

**《数据结构与算法》课内实验报告**

**姓名： 林奕宏**

**学号： 3123004449**

**专业： 软件工程**

**班级： 2班**

**指导教师： 刘添添**

**完成日期： 2024年10月19日**

目录

[一、如何阅读本文档： 3](#_Toc180259225)

[二、课内实验说明： 3](#_Toc180259226)

[三、数据结构实现完成度： 3](#_Toc180259227)

[四、实验内容特色概要： 4](#_Toc180259228)

[五、实验实现思路： 4](#_Toc180259229)

[六、类和接口的封装定义： 5](#_Toc180259230)

[顶点类定义： 5](#_Toc180259231)

[边类定义： 6](#_Toc180259232)

[图的抽象类定义： 6](#_Toc180259233)

[迭代器类的定义： 7](#_Toc180259234)

[图的具体实现类定义： 8](#_Toc180259235)

[七、重要代码具体实现思路： 9](#_Toc180259236)

[1. 图的默认存储结构边列表的存储： 9](#_Toc180259237)

[2. 顶点的删除（使用首尾数据交换法）： 10](#_Toc180259238)

[3. 邻接矩阵存储结构的生成 11](#_Toc180259239)

[4. 邻接表存储结构的生成 12](#_Toc180259240)

[5. 十字链表存储结构的生成 12](#_Toc180259241)

[6. 广度优先搜索 14](#_Toc180259242)

[7. 深度优先搜索 15](#_Toc180259243)

[8. 查找路径 16](#_Toc180259244)

[九、测试： 17](#_Toc180259245)

[十、补充说明： 17](#_Toc180259246)

[1. 如何实现非加权图？ 17](#_Toc180259247)

[2. 如何实现无向图？ 17](#_Toc180259248)

[3. 本次实验使用的默认存储结构？ 17](#_Toc180259249)

[4. 实验文件结构？ 17](#_Toc180259250)

[5. 如果生成可执行文件？ 18](#_Toc180259251)

# 一、如何阅读本文档：

1. 推荐根据目录内容从上往下依次阅读

2. 如果遇到代码截图不清晰可在doc文件夹中查看相应的png图片或者查看源代码

3. 推荐使用vscode的utf-8编码查看代码，防止中文乱码

# 二、课内实验说明：

本次课内实验选题为图

使用C++进行描述,使用VSCode完成编写

代码文件编码格式为UTF-8, 编译器使用Mingw-64 （同时支持VS环境下编译时的GBK编码）

文件内容为：

头文件： Graph.hpp (存储与图相关的类的声明)

源文件： Comment.cpp（存储图的实现所需要的相关辅助函数和辅助数据结构）

Exception.cpp （自定义异常类）

Graph.cpp (图类内各种方法的具体实现代码)

Test.cpp (作为演示和测试程序，用于检验整个数据结构-图的运行情况)

# 三、数据结构实现完成度：

1. 支持的图类型（4）：（加权\非加权）（有向\无向图）
2. 支持的数据存储结构： a. 邻接矩阵 b. 邻接表 c. 边列表. d. 十字链表
3. 支持常见的图的基本操作：
   1. 获取顶点数量
   2. 获取边的数量
   3. 判断顶点是否存在图中
   4. 添加顶点到图中
   5. 删除顶点
   6. 插入边
   7. 删除边
   8. 计算度，出度，入度
   9. 判断边是否存在与某两个顶点
   10. 打印图
   11. 清空图
   12. 设置某个顶点的数据
   13. 设置某条边的权重
   14. 获取某个顶点的数据
   15. 获取某条边的权重
   16. 判断顶点索引是否合法
   17. 清空图
4. 支持的图的存储结构的操作：
   1. 打印边列表
   2. 生成邻接矩阵
   3. 删除邻接矩阵
   4. 打印邻接矩阵
   5. 返回邻接矩阵
   6. 生成邻接表
   7. 删除邻接表
   8. 打印邻接表
   9. 生成十字链表
   10. 删除十字链表
   11. 打印十字链表
5. 支持的图的遍历算法：
   1. 广度优先搜索
   2. 深度优先搜索
   3. 图的路径查找
6. 支持迭代器

