《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

第四章 贪心：最短路径问题

2021211110 杜抒泽

2023 年 12 月 13 日

目录

1 实验内容

1.1 实验要求

- 利用“附件 1-1.基站图的邻接矩阵-v1-23”给出的 LTE 网络基站数据，以基站为顶点，以基站间的距离连线为边，组成图，计算图中的单源最短路径图构造。
- 从昆明 LTE 网络中，选取部分基站，计算基站间的距离，在部分基站间引入边，得到
  - 22 个基站顶点组成的图
  - 42 个基站顶点组成的图
- 对 22 个基站顶点组成的图，以基站 567443 为源点【说明：可以选择其它顶点作为源点】
  1. 计算 567443 到其它各点的单源最短路径
  2. 计算 567443 到 33109 的最短路径
- 对 42 个基站顶点组成的图，以基站 565845 为源点【说明：可以选择其它顶点作为源点】
  1. 计算 565845 到其它各点的单源最短路径
  2. 计算 565845 到 565667 的最短路径

1.2 题目需求分析

本次实验的主要内容是对于图论意义上的图求单源最短路径。题给数据中直接给出了图的邻接矩阵，图带权。没有限定图是有向图还是无向图，包不包含重边和自环。我们假定图是无向图（题给数据按照无向图处理），不会包含重边和自环。由于是贪心章节的作业，基本限死了只能使用 Dijkstra 算法[1]。

形式化地，给定图 $G=(V,E)$，对于 G 中的某个结点 S，求出 S 到 G 中所有结点的最短路径。图中的路径定义为一个结点序列 $v_0,v_1,...,v_k$，使得存在边的序列 $e_1,e_2,...,e_k$，满足对于每个 e_i 都存在 $e_i=(v_{i-1},v_i,w_i)$，其中 w_i 是 e_i 的权值。若 $v_0=S$，则称这个路径是 S 到 v_k 的一条路径。路径的长度定义为所有边的权值之和 $W=\sum_{i=1}^{k} w_i$。从 S 到结点 u 的最短路径即为所有从 S 到 u 的路径中 W 最小的路径。

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。

[1]其他最短路径算法基本和贪心不搭边，即使是代码实现看起来几乎一模一样的队列维护 Bellman-Ford 算法（俗称 SPFA）。

- 将图抽象为类型，使底层数据结构对使用者透明，同时包含从文本文件中读入图的接口。
- 在实现的类型上编写堆优化 Dijkstra 算法，求出图中某个结点到所有结点的最短路径的长度，以及能求出全部的路径的序列的信息。
- 编写大规模数据生成器，能够生成给定规模的随机图。
- 在不同规模的数据上运行程序，统计运行时间，分析算法的时间复杂度。
- 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

由于我是算法竞赛出身，在讲解算法时可能并不会使用和课上相同的术语，但我保证描述一定干练、正确且规范，表达我已经完全掌握了算法的思想，并且希望读者能够更好地理解算法的本质。

2.1 $\mathcal{O}(n^2)$ 朴素 Dijkstra 算法与 $\mathcal{O}(m \log n)$ 堆优化

2.1.1 题目背景与数学抽象

给定一个不含重边和自环的有向图 $G = (V, E)$ 和源点 S。边带正权，也即命题 $\forall e_i \in E(e_i = (u_i, v_i, w_i), w_i > 0)$ 为真。求出 S 到 V 中所有结点的最短路径。要求不仅求得最短路径的长度，而且对于任给的结点 $u \in V$，能够在 $\mathcal{O}(|V|)$ 的时间内求得并输出 S 到 u 的最短路径中所有结点的序列。

注解

求出从 S 到 V 中所有结点 u 的最短路径是 $\mathcal{O}\left(|V|^2\right)$ 的，因为这个答案的规模（所有路径的包含的结点数之和）就是 $\mathcal{O}\left(|V|^2\right)$ 的，不可能在低于该复杂度的时间内全部求得。

为了不让这个时间成为算法的瓶颈，我们可以在 Dijkstra 算法中求得一些信息，使得在算法结束后，对于从 S 到任何结点 $u \in V$ 的最短路径，都可以在与其长度成线性的时间内完成对对路径内容的询问的回答，这是具备现实意义的。

在以下的讨论中，我们记 $n = \Theta(|V|)$，$m = \Theta(|E|)$。

2.1.2 算法介绍

Dijkstra 是基于松弛操作的最短路算法。因此我们先介绍松弛操作。对于图 $G=(V,E)$，记 $dis(u)$ 为目前求得的从 S 到 u 的最短路径的长度。对于有向边 $e=(u,v,w)$，松弛操作对应这个式子：

$$dis(v) \leftarrow \min(dis(v), dis(u)+w)$$

这个式子的含义是显然的：尝试用路径 $S \to \ldots \to u \to v$ 这条路径去更新 $S \to \ldots \to v$ 的最短路径。如果这条路径的长度更小，那么就更新 $dis(v)$ 为这个更小的值。

Dijkstra 算法基于贪心的思想得来。算法流程如下：

将结点分为两个集合 A 和 B。A 中的结点是已经求得最短路径的结点，B 中的结点是还没有求得最短路径的结点。同时维护一个数组 dis，$dis(u)$ 表示从 S 到 u 的已经求得的最短路径的长度。算法开始时，$A=\{S\}$，$B=V-\{S\}$，$dis(S)=0$，$dis(u)=+\infty$（$u \in B$）。然后重复以下操作：

1. 从 B 中选取一个结点 u，使得 $dis(u)$ 最小。将 u 加入 A，从 B 中删除 u。
2. 对于所有 u 的出边 $e=(u,v,w)$，进行松弛操作。

直到 B 集合为空或对于所有 $u \in A, v \in B$，不存在边 $e=(u,v,w)$，算法结束。算法结束后，对于每个结点 $u \in V$，若 $dis(u)=\infty$，则说明结点 u 从 S 出发不可达。否则 $dis(u)$ 为 S 到 u 的最短路径的长度。

讨论算法的正确性。

下面采用数学归纳法证明在所有边权值非负的前提下 Dijkstra 算法的正确性。我们要证明的在执行 1 操作时，取出的结点 u 最短路径均已经被确定，其长度为 $dis(u)$。初始时 $A=\{S\}$，$dis(S)=0$，前提成立。

采用反证法证明归纳。设 u 点为第一个在加入 A 集合时不满足 $dis(u)$ 是最短路径长度的点。显然将 u 加入 A 前，$A \neq \varnothing$（因为一定有 $\{S\} \subset A$）。

