`{default.__getattribute__(key)}`'
                )
                self.__setattr__(key, default.__getattribute__(key))


def run_cmd(*kargs) -> int:
    cmd = ' '.join(kargs)
    print(f'\033[34mExecuting:\033[0m `{cmd}`', end=' ')
    sys.stdout.flush()
    ret = subprocess.run(cmd.split()).returncode
    if ret != 0:
        print(f'\033[31mFailed.\033[0m')
    else:
        print(f'\033[34mDone.\033[0m')
    return ret


def install_deps() -> None:
    try:
        import xlrd as _
    except ImportError:
        print('xlrd not installed, installing...')
        run_cmd(f'{sys.executable} -m pip install xlrd~=2.0.1')


def clean() -> None:
    run_cmd(f'rm -rf {TARGET_DIR}')


def preprocess() -> None:
    install_deps()
```

```
    run_cmd(f'python3 {SRC_DIR}/preprocess.py')


def compile() -> None:
    clean()

    if not os.path.exists(TARGET_DIR):
        os.mkdir(TARGET_DIR)

    arg = Argument()
    CXX: str = arg.CXX  # type: ignore
    CXXFLAGS: List[str] = arg.CXXFLAGS  # type: ignore
    LDFLAGS: List[str] = arg.LDFLAGS  # type: ignore

    for src_file in SRC_FILES:
        if run_cmd(
            f'{CXX} {CXXFLAGS} -c {SRC_DIR}/{src_file} '
            f'-o {TARGET_DIR}/{src_file}.o {LDFLAGS}'
        ) != 0:
            exit(1)

    if run_cmd(
        f'{CXX} {CXXFLAGS} '
        f'{" ".join([f"{TARGET_DIR}/{src_file}.o" for src_file in SRC_FILES])}'
        f' -o {TARGET_DIR}/main {LDFLAGS}'
    ) != 0:
        exit(1)


def run(compiled: bool) -> None:
    if not compiled or not os.path.exists(TARGET_DIR) \
            or not os.path.exists(f'{TARGET_DIR}/main'):
        print('Target dir not found, compiling...')
        compile()

    run_cmd(f'{TARGET_DIR}/main {DATA_DIR}/stations.txt')


def main() -> None:
    parser = argparse.ArgumentParser()
    parser.add_argument(
        '-c', '--compile', action='store_true', help='compile the C++ code'
    )
    parser.add_argument(
```

```
        '-p', '--preprocess', action='store_true', help='preprocess the data'
    )
    parser.add_argument(
        '-r', '--run', action='store_true', help='run the program'
    )
    parser.add_argument(
        '--clean', action='store_true', help='clean the target dir'
    )
    args = parser.parse_args()

    if args.clean:
        clean()

    if args.preprocess:
        preprocess()
    if args.compile:
        compile()
    if args.run:
        run(args.compile)

    if not any([args.preprocess, args.compile, args.run, args.clean]):
        parser.print_help()


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

点此回到附录以前。