# 四、实验内容特色概要：

1. 将图的数据中的各种组成部分进行封装，并手动实现各种数据结构的封装，避免了对于常见STL的依赖（比如支持存储任意数据类型的可变长数组以及队列）

2. 对于图的顶点的数据域：支持使用多种数据类型，包括但不限于常见的基本数据类型、对象等等

3. 对于图的边的权重：支持使用多种数据类型，包括但不限于常见的基本数据类型如int, float, double等等

4. 支持动态调整顶点列表使得在任意删除顶点的情况下保持邻接矩阵和邻接表的顶点编号连续性

5. 封装自定义的异常类，实现多种异常情况的抛出与捕获

6. 支持UTF-8和GBK编码下中文的正确输出显示

# 五、实验实现思路：

1. 对抽象数据结构图进行封装，使其成为一个抽象数据类型Graph

2. 对图所需的顶点和边进行封装，使其成为一个能够适应更多丰富的数据类型的存储与使用的类

3. 对有向加权图进行封装，重写抽象数据类型Graph并添加常用的基本操作和相关的算法

4. 增加常见的图的存储结构，如邻接矩阵，邻接表，边列表和十字链表

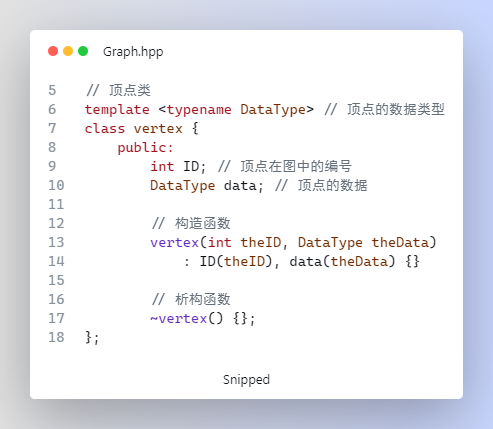
5. 增加各种存储结构相关的操作

6. 编写测试代码，实现对图的增删改查和算法实现的检测

7. 编写构建脚本，添加静态链接编译支持，避免ddl缺失

# 六、类和接口的封装定义：

## 顶点类定义：



每个顶点拥有唯一的ID以及data数据域

## 边类定义：



每条边拥有起始顶点ID、终止顶点ID和该边的权重

## 图的抽象类定义：



将图的各种基本操作封装成虚函数，默认边列表为存储结构，包含图中的顶点数和边数，还有没有边时的标志，以及图中所有顶点构成的顶点列表

## 迭代器类的定义：



作为某个顶点的迭代器，并且能够返回与当前顶点的邻接的下一个顶点

## 图的具体实现类定义：



对图进行具体实现，重写抽象数据结构的各种基本操作，并增加各种操作和算法

# 七、重要代码具体实现思路：

## 1. 图的默认存储结构边列表的存储：

a. 创建一个边对象（初始化其起始ID和终止ID以及边的权重）

b. 检查边是否以及存储

c. 如果不存在则插入到边列表数组中



## 2. 顶点的删除（使用首尾数据交换法）：

a. 获取顶点列表末尾的顶点对象地址

