《数据结构与算法》三级项目报告

基于有向有权图的货运物流管理系统

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 姓　　名 |  |  |  |  |
| 学　　号 |  |  |  |  |
| 成　　绩 |  |  |  |  |
| 指导老师 | 窦燕 陈贺敏 李季辉 郝晓冰 李可 余扬 | | | |
| 时　　间 | 2023年12月 | | | |

2023年9月

# 摘要

本程序通过图的邻接数组实现和栈、优先队列等数据结构的综合应用，处理了实际生活中物流系统适用的相关问题。进入信息系统后，可以按需选择功能，图相关算法中多源最短路径用到了Floyd算法，单源最短路径用了Dijkstra算法、并尝试了A\*算法。此外也包含拓扑排序等功能算法。

程序中的图自动初始化时，可以选择读取已有文本数据，对图进行更新操作（增删点边）时，对应的操作也会影响到文件中保存的数据。用户可以按需选择自己想要的功能进行使用，或者输入自己想要处理的图的数据进行处理。这些功能的实现是我们小组根据自身对数据结构的理解和程序设计的经验得出的实现方式。

总体而言，程序实现了有向有权图的邻接数组实现的数据结构和一系列对应的基础方法和图算法，并尝试以此为基础开发物流管理系统的上层功能

关键词：物流信息系统、图、寻路算法、拓扑排序。

# 目录

[摘要 1](#_Toc3405)

[目录 2](#_Toc25717)

[前言 3](#_Toc10039)

[一、研究背景 3](#_Toc1516)

[二、研究目的 4](#_Toc13771)

[1.目的 4](#_Toc16230)

[2.范围 4](#_Toc8031)

[三、预期结果 4](#_Toc28614)

[四、项目组分工 4](#_Toc14731)

[正文 6](#_Toc18305)

[一、 研究内容的基本原理 6](#_Toc18240)

[二、 所采用的研究方法及相关工具 6](#_Toc18632)

[三、 项目的方案设计 6](#_Toc2258)

[1.实际问题的描述 6](#_Toc26113)

[2.数据结构的选择 7](#_Toc29949)

[3.算法的设计 7](#_Toc6430)

[4.类的设计 7](#_Toc11118)

[5.互动性设计 8](#_Toc25233)

[四、 研究结果并讨论 8](#_Toc29797)

[1.类的实现 9](#_Toc21735)

[2.运行结果 12](#_Toc32341)

[3.DFS的测试与性能 19](#_Toc14942)

[4.讨论 20](#_Toc20924)

[结论 22](#_Toc3046)

[一、主要工作 22](#_Toc29833)

[二、主要结果 22](#_Toc28015)

[三、心得感受 22](#_Toc16830)

[四、主要发现 23](#_Toc16762)

[五、下一步的打算 24](#_Toc24382)

[参考文献 25](#_Toc13881)

[三级项目小组工作量及成绩表自查 26](#_Toc19169)

# 前言

## 一、研究背景

在未来的物流管理系统建设过程中，势必对现有技术进行深刻改革，以更好地满足不断增长的物流需求。伴随着科技的不断进步，一系列新技术将在物流管理系统中得到广泛应用，从而显著提升整体效率。这些技术的改革和研究努力将有助于塑造一套适应未来物流挑战的联合作战指挥协同模式，从而提高管理效率。

物流管理系统的信息化是融合信息技术的发展以及不断完善物流系统的一个重要过程；是信息技术与物流运作相融合，从而构建一个信息化水平较高的物流系统的关键步骤；同时，借助信息化的力量，各个物流环节得以有机结合，创造出更为高效的物流运作体系。

因此，对物流信息的整合和处理显得更加重要，尤其在物流领域，随着市场的快速变化，系统需要能够迅速适应和响应。在这个瞬息万变的物流领域，通过先进的信息化建设，可以更好地应对挑战，提高物流运营的灵活性和反应速度。

本小组取材于生活，通过和对疫情后中国经济现状的思考，结合数据结构与算法的课程感悟，提出了基于有向有权图的货运物流管理系统的课题。希望在项目进程中能不断加深对数据结构的理解、并在将来为国家经济发展战略贡献自己的力量。

## 二、研究目的

### 1.目的

### 我们的项目是物流管理系统，旨在实现对货物运输过程中的位置和运输队伍的相关信息进行高效分析。通过利用地图各点和路径的数据，为物流管理方提供实用信息，简化物流车辆路线、安排等问题，最终达到高效物流管理的目标。

### 2.范围

（1）通过小组成员得了解确定该信息系统需要具备的功能，以便确定大致的方向，以及准备相关的材料。

（2）通过网络了解相关方面的研究以及目前的已有的相关的基础。

## 三、预期结果

预期实现对物流运输网络地点和线路数据的合理收集处理，得出用户想要的信息。最后能够给用户提供相应的功能，以及提供用户友好访问界面。

## 四、项目组分工

：负责基础类的编写、以及继承类的总体架构、拓扑排序和A\*算法的实现。负责项目的总体设计和代码的整合、负责一部分三级项目报告的编写。

：负责迪杰斯特拉算法和佛洛依德算法的实现负责迭代器的实现，负责一部分三级项目报告的编写，负责一部分ppt的制作。

：负责edge类和graph类的设计，负责DFS实现和交互实现，

负责一部分三级项目报告的编写，负责一部分ppt的制作

：负责node和nodeinfo类的实现，负责DFS性能比较，负责册数数据的测试以及程序的测试。

# 正文

## 一、 研究内容的基本原理

本项目需要存储地点以及地点之间的距离关系，所以我们选择了图的结构。考虑到图有顶点和边两部分，我们把点和边分开存储、分别采用栈和邻接数组的方式存储点和边。邻接数组里存的点带有的信息只有边的权值，使得该图的独立性提高，不会影响到点的其他信息。

