数据结构课程第三次实验

1 知识回顾

在开始本次实验之前，先简单回顾一下课程中第六章（二叉树）和第七章（图）的相关内容。

1.1 二叉树

基本概念 二叉树是一种树形数据结构，其中每个节点最多有两个子节点，分别称为左子节点和右子节点。二叉树的根节点是树的起始节点。

常见类型

1. 满二叉树：每个节点要么是叶子节点，要么有两个子节点。
2. 完全二叉树：除最后一层外，所有层的节点都是满的，且最后一层的节点尽可能地左对齐。
3. 平衡二叉树：任意节点的左右子树高度差不超过 1 的二叉树，如 AVL 树。
4. 二叉搜索树（BST）：左子节点的值小于父节点的值，右子节点的值大于父节点的值。

基本操作

1. 插入：将新节点插入到二叉树中。
2. 删除：从二叉树中删除指定节点。
3. 查找：在二叉树中查找特定值的节点。

遍历方法

1. 前序遍历（Pre-order Traversal）：根节点 -> 左子树 -> 右子树
2. 中序遍历（In-order Traversal）：左子树 -> 根节点 -> 右子树
3. 后序遍历（Post-order Traversal）：左子树 -> 右子树 -> 根节点
4. 层次遍历（Level-order Traversal）：按层从上到下，从左到右遍历

1.2 图

基本概念 图是一组顶点和边的集合。图分为有向图和无向图。图可以表示各种关系，如社交网络、路线图等。

常见类型

1. 无向图：图中的边没有方向，边 (u, v) 和 (v, u) 是相同的。
2. 有向图：图中的边有方向，边 (u, v) 和 (v, u) 是不同的。
3. 加权图：图中的边带有权重，表示从一个顶点到另一个顶点的代价或距离。

基本操作

1. 添加顶点：在图中添加新顶点。
2. 添加边：在图中添加新边。
3. 删除顶点：从图中删除指定顶点。
4. 删除边：从图中删除指定边。

遍历方法

1. 深度优先搜索（DFS）：从起始顶点开始，尽可能深入每个分支。
2. 广度优先搜索（BFS）：从起始顶点开始，按层次遍历每个顶点。

2 实验内容

本次实验内容较多，主要分为两大部分：

2.1 二叉树

二叉树部分的实验代码的组织结构说明如下：

```
BinaryTree
├── CMakeLists.txt
├── include
│   ├── BinaryTree.h
│   ├── ExpressionBinaryTree.h
│   └── HuffmanTree.h
└── src
    ├── BinaryTree.cpp
    ├── ExpressionBinaryTree.cpp
    ├── HuffmanTree.cpp
    └── main.cpp
```

2.1.1 BinaryTree.h

在该头文件下，定义了基础的二叉树类。其基本的数据成员包括：

1. left: 左孩子;
2. right: 右孩子;
3. name: 节点名字标识;

其成员函数包括：

1. 四种对树进行遍历的操作函数；
2. 根据中序与另外一种序列（先序或后序任意一种）重构二叉树的构建函数；

2.1.2 ExpressionTree.h

在该头文件下，定义了继承自 BinaryTreeNode 类的子类 ExpressionTreeNode。
其新增的数据成员为 value（用于表示数字常量或运算符）。
其新增的成员函数包括：

1. 根据前缀、中缀、后缀表达式对表达式树进行构建的构建函数；
2. 获取算符优先级的函数；
3. 计算表达式树结果的计算函数；

2.1.3 HuffmanTree.h

在该头文件下，定义了继承自 BinaryTreeNode 类的子类 HuffmanTreeNode。
其新增的数据成员包含：

1. character：字符
2. frequency: 该字符出现的频率

新增的成员函数包括：

1. 哈夫曼树的构建函数；
2. 获取哈夫曼编码的函数；
3. 用于优先队列进行维护的重载比较运算符；

该部分实验需要补全的代码包括：

1. 基类 BinaryTreeNode 四种遍历方式的实现（即先序遍历、中序遍历、后序遍历和层次遍历）；
2. 基类 BinaryTreeNode 基于两种遍历方式的结果进行二叉树的重构；
3. 继承类 ExpressionTreeNode 基于两种表达式（前缀、后缀）进行表达式树的构建；

4. 继承类 HuffmanTreeNode 基于给定频率集实现 Huffman 树的构建；

5. 继承类 HuffmanTreeNode 基于先序遍历输出频率集字符的编码方式；

需要进行代码补全的部分位于对应的 BinaryTree/src/xxx.cpp 文件里，其中的 TODO 部分需要大家进行实现，相应的实现思路已经通过注释的方式放置在对应需要实现的函数里。