b. 使用末尾顶点的数据替换待删除顶点的数据

c. 从顶点列表中删除末尾顶点

d. 删除与被删顶点相关的边

e. 更新与原先末尾顶点有关的边的起始以及终止ID



## 3. 邻接矩阵存储结构的生成

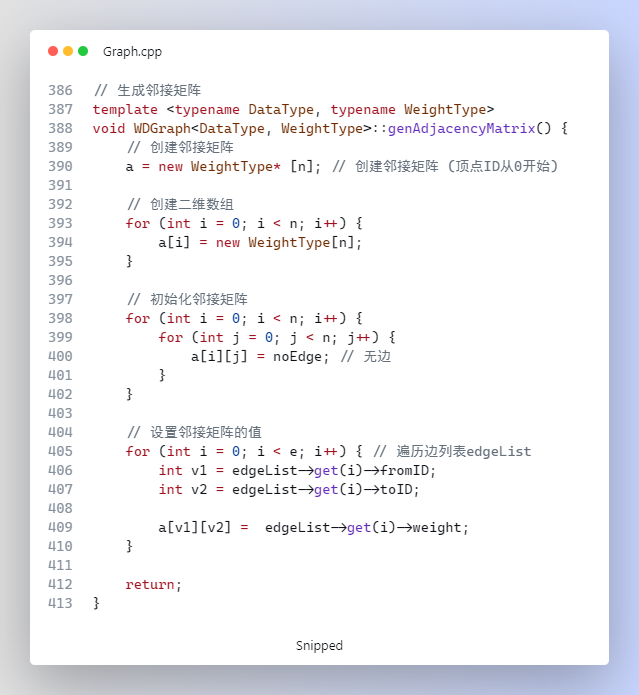
a. 创建数据类型为边的权重类型相同类型的一维数组

b. 一维数组每个位置的index对应顶点列表中顶点的ID编号

c. 对数组中每个位置生成一组一维数组

d. 将构成的矩阵的值都设为noEdge

e. 遍历边列表将每条边的信息设置到邻接矩阵中相应的位置



## 4. 邻接表存储结构的生成

a. 将邻接表结点对象初始化成一组一维数组

b. 再依次对以为数组中的每个结点值设为编号ID与index相同的结点信息

c. 遍历边列表，对获取到每一条边的信息，根据边的信息构建邻接表结点

d. 将构建的每一个邻接表结点插入到相应的位置



## 5. 十字链表存储结构的生成

a. 判断顶点列表和边列表是否为空, 不为空则继续，否则抛出异常

b. 创建长度为n的顶点数组（类型为VexNode\*）

c. 初始化顶点数组， 将数组中每个位置创建一个对应的VexNode对象

d. 遍历边列表，获取每条边的起始和终止顶点，并创建十字链表相应的边结点ArcBox

e. 分别将每条边得到的起始顶点插入到相应顶点的出边链表

f. 分别将每条边得到的终止顶点插入到相应顶点的入边链表



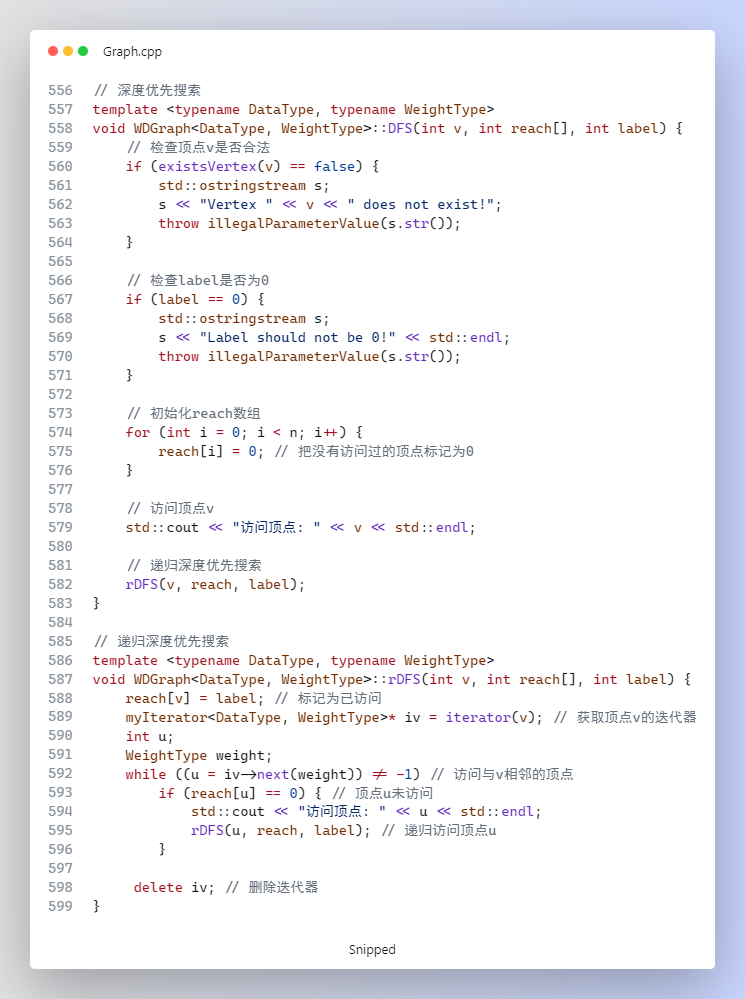
## 广度优先搜索

1. 传入一个reach[]数组，用于标记顶点是否被遍历过，传入int label用来标记顶点
2. 传入一个顶点v作为起始遍历顶点并检查其合法性
3. 初始化reach数组
4. 创建一个队列q
5. 把起始顶点v放入队列，并在reach数组中标记为已经访问
6. 使用while循环当队列不为空时，将队头出队，然后利用迭代获取到当前顶点的邻接顶点
7. 若该顶点未被访问过，则入队并标记为已访问
8. 删除掉当前迭代器，反复执行循环知道队列为空时while结束



## 深度优先搜索

1. 传入reach[]数组用于标记顶点是否被访问过，传入int label用来标记访问过的顶点
2. 初始化reach[]数组，获取传入的起始顶点v的迭代器