## 二、 所采用的研究方法及相关工具

本程序以数据结构与算法实现的数据处理功能为中心，以ISO C++11标准进行开发，开发过程中组员各自使用自己熟悉的IDE（集成开发环境，VisualStudio2022、devc++、VSCode等），最终通过JetBrains CLion进行功能汇总和各模块代码整合。对控制台的开发完全由代码控制，本组通过打印的方式展示菜单等互动信息，使用者可以通过输入对应的编号来使用对应的功能。项目主要采用实证研究法，将代码编写完成后对其进行测试，观察是否符合预期来进行研究。

## 三、 项目的方案设计

### 1.实际问题的描述

我们项目所对应的实际问题是物流地图信息整合与处理，具体到我们的项目里就是对物流中地点路线的安排。我们计划解决主要的问题就是地点间最短路径的计算，这也包含了要求必须经过某点类似的特殊路径要求。对地点间的最短路径的计算，物流运输主要成本就是距离和耗油量、而具体化到城区内派送便是距离和派件送达时间。需要在不同的地点运输，所以地点间的行车的路线一定要是最短的才可以，这样就可以迅速地抵达目的地，满足物流系统要求。

### 2.数据结构的选择

数据结构选用的是邻接数组实现，邻接数组不会浪费空间而且相对容易访问到目的数据。

邻接数组中存储的是点与点之间的边的关系，同时有一个栈来存储点的相关信息。

### 3.算法的设计

（1）图的拓扑排序[1]的算法，通过拓扑排序可以将有先后顺序（紧急程度）的运输要求排好序。

（2）地点间最短路径，我们设计采用佛洛依德算法，佛洛依德算法可以给出每个点到其他点的最短路径，可以很好的解决问题。

（3）迪杰斯特拉算法，我们也通过该算法得出两个点之间的最短路径，与佛洛依德算法相结合，这个偏向于单纯两点间的最短路径计算。

（4）A\*寻路算法，可以根据后续点的更多属性信息进行启发式搜素，加快路径寻找。

### 4.类的设计

基于面向对象的理念，我们构建了5个类，同时还引入了一个基础的graph类。在这些类中，arrayWDgraph扮演着最为核心的角色，其余类均为其提供支持。有关图的功能和操作也主要集中在arrayWDgraph中。arrayWDgraph继承了graph这一抽象类，并成功实现了其中的方法，并额外进行了一些扩展，新增了其他方法。为了更便于后续的维护和修改，我们将程序的设计与实现进行了分离，将模板类函数以tpp文件的形式按照功能分类存储了具体实现。

最终呈现的数据结构UML图如下：

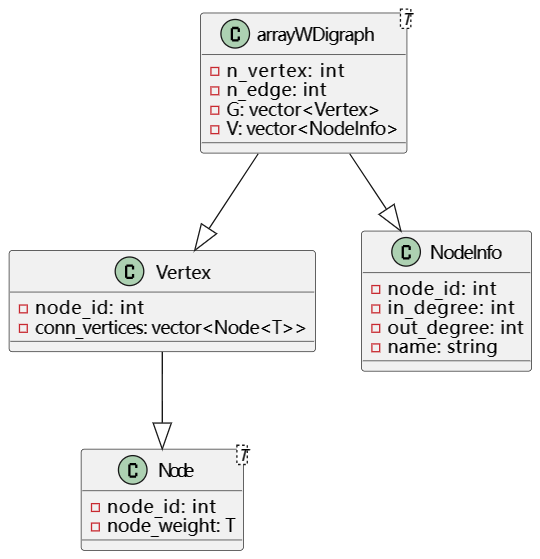


图1

### 5.互动性设计

通过多种方式实现与用户的互动，完成用户所选功能的同时，输出相应的展示。每个步骤都引导用户提供必要的信息，并展示处理后的结果。

## 四、 研究结果并讨论

### 1.类的实现

图中的节点由node类表示，边由edge类表示，迭代器由vertexIterator类表示，nodeinfo用于表示节点的另一种形式，graph是基类，而arrayWDgraph是继承自基类的有向有权图类。

（1）在类的实现中，node类仅包含点的编号和边的权值这两种信息。每个点存在邻接数组中，表示相应边。node类的方法仅包括获取编号和获取权值。

（2）nodeinfo类包括点的编号、名字、入度和出度等信息。除构造函数外没有其他方法，仅用作节点的抽象表示。

（3）edge类表示图中的边，包含起点、终点和权值等信息。提供的方法包括返回起点、返回终点、返回权值以及输出边的信息。

（4）在vertexIterator类的定义中，它被抽象为一个类，后续通过继承实现其中的方法。这些方法主要用于将游标移动到下一个位置。

（5）arrayWDgraph类是最关键的类，调用其他类。作为有向有权图的代表，它包含顶点数、边数、邻接数组以及点对应的栈等信息。

其中的方法可大致归为四类。第一类是基于图的基础方法，可被其他方法调用，用于获取点或边的相关信息。具体包括获取编号为x的点的索引、获取点的个数、获取边的个数、判断点x是否存在、判断边（a，b）是否存在、判断点x的合法性、判断边（a，b）的合法性、获取入度、获取出度、获取度、判断是否为有向图、判断是否为有权图、获取文件中第一部分字符串、获取文件中第二部分字符串、获取文件中第三部分字符串。