如果不存在 S 到 u 的路径，则 $dis(u)=+\infty$，等于最短路径长度，与假设矛盾。于是一定存在路径 $S \to \ldots \to x \to \ldots \to y \to \ldots \to u$，其中 y 为 $S \to \ldots \to u$ 路径上第一个属于集合 B 的点，x 为 y 的前驱结点（显然 $x \in A$）。需要注意可能有 $S=x$ 或 $y=u$。由 $x \in A$ 可知 $dis(x)$ 是最短路径的长度；当 x 被加入 A 时，边 (x,y) 会被松弛，从而将 u 加入到 S 时，一定有 $dis(y)$ 是最短路径的长度。

记每个点 $u \in V$ 的实际最短路径的长度为 $D(u)$。在路径 $S \to \ldots \to x \to \ldots \to y \to \ldots \to u$ 中，因为所有边边权非负，因此 $D(y) \leq D(u)$。从而 $dis(y) \leq D(y) \leq D(u) \leq dis(u)$。根据假设，

结点 u 被取出集合 B 时结点 y 还没有被取出集合 B，因此此时有 $dis(u) \le dis(y)$，从而得到 $dis(y) = D(y) = D(u) = dis(u)$，与 $D(u) \neq dis(u)$ 的假设矛盾，故假设不成立。

因此我们证明了，1 操作每次取出的点，其最短路径均已经被确定。∎

注意到证明过程中的关键不等式 $D(y) \le D(u)$ 是在图上所有边边权非负的情况下得出的。当图上存在负权边时，这一不等式不再成立，Dijkstra 算法的正确性将无法得到保证，算法可能会给出错误的结果。

讨论时间复杂度。

朴素地维护 B 操作中的结点及其 dis 值是 $\mathcal{O}(n^2)$ 的。上述 1 操作会进行 $\mathcal{O}(n)$ 次，每次需要 $\mathcal{O}(n)$ 的时间遍历 B 集合找到 dis 的最小值，以及取到最小值的结点；2 操作会在每条边上进行一次松弛操作，因此是 $\mathcal{O}(m)$ 的。总时间复杂度是 $\mathcal{O}(n^2 + m) = \mathcal{O}(n^2)$ 的。

用二叉堆（或优先队列）维护 B 操作中的结点及其 dis 值是 $\mathcal{O}(m \log n)$ 的。每成功松弛一条边，就将 v 插入小根堆中，其排序键为 $dis(v)$；进行 1 操作时，直接取出堆顶元素即可。堆中元素的个数是 $\mathcal{O}(m)$ 的，因此总复杂度是 $\mathcal{O}(m \log m)$ 的。注意到不含重边、自环的有向图中 $m \le 2 \times \binom{n}{2} = n(n-1)$，因此 $m = \mathcal{O}(n^2)$，$\log m = \mathcal{O}(\log n)$，$m \log m = \mathcal{O}(m \log n)$。

用 Fibonacci 堆维护 B 操作中的结点及其 dis 值是 $\mathcal{O}(m + n \log n)$ 的，但常数较大，实践中不够优秀，也难以实现，故不在此过多介绍。

总之，堆优化 Dijkstra 算法的时间复杂度是 $\mathcal{O}(m \log n)$ 的，需要用到 $\mathcal{O}(n)$ 的额外空间。

至此我们求得了从 S 出发到所有点 $u \in V$ 的最短路径长度。接下来我们需要求得从 S 出发到所有点 $u \in V$ 的最短路径的序列。

这个问题的解决方法是在松弛操作中维护一个前驱数组 $from$。$from(u)$ 表示从 S 到 u 的最短路径中，u 的前驱结点。对于每次松弛操作，若松弛成功，则同时更新 $from(v) \leftarrow u$。这样，当算法结束后，对于每个结点 $u \in V$，$from(u)$ 就是 S 到 u 的最短路径中，u 的前驱结点。从而可以通过递归向前 $\mathcal{O}(n)$ 地查询 $from$ 数组，求得 S 到 u 的最短路径的序列。

至此，我们求得了从 S 出发到所有点 $u \in V$ 的最短路径的长度和序列信息。利用堆优化，时间复杂度是 $\mathcal{O}(m \log n)$ 的，需要用到 $\mathcal{O}(n)$ 的额外空间。

2.1.3 算法实现

实现非常简洁，但其中包含了诸多细节，这是在算法竞赛的长期发展中被所有人一同积累的经验与最佳实践。

首先，我创建了数组 `dis`，`from` 和 `vis`，其中 `dis` 和 `from` 分别与上文中 dis 和 $from$ 意义相同。`vis` 用于标记结点是否已经被加入到 A 集合中。

dis 集合中的无穷大用 std::numeric_limits<T>::max() / 2 表示。除以 2 的原因是，在松弛操作中会将两个 dis 值相加，如果不除以 2，可能会发生有符号整数溢出。

优先队列的实现采用了 C++ STL 中的 std::priority_queue。其默认是大根堆，但是我们需要小根堆。我们对此的解决方案是将其中比较关键字 dis 的值取负，这样就可以用默认的大根堆实现小根堆。这个技巧在算法竞赛中被广泛使用。

以下是具体代码：

```
template<typename T>
  requires std::is_arithmetic_v<T>
inline std::pair<std::vector<T>, std::vector<int>>
dijkstra(const Graph<T> &g, int src) {
  std::vector<T> dis(g.n + 1, std::numeric_limits<T>::max() / 2);
  std::vector<int> from(g.n + 1, -1);
  std::vector<bool> vis(g.n + 1);

  std::priority_queue<std::pair<T, int>> q;
  q.push({dis[src] = 0, src});
  while (!q.empty()) {
    int u = q.top().second;
    q.pop();
    if (vis[u])
      continue;
    vis[u] = true;
    for (auto [v, w] : g.adj[u])
      if (dis[v] > dis[u] + w) {
        from[v] = u;
        q.push({-(dis[v] = dis[u] + w), v});
      }
  }
  return {dis, from};
}
```

3 程序设计说明

项目名称为 GRD（贪心 GReeDy 的缩写）。程序在类 Unix 系统下开发与调试，Python 代码用于数据预处理，大规模数据的生成和控制程序的编译、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] [-r] [-R] [-c] [--clean]

options:
  -h, --help       show this help message and exit
  -r, --run        run the program with given data
  -R, --run-extra  run the program with extra data
  -c, --compile    compile the C++ code
  --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 -r # 使用样例输入运行程序（若不存在则先编译）
./run.py -R # 生成大规模输入并运行程序（若不存在则先编译）
./run.py --clean # 清理编译产物