3. 执行while循环直到迭代器无法获取到邻接顶点为止
4. While循环的内容为对每次得到的邻接顶点判断是否未访问，如果未访问过则在reach[]数组中标记为已访问，然后递归深度优先搜索访问当前得到的顶点



## 查找路径

1. 函数返回一个从源头到终点的路径数组，若无法到达则返回空指针nullptr
2. Path[]数组的0号位存储路径长度，从1开始存储路径
3. 创建reach数组和path数组
4. 搜索路径，从顶点s开始实施深度优先搜索
5. 顶点s不应该等于终点
6. 当且仅当一条路径找到了，返回true
7. 每次递归标记reach[s]为已经访问，并创建顶点s的迭代器
8. 若迭代器能获取到顶点s的邻接顶点，则判断该邻接顶点是否已经访问过
9. 若未访问过，则将其加入路径，然后继续把得到的邻接顶点传入函数作为下一次递归的顶点s
10. 如果从邻接顶点到终点没有路径则从路径中删除邻接顶点并返回false



# 九、测试：

执行构建完成的二进制文件即可查看运行结果。

Test.cpp已包含本次实验内容所有接口和数据结构的测试

# 十、补充说明：

## 如何实现非加权图？

想要实现非加权图，只需另每条边的权重为相同的权重，如另每条边的权重为1即可将加权图转变为非加权图

## 如何实现无向图？

利用双向边模拟无向边，及也就是当两个顶点之间存在一条无向边时，只需利用两条权重相同，方向相反的边即可模拟无向边。

## 本次实验使用的默认存储结构？

本次实验使用的默认存储结构为边列表，其余的存储结构可利用相应的构造函数进行生成

## 实验文件结构？

├─doc

│ 课内实验文档.docx // 本次实验内容的文档

│

├─release

│ main.exe // 根据测试文件test.cpp构建生成的二进制可执行文件

│

└─source

build.sh // 源码构建脚本

Common.cpp // 存放各种辅助数据结构和函数

Exception.cpp // 自定义异常类

Graph.cpp // 图的主要源代码

Graph.hpp // 与图相关的各种类的定义与接口声明的源文件

Test.cpp // 实验数据结构验证与测试源文件

## 如果生成可执行文件？

执行source文件夹下面的build.sh脚本即可完成构建，编译完成的二进制文件存放在release文件夹中