第二类是图修改的方法，包括增加点、删除点、增加边、删除边、更新权值、更新占领值、对100个点的初始化、对5个点的初始化以及清空图等操作。

第三类是与算法相关的方法，包括拓扑排序、佛洛依德算法、深度优先算法等。

第四类是方便用户互动的方法，为其他算法提供服务，包括打印路线、删除点和边、增加点和边等引导方法。在项目具体实现中，arrayWDgraph类函数定义接口设计和实现的文件拆分方式如下：

1. /// arrayWDigraph基础，包含基类函数、迭代器和实用子方法（实现于arrayWDigraph\_basic.tpp）
2. // 基类函数实现
3. **explicit** arrayWDigraph(**int** n\_vertex\_primary);
4. ~arrayWDigraph(){};
5. **int** numberOfVertices() **const** {**return** n\_vertex;}; // 获取点的个数
6. **int** numberOfEdges() **const** {**return** n\_edge;}; // 获取边的个数
7. **bool** existsEdge(**int** vertex\_in\_id, **int** vertex\_out\_id) **const**; // 判断是否存在边a->vertex\_out\_id
8. **void** insertEdge(edge<T>\* Edge); // 插入边
9. **bool** eraseEdge(**int** v1, **int** v2); // 删除边
10. **int** degree(**int** v) **const** {**return** indegree(v) + outdegree(v);}; // 获取度
11. **int** indegree(**int** v) **const** {**return** V[v].in\_degree;}; // 获取入度
12. **int** outdegree(**int** v) **const** {**return** V[v].out\_degree;}; // 获取出度
13. **bool** directed()  **const** { **return** **true**; }; // 是否为有向图
14. **bool** weighted()  **const** { **return** **true**; }; // 是否为有权图
16. // 实用基础子函数和图算法
17. **int** \_indegree(**int** x) {**return** indegree(getIndex(x));} // 获取入度子方法
18. **int** \_outdegree(**int** x) {**return** outdegree(getIndex(x));} // 获取出度子方法
19. **int** getIndex(**int** x); // 获取编号为x的点的索引
20. **bool** existVertex(**int** node\_id); // 判断是否已经存在点v
21. **bool** isEdgeLegal(**int** v1, **int** v2); // 判断边v1->v2是否合法
22. **bool** isVertexLegal(**int** node\_id); // 判断点v是否合法
23. **bool** isWeightLegal(T weight); // 判断权值是否合法
24. **void** updateWeight(**int** vertex\_in\_id, **int** vertex\_out\_id, T new\_weight); // 更新权值
25. **void** addVertexSingle(**int** node\_id, string node\_info\_name); // 增加点
26. **void** addVertex(); // 添加多个点
27. **void** addVertex(**int** x); // 添加单个点（重载函数）
28. **void** deleteVertexSingle(**int** node\_id); // 删除点
29. **void** deleteVertex(); // 删除多个地点
30. **void** addEdge(); // 添加路
31. **void** deleteEdge(); // 删除路
32. **void** connectedSets(); // 输出连通集
33. **void** outputG(); // 输出当前图所有边
34. **void** outputV(); // 输出当前图所有点
35. **void** clearAllFiles(); // 重置文件存储的信息
37. /// DFS（实现于arrayWDigraph\_dfs.tpp）
38. **void** rDFS(**int** start, **int** label);// 被DFS调用
39. **void** sDFS(**int** start, **int** label); //被dfs调用
40. **void** customizedDFS(**int** start, **int** label);//自带的迭代器dfs
41. **void** DFS(**int** start, **int** label);//课本的迭代器的DFS
42. **void** DFSPerformCompare(); // dfs性能测试
44. /// 文件操作（实现于arrayWDigraph\_file\_io.tpp）
45. T stringToT(**const** string& str); // 将string转化为T，用于从文件读取时使用
46. string getFirstPart(string s); // 返回第一部分的字符串
47. string getSecondPart(string s); // 返回第二部分的字符串
48. string getThirdPart(string s); // 返回第三部分的字符串
49. **int** locateFirstBlank(string s); // 返回第一个空格位置的字符串索引
50. **int** locateSecondBlank(string s); // 返回第二个空格位置的字符串索引
51. **void** updateEdgeWeight(**int** x, **int** y, T z); // 在文件中更新权值
52. **void** addVertexFile(**int** x, string y); // 在文件中写入点x
53. **void** deleteVertexFile(**int** x); // 删除点的文件操作
54. **void** addEdgeFile(**int** x, **int** y, T z); // 在文件中写入边x->y
55. **void** deleteEdgeFile(**int** x, **int** y); // 删除路的文件操作
57. /// 拓扑相关算法（实现于arrayWDigraph\_topo.tpp）
58. **void** \_topologicalSorting(); // 拓扑排序
59. **void** topoRank(); // 拓扑排序方法
61. /// 路径相关图算法（实现于arrayWDigraph\_path.tpp）
62. **void** floydGetPath(); // 最短路径方法，floyd前端函数
63. **void** \_floydPrintPath(); // 打印最短路径，依据\_floyd函数运行结果递归输出
64. **void** \_floyd\_PrintPath(**int** i, **int** j); // 打印最短路径子方法
65. **void** \_floyd(); // 佛洛依德算法
67. **void** dijkstraGetPath(); // 最短路径方法，dijkstra前端函数
68. **void** \_dijkstra(**int** start); // Dijkstra算法核心部分，计算了起始点到所有其他点的最短路径（单源）并保存到dis\_dijkstra数组中
69. T\* \_shortPathDijkstra(**int** start, **int** end); // 算法中间层，初始化必要数据，异常捕获，预处理路径细节数据等
70. **void** \_dijkstraResult(**int** start, **int** end); // 此函数仅取目标终点的最短路径并输出
72. T \_aStarHeuristic(**int** current, **int** goal); // 启发函数
73. vector<**int**> \_aStar(**int** start, **int** end); // 核心算法
74. **void** AStarGetPath(); // A\*寻路 前端函数
76. **void** \_shortPathVia(**int** start, **int** through, **int** end);
77. **void** shortPathVia(); // 两点间且通过某点的最短路径
79. /// 初始化相关函数（实现于arrayWDigraph\_init.tpp）
80. **void** clearProgramData(); // 清空图
81. **void** autoInitFromFile1(); // 文件初始化1
82. **void** autoInitFromFile2(); // 文件初始化2
83. **void** init(); // 初始化