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

3.2 图数据结构的抽象

我在 graph.h 中实现了图的抽象，定义如下：

```
template<typename T>
  requires std::is_arithmetic_v<T>
struct Graph {
  int n, src;
  std::vector<int> id;
  std::vector<std::vector<std::pair<int, T>>> adj;

  void push_edge(int u, int v, T w);

  Graph(int n_, int src_);

  static std::vector<Graph<T>> read_from_adjacency_matrix(std::istream &is);

  static std::vector<Graph<T>> read_from_edge_list(std::istream &is);
};
```

图的类型是 `Graph<T>`，其中 `T` 是边权的类型。存储采用邻接表，`adj` 是一个 `std::vector<std::vector<std::pair<int, T>>>`，其中 `adj[u]` 是结点 `u` 的出边集合，其中的元素 `[v, w]` 表示从 `u` 到 `v` 的一条边，边权为 `w`。`n` 是图中结点的个数，`src` 是图的源点。`id` 是一个 `std::vector<int>`，其中 `id[u]` 表示结点 `u` 在原图中的编号。

3.3 输入数据的格式

输入数据支持两种格式：邻接矩阵和边列表。邻接矩阵的格式如下：

```
t // 图的个数

n1 src1 // 第一个图的结点个数和源点编号
id1_1 id1_2 ... id1_n1 // 第一个图的结点编号
a1_11 a1_12 ... a1_1n1 // 第一个图的邻接矩阵
a1_21 a1_22 ... a1_2n1
...
a1_n11 a1_n12 ... a1_n1n1

n2 src2 // 第二个图的结点个数和源点编号
id2_1 id2_2 ... id2_n2 // 第二个图的结点编号
a2_11 a2_12 ... a2_1n2 // 第二个图的邻接矩阵
...
a2_n21 a2_n22 ... a2_n2n2

... // 更多图
```

边列表的格式如下：

```
t // 图的个数

n1 m1 src1 // 第一个图的结点个数、边数和源点编号
u1_1 v1_1 w1_1 // 第一个图的边列表
u1_2 v1_2 w1_2
...
u1_m1 v1_m1 w1_m1

n2 m2 src2 // 第二个图的结点个数、边数和源点编号
u2_1 v2_1 w2_1 // 第二个图的边列表
...
u2_m2 v2_m2 w2_m2

... // 更多图
```

其中边列表格式中默认结点的编号依次为 $1, 2, ..., n$，也即 $\mathrm{id}_i = i$。

3.4 算法接口

前文中提到 dijkstra 函数的签名为 `std::pair<std::vector<T>, std::vector<int>> dijkstra(const Graph<T> &g, int src)`，接收两个参数，分别是图和源点索引。返回值包含两个数组，分别是算法介绍环节中的 *dis* 和 *from*。

我为这个函数提供了两个包装器，分别如下（输出与计时省略）：

```cpp
template<typename T>
void algo_main(const Graph<T> &g) {
  auto get_index = [&](int id) -> int { /* ... */ }; // 获取编号对应的索引的工具函数
  auto [dis, from] = dijkstra(g, get_index(g.src));

  // 输出 dijkstra 函数所用时间（单位为毫秒）

  // 分结点数小于等于 100 和大于 100 两种情况进行输出
  // 若结点数小于等于 100，输出所有结点的最短路径长度和序列
  // 若结点数大于 100，只输出最短路径中最长的一条的长度和序列
}

template<typename T>
void algo_main(const std::vector<Graph<T>> &graphs) {
  for (size_t i = 0; i < graphs.size(); ++i)
    algo_main(graphs[i]);
}
```

它们分别实现了对于单个图和多个图的运行。对于单个图，我们需要将源点的编号转换为索引，这个过程由 get_index 函数完成。对于多个图，我们只需要对每个图调用单个图的运行函数即可。

3.5 大规模数据的生成

大规模数据的生成使用了 Python 第三方库 CYaRon。相关代码如下：

```python
def gen_cyaron_data(testcases: int) -> List[Tuple[int, int, Graph]]:
    N = 300000 # 结点数
    M = 1000000 # 边数

    return [(N, M, Graph.graph(
        N, M, self_loop=False, repeated_edges=False, directed=True,
        weight_limit=(((1 << 30) - 1) // M)
    )) for _ in range(testcases)]
```

其中 `Graph.graph` 是 CYaRon 提供的生成图的函数。我生成了若干个结点数为 N，边数为 M 的不含自环和重边的有向图，将其作为测试数据。N 和 M 的值可以被修改，用来测试不同规模的数据。

这个第三方库的性能不尽人意，但是生成的数据质量还算可以接受。

4 测试、运行结果与分析说明

4.1 小规模数据：正确性测试

拿到代码运行 `./run.py -cr`，对于两个图的输出分别如下：

```
$ CXX=g++-13 ./run.py -cr
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 ./target/main.cpp.o -o ./target/main `
Done.
Executing: `python3 ./src/preprocess.py -i ./data/附件1-1.基站图的邻接矩阵-v1-23.xls -o ./data/graphs.txt`
Done.
Executing: `./target/main ./data/graphs.txt 1`
Found 2 graph(s)
Graph #1
Time Elapsed: 0.005 ms
Shortest path from 567443: {
   33109: [length 1956.926] 567443 → 566750 → 567439 → 33109
  565696: [length 1343.406] 567443 → 566783 → 566993 → 565696
  566631: [length  761.938] 567443 → 566783 → 566631
  566720: [length 2111.289] 567443 → 566750 → 567439 → 566751 → 566720
  566742: [length  302.540] 567443 → 566742
  566747: [length 1988.138] 567443 → 566742 → 566802 → 567322 → 566747
  566750: [length  683.088] 567443 → 566750
  566751: [length 1622.912] 567443 → 566750 → 567439 → 566751
  566783: [length  344.546] 567443 → 566783
  566798: [length 1778.055] 567443 → 566750 → 567439 → 566798
  566802: [length  963.852] 567443 → 566742 → 566802
  566967: [length 1562.252] 567443 → 566783 → 566993 → 566967
  566993: [length  988.629] 567443 → 566783 → 566993
  566999: [length 2072.921] 567443 → 566783 → 566993 → 566967 → 566999
  567203: [length 1592.307] 567443 → 566783 → 566993 → 567203
  567238: [length  780.892] 567443 → 566783 → 567238
  567260: [length  244.053] 567443 → 567260
  567322: [length 1582.908] 567443 → 566742 → 566802 → 567322
  567439: [length 1309.046] 567443 → 566750 → 567439
  567443: [length    0.000] 567443
  567547: [length 1733.000] 567443 → 566750 → 567439 → 567547
  568098: [length  810.555] 567443 → 566742 → 568098
}
```

```
Graph #2
Time Elapsed: 0.003 ms
Shortest path from 565845: {
  565675: [length 1369.368] 565845 → 567526 → 567500 → 565675
  565621: [length 1928.895] 565845 → 566010 → 565631 → 565801 → 565630 → 565621
  565667: [length 2900.123] 565845 → 567526 → 567500 → 565675 → 565551 → 565633 → 565667
  567510: [length  645.041] 565845 → 567526 → 567510
  565801: [length 1153.114] 565845 → 566010 → 565631 → 565801
  566010: [length  403.433] 565845 → 566010
  567891: [length 2401.899] 565845 → 567526 → 567500 → 565675 → 565551 → 567891
  565492: [length 2223.014] 565845 → 567526 → 567500 → 565675 → 565551 → 565492
  565558: [length 2171.292] 565845 → 567526 → 567500 → 565675 → 565551 → 565558
  565627: [length 2697.463] 565845 → 567526 → 567500 → 565675 → 565551 → 565558 → 565627
  565572: [length 2440.918] 565845 → 567526 → 567500 → 566074 → 565610 → 565572
  565610: [length 2025.887] 565845 → 567526 → 567500 → 566074 → 565610
  565859: [length 2050.978] 565845 → 567526 → 565964 → 567531 → 565859
  565630: [length 1468.955] 565845 → 566010 → 565631 → 565801 → 565630
  565559: [length 2381.337] 565845 → 567526 → 567500 → 565675 → 565516 → 565559
  565845: [length    0.000] 565845
  565527: [length 2594.338] 565845 → 566010 → 565631 → 565801 → 565630 → 565648 → 565527
  565633: [length 2347.844] 565845 → 567526 → 567500 → 565675 → 565551 → 565633
  565496: [length 2308.237] 565845 → 566010 → 565631 → 565801 → 565630 → 565648 → 565496
  565865: [length 2489.072] 565845 → 567526 → 565964 → 567531 → 565859 → 565865
  565773: [length 2281.464] 565845 → 566010 → 565631 → 565801 → 565630 → 565621 → 565773
  567531: [length 1402.786] 565845 → 567526 → 565964 → 567531
  565516: [length 1918.104] 565845 → 567526 → 567500 → 565675 → 565516
  565393: [length 2339.027] 565845 → 567526 → 565964 → 567531 → 565859 → 565393
  565753: [length 1122.454] 565845 → 566010 → 565562 → 565753
   33566: [length 2169.678] 565845 → 566010 → 565562 → 565753 → 567618 → 33566
  566074: [length 1573.635] 565845 → 567526 → 567500 → 566074
  565648: [length 1997.174] 565845 → 566010 → 565631 → 565801 → 565630 → 565648
  567526: [length  488.237] 565845 → 567526
  565551: [length 1806.748] 565845 → 567526 → 567500 → 565675 → 565551
  565631: [length  843.923] 565845 → 566010 → 565631
  565608: [length 1883.380] 565845 → 566010 → 565562 → 565753 → 567618 → 565608
  567500: [length 1055.673] 565845 → 567526 → 567500
  565531: [length 2161.483] 565845 → 566010 → 565562 → 565753 → 567618 → 565531
  565562: [length  853.566] 565845 → 566010 → 565562
   32788: [length 2187.665] 565845 → 567526 → 565964 → 567531 → 565859 → 32788
  567497: [length 1561.461] 565845 → 566010 → 565562 → 565753 → 567497
  566316: [length 2592.690] 565845 → 567526 → 567500 → 565675 → 565551 → 565558 → 566316
  568056: [length 2787.200] 565845 → 567526 → 567500 → 565675 → 565551 → 565633 → 568056
  565964: [length  741.608] 565845 → 567526 → 565964
  567618: [length 1655.160] 565845 → 566010 → 565562 → 565753 → 567618
  565898: [length  978.426] 565845 → 566010 → 565898
}
```

图 2　运行结果

图片中包含大量信息。

- 对于图 1 输出了从基站 567443 到所有基站的最短路径长度和序列。
  - 算法运行了 0.005 毫秒。
  - 例如，从基站 567443 到基站 33109 的最短路径长 1956.926，路径为 `567443` → `566750` → `567439` → `33109`。
- 对于图 2 输出了从基站 565845 到所有基站的最短路径长度和序列。
  - 算法运行了 0.003 毫秒。

- 例如，从基站 565845 到基站 565667 的最短路径长 2900.123，路径为 `565845 → 567526 → 567500 → 565675 → 565551 → 565633 → 565667`。

我们可以更改代码使得程序输出其他信息，例如，将图 1 的源点更改为 33109，图 2 的源点更改为 565667，再次运行 `./run.py -r`，得到的输出如下：