### 2.运行结果

**（1）项目的主界面**

主界面上有18个功能可供选择，如下图所示。

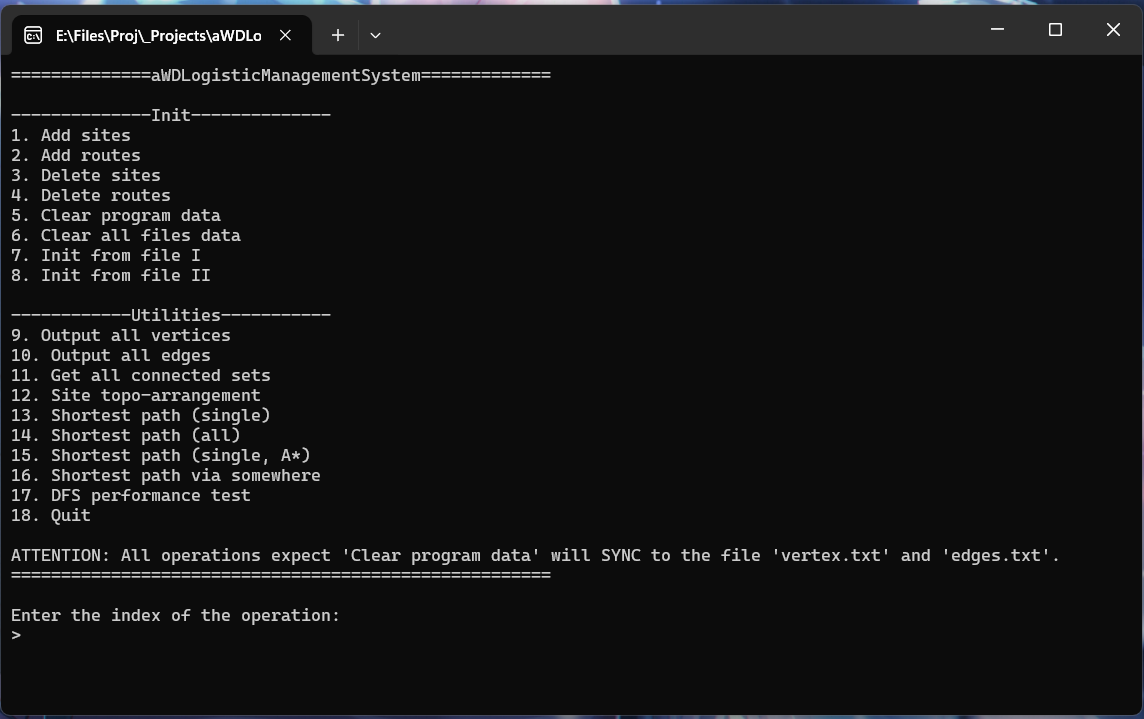


图2

**（2）读取文件初始化**

在图的初始化阶段，必须先检查图是否已经存在。若已存在，则无需再次创建，可以选择清空原有图，然后重新创建新图。为了方便调试，我们设计了两个初始化选项，以选择不同的文件进行初始化。同时，我们还设计了相关的方法，用于清空文件数据等操作。从以数据为中心的视角考虑，我们可以采用SQLite等内嵌数据库，并结合命令行程序，实现数据的长期存储和完善的数据访问控制。

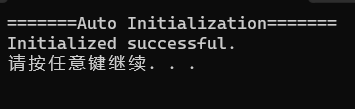


图3

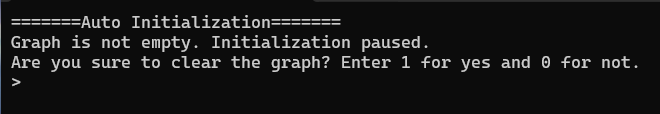


图4

**（3）对图的更改操作**

对图的更改中同时也会影响到文件中的数据，在下一次使用的时候也会使用到。在更改过程中也会对输入数据进行检验，判断更改的是否合法，是否重复，只有符合标准的才可以更改，否则不允许更改。如图所示。