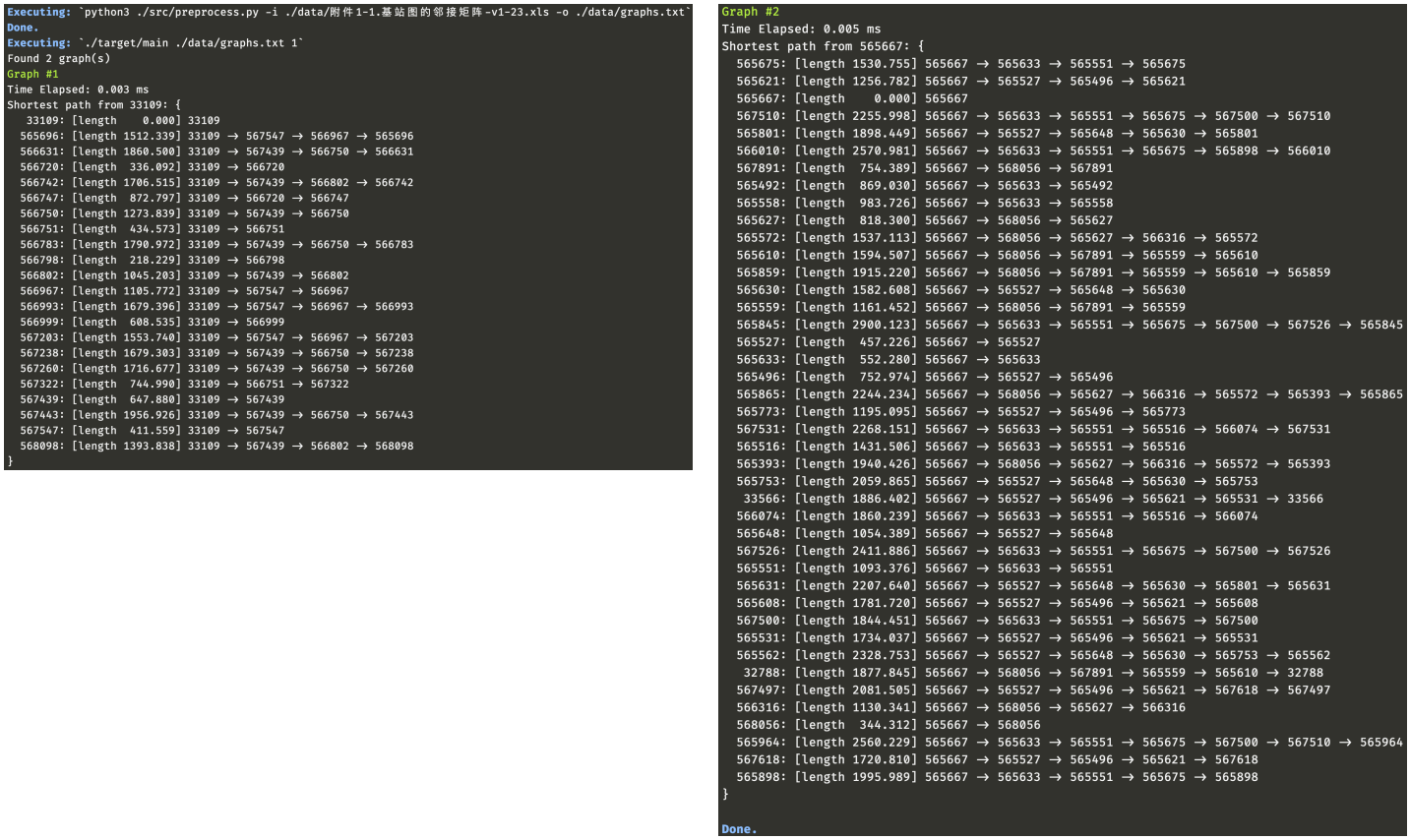

图 3　运行结果

可以验证图中的输出都是正确的，不再赘述。

4.2 大规模数据：时间分析

我使用了大规模数据生成器生成了以下配置的输入，配置以及得到的输出如下：

- 5 个结点树为 30000，边数为 100000 的图。

  算法在 5 张图上运行的平均时间为 4.014 毫秒。

```
$ CXX=g++-13 ./run.py -R
Executing: `python3 ./src/preprocess.py -t 5 -o ./data/extra_graphs.txt`
Generating data ... Please wait, this could take a while.
Generating 5 graphs with 30000 nodes and 100000 edges ...
Data generated in 2.21 second(s).
Done.
Executing: `./target/main ./data/extra_graphs.txt 2`
Found 5 graph(s)
Graph #1
Time Elapsed: 4.236 ms
The vertex can be reached from 1 with longest shortest path is 2311 with length 65460:
 1 → 28878 → 8582 → 8497 → 29637 → 8370 → 28813 → 15132 → 24207 → 11452 → 15610 → 16278 → 17512 → 17649 → 8449 → 919 → 18192 → 21421 → 2311

Graph #2
Time Elapsed: 4.032 ms
The vertex can be reached from 1 with longest shortest path is 22258 with length 71909:
 1 → 5848 → 13434 → 3344 → 29097 → 4098 → 10137 → 16631 → 15830 → 19771 → 18632 → 29564 → 13464 → 23067 → 3834 → 14821 → 25068 → 17081 → 10478 → 22258

Graph #3
Time Elapsed: 4.008 ms
The vertex can be reached from 1 with longest shortest path is 7921 with length 73491:
 1 → 4927 → 2947 → 2946 → 16997 → 15447 → 474 → 8193 → 19638 → 18813 → 20730 → 4292 → 20034 → 8227 → 13579 → 13019 → 4632 → 7921

Graph #4
Time Elapsed: 3.913 ms
The vertex can be reached from 1 with longest shortest path is 6753 with length 69349:
 1 → 23794 → 27122 → 1881 → 7214 → 26054 → 18806 → 17690 → 5158 → 21206 → 2521 → 12932 → 6753

Graph #5
Time Elapsed: 3.879 ms
The vertex can be reached from 1 with longest shortest path is 8710 with length 66626:
 1 → 10719 → 26507 → 5646 → 499 → 12566 → 18982 → 11399 → 27754 → 481 → 10735 → 10362 → 13002 → 1760 → 1887 → 23561 → 15569 → 16986 → 8710

Done.
```

图 4　$N = 30000, M = 100000$

- 5 个结点数为 30000，边数为 200000 的图。

  算法在 5 张图上运行的平均时间为 6.257 毫秒。

```
$ CXX=g++-13 ./run.py -R
Executing: `python3 ./src/preprocess.py -t 5 -o ./data/extra_graphs.txt`
Generating data ... Please wait, this could take a while.
Generating 5 graphs with 30000 nodes and 200000 edges ...
Data generated in 5.09 second(s). Sorry for the delay!
Done.
Executing: `./target/main ./data/extra_graphs.txt 2`
Found 5 graph(s)
Graph #1
Time Elapsed: 6.413 ms
The vertex can be reached from 1 with longest shortest path is 14781 with length 18595:
 1 → 7476 → 10950 → 6652 → 484 → 27014 → 1681 → 8991 → 8414 → 14917 → 29771 → 20306 → 21562 → 23718 → 22849 → 5357 → 14781

Graph #2
Time Elapsed: 6.278 ms
The vertex can be reached from 1 with longest shortest path is 14933 with length 18379:
 1 → 17524 → 17929 → 8525 → 16831 → 9160 → 23714 → 2569 → 20384 → 10405 → 22662 → 14933

Graph #3
Time Elapsed: 6.146 ms
The vertex can be reached from 1 with longest shortest path is 19954 with length 16883:
 1 → 16265 → 16660 → 11637 → 11116 → 7030 → 17531 → 19954

Graph #4
Time Elapsed: 6.388 ms
The vertex can be reached from 1 with longest shortest path is 29688 with length 17368:
 1 → 20243 → 8263 → 12558 → 13994 → 4913 → 18634 → 16764 → 23291 → 9201 → 6368 → 13373 → 4328 → 27497 → 29688

Graph #5
Time Elapsed: 6.057 ms
The vertex can be reached from 1 with longest shortest path is 22775 with length 16122:
 1 → 23686 → 20158 → 13869 → 1455 → 1754 → 17401 → 29236 → 11393 → 23881 → 14951 → 7002 → 26873 → 22775

Done.
```

图 5　$N = 30000, M = 200000$

- 5 个结点数为 300000，边数为 1000000 的图。

  算法在 5 张图上运行的平均时间为 70.134 毫秒。

```
$ CXX=g++-13 ./run.py -R
Executing: `python3 ./src/preprocess.py -t 5 -o ./data/extra_graphs.txt`
Generating data ... Please wait, this could take a while.
Generating 5 graphs with 300000 nodes and 1000000 edges ...
Data generated in 32.38 second(s). Sorry for the delay!
Done.
Executing: `./target/main ./data/extra_graphs.txt 2`
Found 5 graph(s)
Graph #1
Time Elapsed: 71.365 ms
The vertex can be reached from 1 with longest shortest path is 5110 with length 9882:
 1 → 185477 → 146324 → 298619 → 175489 → 249031 → 1321 → 36582 → 61448 → 139937 → 186992 → 133705 → 45894 → 269131 → 44083 → 41719 → 85913 → 251529 → 4002 → 5110

Graph #2
Time Elapsed: 72.440 ms
The vertex can be reached from 1 with longest shortest path is 251298 with length 8341:
 1 → 241478 → 288992 → 106560 → 36654 → 250699 → 153824 → 88160 → 103861 → 168161 → 170587 → 299880 → 72938 → 249556 → 299716 → 61658 → 157712 → 12128 → 37370 → 251298

Graph #3
Time Elapsed: 66.321 ms
The vertex can be reached from 1 with longest shortest path is 218306 with length 8555:
 1 → 120000 → 30047 → 270474 → 77637 → 86155 → 162213 → 153158 → 36361 → 145046 → 230613 → 288284 → 121814 → 221314 → 1333 → 230425 → 218306

Graph #4
Time Elapsed: 69.561 ms
The vertex can be reached from 1 with longest shortest path is 197439 with length 8376:
 1 → 129627 → 233864 → 162620 → 273354 → 187395 → 235732 → 228942 → 274871 → 40962 → 161616 → 247076 → 184829 → 77614 → 56455 → 178689 → 291720 → 132857 → 255402 → 65077 → 245563 → 130569 → 167443 → 184665 → 213216 → 197439

Graph #5
Time Elapsed: 71.022 ms
The vertex can be reached from 1 with longest shortest path is 31686 with length 8858:
 1 → 150867 → 164658 → 150762 → 199718 → 187705 → 201830 → 231920 → 107319 → 44005 → 184846 → 212762 → 144525 → 89935 → 138600 → 177903 → 120084 → 192310 → 187452 → 17500 → 199284 → 11135 → 97180 → 240707 → 259121 → 31686
```

图 6　$N = 300000, M = 1000000$

- 5 个结点数为 300000，边数为 2000000 的图。

  算法在 5 张图上运行的平均时间为 95.727 毫秒。

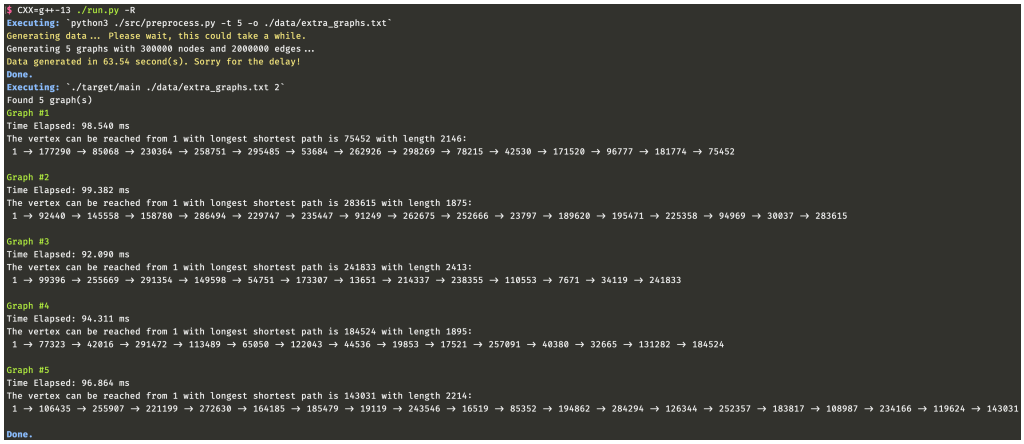

图 7　$N = 300000, M = 2000000$

以下是统计表格：

序号	数据规模	实际运行时间
1	$n = 22, m = 134$	0.005 ms
2	$n = 42, m = 228$	0.003 ms
3	$n = 30000, m = 100000$	4.014 ms
4	$n = 30000, m = 200000$	6.257 ms
5	$n = 300000, m = 1000000$	70.134 ms
6	$n = 300000, m = 2000000$	95.727 ms

图 8　算法运行时间统计表

算法的高性能可见一斑！

从表中可以看出，所有算法的运行时间大致符合时间复杂度理论估计 $\mathcal{O}(m \log n)$ 的预期：

- 首先排除前两组数据，因为数据量太小，时间抖动幅度甚至和总用时相当。
- 比较第三组和第四组数据，n 不变，m 扩大了 2 倍，理论应增加 2 倍，实际增加了约 1.5 倍。
- 比较第三组和第五组数据，n 和 m 都扩大了 10 倍，理论应增加 13.3 倍，实际增加了约 17.5 倍。
- 比较第四组和第六组数据，n 和 m 都扩大了 10 倍，理论应增加 13.3 倍，实际增加了约 15.3 倍。

- 比较第五组和第六组数据，n 不变，m 扩大了 2 倍，理论应增加 2 倍，实际增加了约 1.4 倍。

以上的实验结果表明，算法的时间复杂度是 $\mathcal{O}(m \log n)$ 的，且常数较小。

5 总结

本次实验到此完全结束。我在本次实验中实现了堆优化的 Dijkstra 算法，能够在 $\mathcal{O}(m \log n)$ 的时间内求出图中某个结点到所有结点的最短路径的长度和序列。我还实现了大规模数据生成器，能够生成给定规模的随机图。我在不同规模的数据上运行程序，统计运行时间，分析算法的时间复杂度。

- 可以尝试使用更多的数据结构，如 Fibonacci 堆，以进一步提高算法的性能。
- 在更多不同的数据量下进行测试，验证算法面对大数据集时的性能和鲁棒性。
- 与其他算法（如 Floyd 算法、Bellman-Ford 算法）进行对比，分析算法的优劣。
- 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。