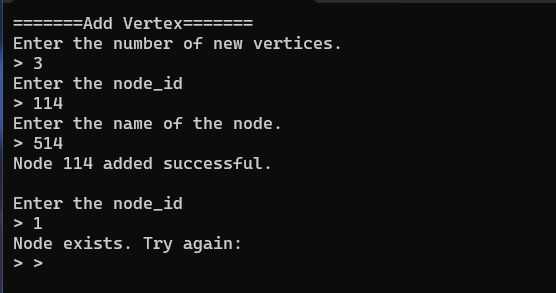


图5

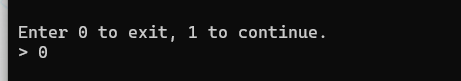


图6

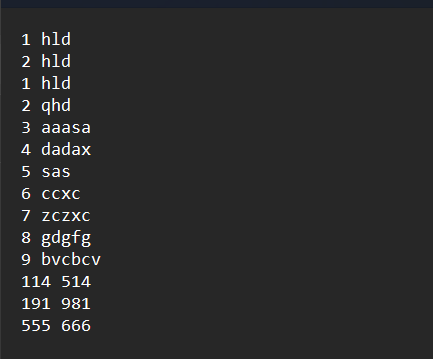


图7

可见文本中的数据也被同步正确更改了。

**（4）每个点和边的情况输出和连通性测试**

对点和边的情况的反馈和更改，能够检查是否存在地点x的信息等，以供信息核验校对。连通集也能便利地检测一组区域地点的联通情况，从而方便地进行物流运输安排。

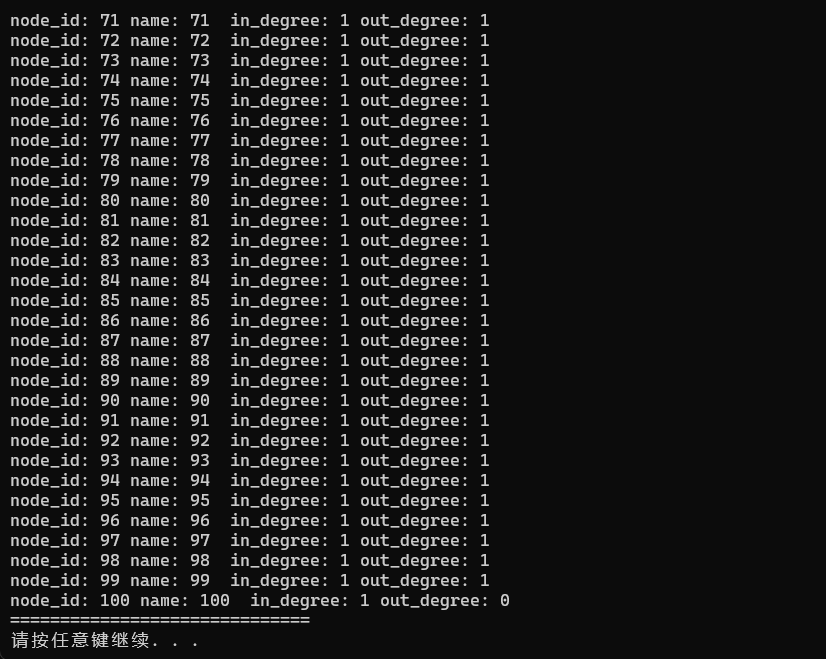


图8



图9

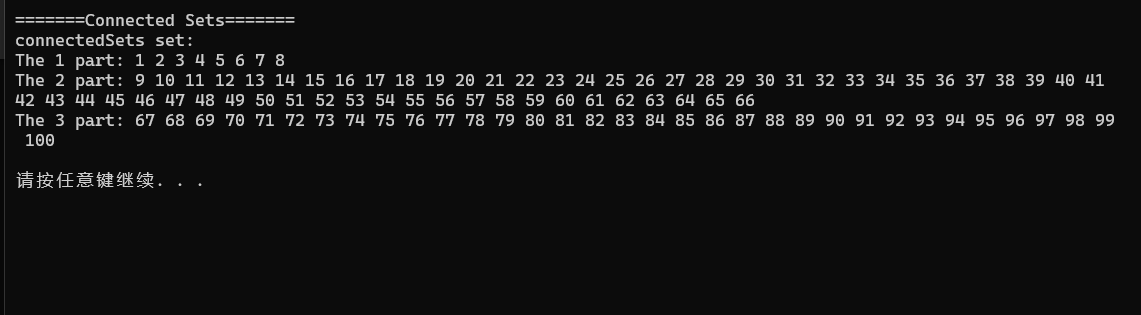
****

图10

**（5）拓扑排序**

按照图示，拓扑排序可依据预先创建的图执行。同时，会对读入的数据进行条件判断，只有在满足条件的情况下才进行拓扑排序。某些图可能因无法通过拓扑排序得出结果而产生提示，表明无法获得结果（见下图）。