6 附录 1 - 附件清单

```
GRD
├── run.py
└── src
    ├── algorithm.h
    ├── graph.h
    ├── main.cpp
    └── preprocess.py
```

7 附录 2 - 源代码

7.1 src/algorithm.h

```cpp
#pragma once

#ifndef ALGORITHM_H
#  define ALGORITHM_H

#  include "graph.h"

#  include <queue>
#  include <type_traits>
#  include <vector>

// clang-format off
template<typename T>
  requires std::is_arithmetic_v<T>
inline std::pair<std::vector<T>, std::vector<int>>
dijkstra(const Graph<T> &g, int src) {
  // clang-format on
  std::vector<T> dis(g.n + 1, std::numeric_limits<T>::max() / 2);
  std::vector<int> from(g.n + 1, -1);
  std::vector<bool> vis(g.n + 1);

  std::priority_queue<std::pair<T, int>> q;
  q.push({dis[src] = 0, src});
  while (!q.empty()) {
    int u = q.top().second;
    q.pop();
    if (vis[u])
      continue;
    vis[u] = true;
```

```
    for (auto [v, w] : g.adj[u])
      if (dis[v] > dis[u] + w) {
        from[v] = u;
        q.push({-(dis[v] = dis[u] + w), v});
      }
  }
  return {dis, from};
}

#endif // !ALGORITHM_H
```

7.2 src/graph.h

```
#pragma once

#ifndef GRAPH_H
#  define GRAPH_H

#  include <iostream>
#  include <type_traits>
#  include <vector>

template<typename T>
  requires std::is_arithmetic_v<T>
struct Graph {
  int n, src;
  std::vector<int> id;
  std::vector<std::vector<std::pair<int, T>>> adj;

  void push_edge(int u, int v, T w) {
    adj[u].push_back({v, w});
  }

  Graph(int n_, int src_): n(n_), src(src_) {
    if (n ⩽ 100)
      id.resize(n + 1);
    adj.resize(n + 1);
  }

  static std::vector<Graph<T>> read_from_adjacency_matrix(std::istream &is) {
    std::vector<Graph<T>> graphs;
    int t;
    for (is >> t; t; --t) {
```

```
      int n, src;
      is >> n >> src;
      Graph<T> g(n, src);
      for (int u = 1; u ⩽ n; ++u)
        is >> g.id[u];
      for (int u = 1; u ⩽ n; ++u)
        for (int v = 1; v ⩽ n; ++v) {
          T w;
          is >> w;
          if (w ≠ T(-1))
            g.push_edge(u, v, w);
        }
      graphs.push_back(g);
    }
    return graphs;
  }

  static std::vector<Graph<T>> read_from_edge_list(std::istream &is) {
    std::vector<Graph<T>> graphs;
    int t;
    for (is >> t; t; --t) {
      int n, m, src;
      is >> n >> m >> src;
      Graph<T> g(n, src);
      // for (int u = 1; u ⩽ n; ++u)
      //   is >> g.id[u];
      for (; m; --m) {
        int u, v;
        T w;
        is >> u >> v >> w;
        g.push_edge(u, v, w);
      }
      graphs.push_back(g);
    }
    return graphs;
  }
};

#endif // !GRAPH_H
```

7.3 src/main.cpp

```cpp
#include "algorithm.h"
#include "graph.h"

// 这些头文件全部都用到了
// 让我们感谢 C++ STL 的基础设施
// 我给 STL 恭恭敬敬磕大头
#include <chrono>
#include <cstdint>
#include <cstdlib>
#include <format>
#include <fstream>
#include <iostream>
#include <limits>
#include <string>
#include <vector>

template<typename T>
void algo_main(const Graph<T> &g) {
  auto start = std::chrono::high_resolution_clock::now();

  auto get_index = [&](int id) -> int {
    if (g.n > 100)
      return id;

    if (auto it = std::find(g.id.begin(), g.id.end(), id);
        it != g.id.end() && *it == id)
      return it - g.id.begin();
    throw std::invalid_argument(std::format("Invalid vertex id {}", id));
  };

  auto [dis, from] = dijkstra(g, get_index(g.src));
  auto end = std::chrono::high_resolution_clock::now();

  std::string buf = std::format(
      "Time Elapsed: {:.3f} ms\n",
      static_cast<double>(
          std::chrono::duration_cast<std::chrono::microseconds>(end - start)
              .count()
      ) / 1000
  );

  if (g.n > 100) {
    int max_index = 1;
    for (int i = 1; i <= g.n; ++i)
```

```
      if (dis[i] ≠ std::numeric_limits<T>::max() / 2 &&
          dis[i] > dis[max_index])
        max_index = i;

    buf += std::format(
        "The vertex can be reached from {} with longest shortest path is {} "
        "with length {}:\n",
        g.src, max_index, dis[max_index]
    );
    std::vector<int> path;
    for (int v = max_index; v ≠ -1; v = from[v])
      path.push_back(v);
    for (auto it = path.rbegin(); it ≠ path.rend(); ++it)
      buf += std::format(" {} →", *it);
    buf.pop_back(), buf.pop_back(), buf.pop_back();
    buf += "\n";
    std::cout << buf << std::endl;
    return;
  }

  buf += std::format("Shortest path from {}: {{\n", g.src);
  for (int u = 1; u ⩽ g.n; ++u) {
    if (dis[u] ≠ std::numeric_limits<T>::max() / 2) {
      if constexpr (std::is_integral_v<T>)
        buf += std::format("  {:6}: [length {}]", g.id[u], dis[u]);
      else if (std::is_floating_point_v<T>)
        buf += std::format("  {:6}: [length {:8.3f}]", g.id[u], dis[u]);

      std::vector<int> path;
      for (int v = u; v ≠ -1; v = from[v])
        path.push_back(g.id[v]);
      for (auto it = path.rbegin(); it ≠ path.rend(); ++it)
        buf += std::format(" {} →", *it);
      buf.pop_back(), buf.pop_back(), buf.pop_back();
    } else {
      buf += std::format("  {}: [unreachable] ", g.id[u]);
    }
    buf += "\n";
  }
  buf += "}\n";
  std::cout << buf << std::endl;
}

template<typename T>
```

```cpp
void algo_main(const std::vector<Graph<T>> &graphs) {
  std::cout << std::format("Found {} graph(s)\n", graphs.size());
  for (size_t i = 0; i < graphs.size(); ++i) {
    std::cout << std::format("\033[32mGraph #{}\033[0m\n", i + 1);
    algo_main(graphs[i]);
  }
}

enum class InputType : std::uint8_t {
  AdjacencyMatrix = 1,
  EdgeList = 2,
};

/// @param argc
/// @param argv {input file, input type}
int main(int argc, char **argv) {
  if (argc != 3) {
    std::cerr << "Usage: " << argv[0] << " <input file> <input type>\n";
    return EXIT_FAILURE;
  }
  char *input_file = argv[1];
  InputType input_type = static_cast<InputType>(std::stoi(argv[2]));

  std::ifstream ifs(input_file);
  if (!ifs) {
    throw std::runtime_error("Failed to open input file");
  }

  switch (input_type) {
    case InputType::AdjacencyMatrix:
      algo_main(Graph<double>::read_from_adjacency_matrix(ifs));
      break;
    case InputType::EdgeList:
      algo_main(Graph<int>::read_from_edge_list(ifs));
      break;
    default:
      throw std::invalid_argument("Invalid input type");
  }
}
```