图11

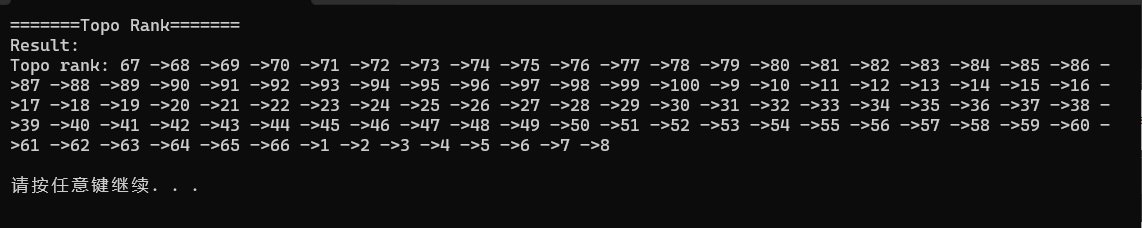


图12

**（6）佛洛依德算法**

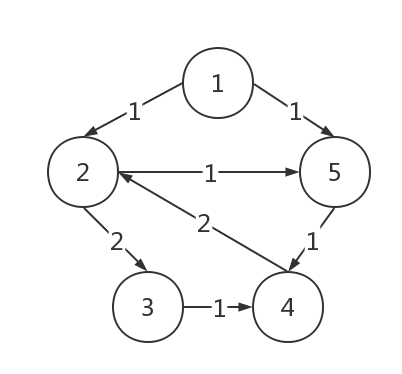
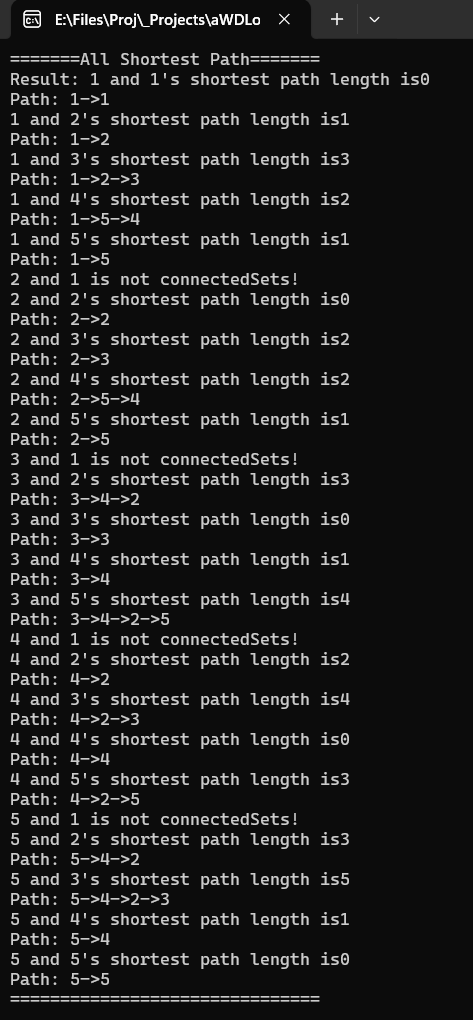
此处的最短路径是初始化5个点（如下图）后得出的最短路径，可以分别得出每个点到其他各点的最短路径。

图13

**（7）单点最短路径**

该功能使用迪杰斯特拉算法或A\*，计算某个点为起点到另一个点的最短路径，并且输出。仅仅得出目标路径，不会输出其他的路径。

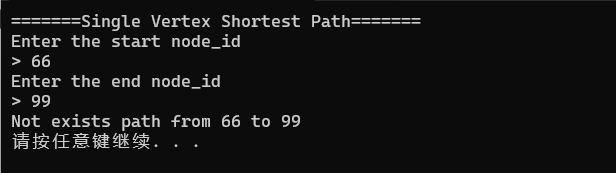


图14

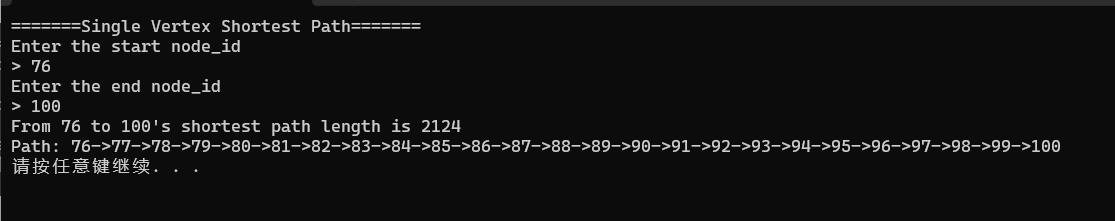


图15

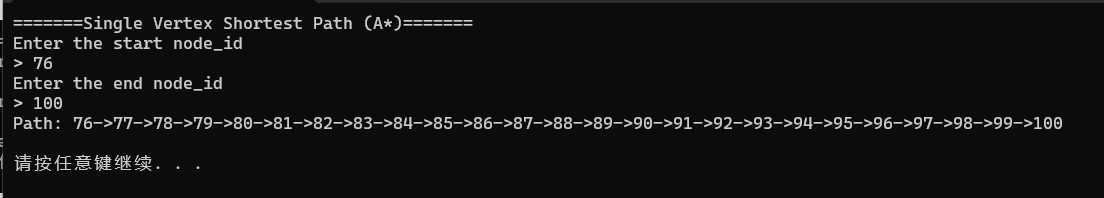


图16

**（8）性能测试**

这里是对课本上的深度优先算法和自己编写的深度优先算法的性能的比较，通过对时间的记录，来比较两个算法的时间快慢，从而得出哪个算法性能更好。

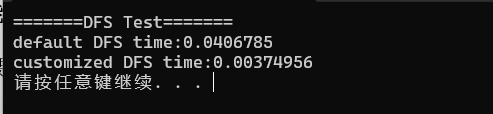


图17

### 3.DFS的测试与性能

**（1）测试**

对于DFS的测试采用了白盒法，其中数据确保DFS的每条语句至少被执行一次。相应的数据只需保证是一个连通图，就可以确保每条语句至少执行一次。测试结果表明，代码的正确性毫无问题，获得的结果也与预期一致。

**（2）性能**

|  |  |  |  |
| --- | --- | --- | --- |
| 数据量n | 课本DFS时间（ms） | 定制DFS时间（ms） | 差值（课本-定制）（ms） |
| 5 | 0.0138622 | 0.00784345 | -0.01198125 |
| 10 | 0.0183705 | 0.0290301 | -0.0106596 |
| 50 | 0.1879511 | 0.0918964 | 0.0960547 |
| 100 | 0.414025 | 0.195434 | 0.218591 |
| 200 | 1.09989 | 0.367477 | 0.732413 |
| 500 | 4.60896 | 0.93633 | 3.67263 |
| 1000 | 15.4769 | 1.84532 | 13.6316 |
| 2000 | 53.5456 | 4.04722 | 49.4984 |

表1

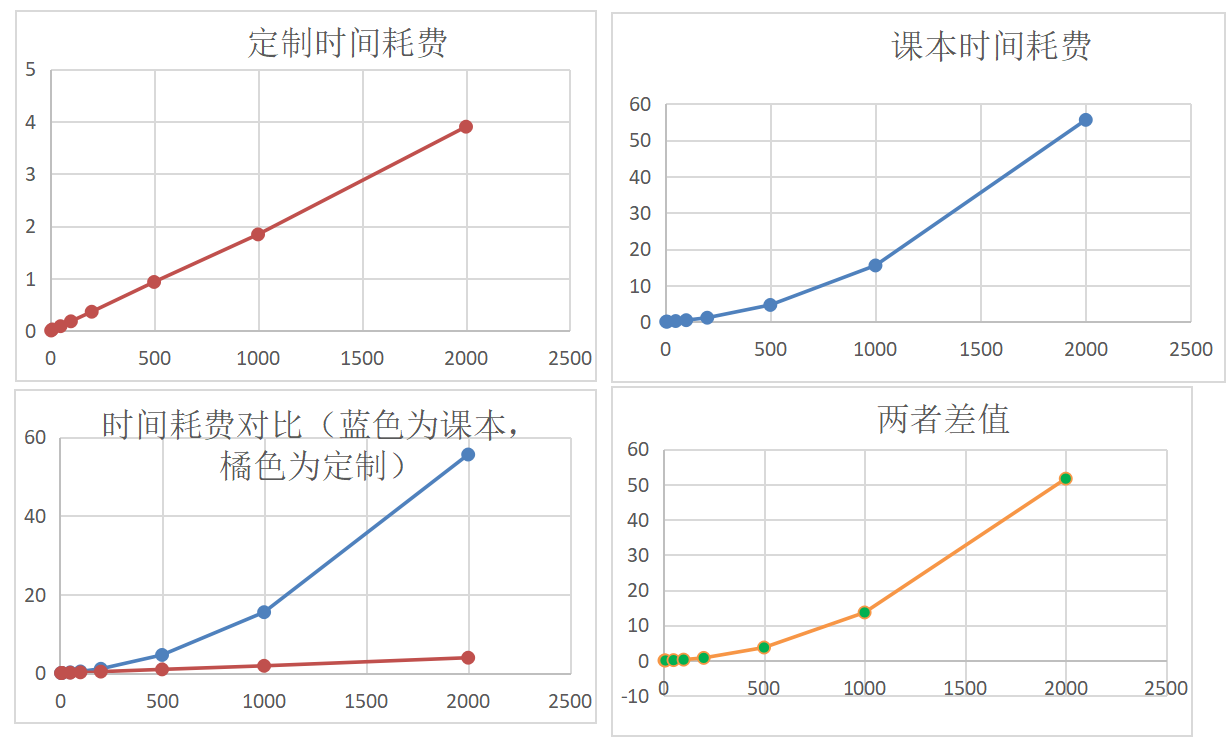
通过对8次不同数据量的测试，我们比较了两个DFS的性能。发现当数据量较小时，它们的时间耗费差异不大，但当数据量超过200后，两者的时间耗费逐渐拉开。然而当数据量达到1000时，定制的DFS的时间效率是课本DFS的8倍多。随着数据量的进一步增加，课本DFS的效率逐渐下降，因为定制的DFS的时间复杂度为O(E+N)，而课本DFS执行的代码更为简洁，因此其相应的时间耗费较少。

图18

### 4.讨论

（1）本项目中的邻接数组采用的具体的形式？该形式有什么意义？

在本项目中，邻接数组采用了一种特定的形式。这种形式是基于结构体类型的vector容器，其中结构体包含一个int类型的数据和一个node类型的vector容器。这种设计充分利用了空间，相较于邻接矩阵避免了空间上的浪费。与邻接链表相比，该形式可以通过下标方便地访问指定位置的数据。结构体中的int类型数据代表边的起点编号，而node类型中存储的是终点的编号。这样清晰地表示了边的起点和终点，而且在点的删除和增加过程中不会因为点及其编号的变化而无法正确识别相应的点。