7.4 src/preprocess.py

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


from typing import List, Tuple
from cyaron import Graph

import argparse
import dataclasses
import sys
import time
import xlrd


@dataclasses.dataclass
class AdjMat:
    n: int
    src: int
    id: List[int]
    weight: List[List[float]]


def read_adj_mat(path: str) -> List[AdjMat]:
    book = xlrd.open_workbook(path)

    graphs = []
    for sheet_index in (0, 1):
        sheet = book.sheet_by_index(sheet_index)
        first_row = sheet.row_values(0)
        n = int(max(first_row[2:]))

        second_row = sheet.row_values(1)
        id = list(map(int, second_row[2:]))

        weight = []
        for i in range(2, n + 2):
            row = sheet.row_values(i)
            weight.append(list(row[2:]))

        # inject
        if n == 22:
            src = 567443
        elif n == 42:
            src = 565845
```

```
        else:
            src = id[0]

        graphs.append(AdjMat(n, src, id, weight))
    return graphs


def write_adj_mat(path: str, data: List[AdjMat]) -> None:
    buf = []
    buf.append(f'{len(data)}\n')
    for graph in data:
        buf.append(f'{graph.n} {graph.src}\n')
        buf.append(' '.join(map(str, graph.id)))
        buf.append('\n')
        for row in graph.weight:
            buf.append(' '.join(map(str, row)))
            buf.append('\n')

    with open(path, 'w') as f:
        f.writelines(buf)


def gen_cyaron_data(testcases: int) -> List[Tuple[int, int, Graph]]:
    N = 300000
    M = 1000000

    print(f'Generating {testcases} graphs with {N} nodes and {M} edges ... ')
    return [(N, M, Graph.graph(
        N, M, self_loop=False, repeated_edges=False, directed=True,
        weight_limit=(((1 << 30) - 1) // M)
    )) for _ in range(testcases)]


def write_cyaron_data(path: str, data: List[Tuple[int, int, Graph]]) -> None:
    buf = []
    buf.append(f'{len(data)}\n')
    for (n, m, graph) in data:
        buf.append(f'{n} {m} 1\n')
        # buf.append(' '.join(map(str, range(1, n + 1))))
        # buf.append('\n')
        for edge in graph.iterate_edges():
            buf.append(f'{edge.start} {edge.end} {edge.weight}\n')

    with open(path, 'w') as f:
```

```
        f.writelines(buf)


def main() → None:
    parser = argparse.ArgumentParser()
    parser.add_argument('-t', '--testcases', dest='testcases', type=int)
    parser.add_argument('-i', '--input', dest='file_path', type=str)
    parser.add_argument('-o', '--output', dest='output_path', type=str)
    args = parser.parse_args()

    if args.testcases and args.file_path:
        sys.stderr.write('Cannot specify both testcases and file_path\n')
        parser.print_help()
    elif not args.testcases and not args.file_path:
        sys.stderr.write('Must specify either testcases or file_path\n')
        parser.print_help()
    elif args.testcases:
        print(
            '\033[33mGenerating data ... '
            'Please wait, this could take a while.\033[0m'
        )
        start = time.time()

        write_cyaron_data(args.output_path, gen_cyaron_data(args.testcases))

        time_cost = time.time() - start
        if time_cost ⩽ 5:
            print(
                f'\033[33mData generated in {time_cost:.2f} second(s).\033[0m'
            )
        else:
            print(
                f'\033[33mData generated in {time_cost:.2f} second(s). '
                'Sorry for the delay!\033[0m'
            )
    elif args.file_path:
        write_adj_mat(args.output_path, read_adj_mat(args.file_path))


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

7.5 run.py

```python
#! /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',
]


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, '
                    f'default to `{default.__getattribute__(key)}`'
                )
                self.__setattr__(key, default.__getattribute__(key))


def run_cmd(*kargs) -> int:
    cmd = ' '.join(kargs)
    print(f'\033[1;34mExecuting:\033[0m `{cmd}`')
    sys.stdout.flush()
```

```
    ret = subprocess.run(cmd.split()).returncode
    if ret ≠ 0:
        print(f'\033[1;31mFailed.\033[0m')
    else:
        print(f'\033[1;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')
    try:
        import cyaron as _
    except ImportError:
        print('cyaron not installed, installing...')
        run_cmd(f'{sys.executable} -m pip install cyaron~=0.5.0')


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


def preprocess() → None:
    install_deps()
    run_cmd(
        f'python3 {SRC_DIR}/preprocess.py '
        f'-i {DATA_DIR}/附件 1-1.基站图的邻接矩阵-v1-23.xls '
        f'-o {DATA_DIR}/graphs.txt'
    )


def gen_extra_data() → None:
    install_deps()
    run_cmd(
        f'python3 {SRC_DIR}/preprocess.py '
        f'-t 5 '
        f'-o {DATA_DIR}/extra_graphs.txt'
    )


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 and (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}/graphs.txt 1')


def run_extra(compiled: bool) -> None:
    if not compiled and (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}/extra_graphs.txt 2')


def main() -> None:
```

```
    parser = argparse.ArgumentParser()

    parser.add_argument(
        '-r', '--run', action='store_true', help='run the program with given
data'
    )
    parser.add_argument(
        '-R', '--run-extra', action='store_true', help='run the program with
extra data'
    )
    parser.add_argument(
        '-c', '--compile', action='store_true', help='compile the C++ code'
    )
    parser.add_argument(
        '--clean', action='store_true', help='clean the target dir'
    )
    args = parser.parse_args()

    if args.clean:
        clean()

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

    if not any((args.run, args.run_extra, args.compile, args.clean)):
        parser.print_help()


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

点此回到附录以前。