（2）本项目最短路径问题如何实现？

本项目通过两种算法解决最短路径问题，分别解决了单源最短路径和多源最短路径问题。使用弗洛伊德算法可以打印出所有点每个点到其余点的最短路径。这种方法代码更为简洁，可让军队指挥官清晰了解全局最短路径策略。

另一方面，使用迪杰斯特拉算法可以打印出目标起点和目标终点之间的最短路径。这种方法时间复杂度较小，对于已知起点与终点的最短路径的求解更为高效，有助于士兵更清晰地了解目的地的最短路径。

此外，探索了A\*寻路算法，可以打印出目标起点和目标终点之间的最短路径，并可根据某些信息有选择性地扩展节点（通过启发函数实现）。

# 结论

## 一、主要工作

## 本次项目的主要任务是编写基类及其相关方法。相对于算法部分，基类部分的代码量较大，其中包括许多小方法，用于检查用户输入的合法性以及与当前图做对比查重等。此外，代码的合并过程也涉及较大的工作量，包括变量和函数名的重构等。算法部分涵盖了拓扑排序、弗洛伊德算法、迪杰斯特拉算法、A\*算法和深度优先搜索算法等。

## 二、主要结果

本项目主要是对有向有权图的一些操作及算法，得出的成果也是具有一定意义。实现了预期的结果，能够流畅地实现用户需要的功能。

## 三、心得感受

首先，该项目拓展了我们的知识面，为我们提供了更广阔的视野。在项目开发过程中，我们深入了解了我们所学知识的实际应用。虽然项目中包含许多小方法，这些小方法看似微不足道，但实际上其他方法都是基于这些小方法的基础上实现的。因此，小方法在整体项目中也显得非常重要。如果小方法出现错误，可能导致调用它的其他算法也出现错误。因此，在项目开发中，全局视野非常重要，不能只关注局部，否则整个项目可能会混乱。

在项目中，我们深入了解了深度优先遍历的运行过程和实现原理。通过运用DFS进行性能测试，比较了内存和运行时间前后的变化，还能够判断一个图是否连通。

在基本架构的基础上，我们了解了实现最短路径和队伍调度的方法，发现了弗洛伊德算法和迪杰斯特拉算法两种实现最短路径的方法的相同和不同之处。根据不同之处，我们能够解决不同情况、不同角色的实际问题。在编写代码时，我们通过编写相同的变量，尽可能减少空间申请，简洁地解决问题。

在迪杰斯特拉算法的实现过程中，我们遇到了一些困难。一开始，由于对邻接数组的理解不够深入，再加上对于使用vector嵌套结构体vector的储存方式不够熟悉，导致了一些问题。通过多次的尝试和调试，加之团队合作的协助，最终克服了这些困难，成功输出了正确结果。这也强调了团队合作在解决问题中的重要性。

## 四、主要发现

1. 在处理复杂项目时，一定需要先制定整体架构和计划，这能帮我们明确目标并避免在开发过程中出现不知道接下来要做什么的情况。软件工程的原则在大型项目中尤为重要，有助于项目的组织和管理。

2. 算法万变不离其宗，对于算法的编写要以课本为主，同时加强对算法本身思想的理解，以及与数据结构结合时算法核心思想的不变性以及算法书写的可变性。

3. 提高自主学习和知识拓展的能力是提升计算机综合能力的关键。在三级项目中学习新知识，比较不同算法的优劣，深入理解迭代器和DFS等概念，都是我们培养这种能力的有效方式。

4. 断点调试和自查自改的能力对于解决问题至关重要。在面对错误和混乱时，我们需要耐心地使用断点调试，逐步查找和修正错误，对于项目的顺利进行起到了关键作用。坚持不懈地解决问题，这可以我们帮助提高编程的质量和效率。

## 五、下一步的打算

- 基于基本函数开发更多上层功能，实现更友好的用户接口

- 开发数据库系统，实现数据的持久化存储和完善的日志记录

- 开发图形化界面（例如Qt框架）、或以此思路开发Web应用，实现跨平台部署

下一部我们打算基于当前的成果继续拓展，研究新的物流运输方面的问题，并给出解决，不断完善我们的项目，最后能够真正成为一个成熟的项目。

# 参考文献

[1]王顺凤.一种有向权图的拓扑排序算法及其应用 [J].南京:南京气象学院,2002.

# 三级项目小组工作量及成绩表自查

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 所做的工作 | 成绩 | 签字确认 |
|  | 负责基础类的编写、以及继承类的总体架构、拓扑排序和A\*算法的实现。负责项目的总体设计和代码的整合、负责一部分三级项目报告的编写、负责汇报ppt的制作。 |  |  |
|  | 负责迪杰斯特拉算法和佛洛依德算法的实现负责迭代器的实现，负责一部分三级项目报告的编写，负责一部分ppt的制作。 |  |  |
|  | 负责edge类和graph类的设计，负责DFS实现和交互实现，负责一部分三级项目报告的编写，负责一部分ppt的制作 |  |  |
|  | 负责node和nodeinfo类的实现，负责DFS性能比较，负责册数数据的测试以及程序的测试。 |  |  |