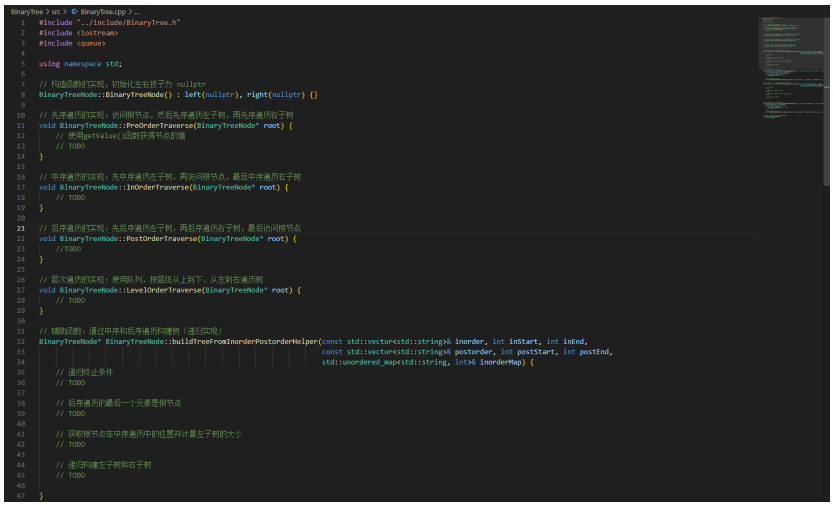

图 1: 需要填补的代码示例

在实现完这些.cpp 文件的内容后，大家可以根据自身的需要修改 BinaryTree/src/main.cpp 文件里的测试用例，以保证编写程序的健壮性。最终的实验检查中，助教会通过修改该文件的测试样例，将运行结果与标准结果进行比对，因此请充分检查自身程序的运行结果！

2.2 图

图部分的实验代码组织结构说明如下：

```
graph
├── CMakeLists.txt
├── include
│   ├── dijkstra.h
│   └── graph.h
└── src
    ├── dijkstra.cpp
    ├── graph.cpp
    └── main.cpp
```

2.2.1 dijkstra.h

该头文件下，定义了 dijkstra 类。其基本的数据成员包括：

1. g：图；

2. struct Vertex：记录最短路径信息的结构体，结构体成员有：
   (a) sure：最短路径是否确定的标志位；
   (b) path：记录最短路径的前驱节点的标号；
   (c) dist：记录该点到源点最短路径的长度。
3. vertex[N]：记录该图所有顶点到达源点最短路径信息的数组。

其成员函数有：

1. 求解最短路径的函数；
2. 输出最短路径的函数。

2.2.2 graph.h

在头文件下，定义了 graph 类。其基本的数据成员包括：

1. vertex_num：顶点数；
2. edge_num：边数；
3. struct Edge：记录边信息的结构体，结构体成员有：
   (a) adj：邻接点；
   (b) weight：边的权值；
   (c) next：某节点相连的所有边构成一条链表，该 next 域指向当前节点的下一条边。
4. struct Vertex：记录顶点信息的结构体（注意：和 dijkstra.h 头文件里定义的同名类型 struct Vertex 不一样），其成员包括：
   (a) head：与该节点相连的所有边构成的链表的头节点；
   (b) visited：是否被访问过的标志位，可用于深度优先搜索。

   其成员函数包括：
   (a) 深度优先遍历函数；
   (b) 给定节点、边和权值，完成图的构建函数。

该部分实验需要补全的代码包括：

1. 类 dijkstra 中以 s 为起始点的 dijkstra 算法求最短路径的函数；
2. 类 dijkstra 中输出最短路径的辅助函数；
3. 类 graph 中根据输入进行图的构建的函数；
4. 类 graph 中深度优先搜索的函数。

需要进行代码补全的部分位于对应的 graph/src/xxx.cpp 文件里，其中的 TODO 部分需要大家进行实现。

图 2: 需要补全的代码示例

3 其它说明

3.1 HuffmanTree.cpp 中的 priority_queue 容器说明

priority_queue 是一种特殊的队列，其元素按优先级排列。它的原型为：

```
std::priority_queue<
    Type, // 存储的元素类型
    std::vector<Type>, // 底层容器类型（通常是std::vector）
    Compare // 比较函数对象类型
>;
```

priority_queue 的第三个参数是一个比较函数，用于定义元素之间的优先级顺序。默认情况下，这个比较函数是 std::less，这意味着 priority_queue 是一个最大堆（最大优先队列），即优先级最高的元素总是位于队列的顶部。
其常用的成员函数有：

1. push(const value_type& val)：向优先级队列中添加元素 val。
2. pop()：移除优先级队列顶部的元素。
3. top() const：返回优先级队列顶部的元素。
4. empty() const：检查优先级队列是否为空。
5. size() const：返回优先级队列中的元素数量。

3.2 实验时间安排

由于后续的实验课时不足，所以助教们决定将二叉树和图这两章内容对应的实验内容放在一起，本次实验的线下检查截止时间定于 12 月 6 日（线上检查时间为线下截止时间对应的周末），

即第 12 周到第 14 周共计两周半的时间。另外第 15 周的实验课用于本学期三次实验补交的检查，实验全部完成的同学无需再来，仍有实验未完成的同学请在十五周的实验课进行实验的补交，过后**不再**提供补交实验的机会。

3.3 项目构建说明

本次实验的两个部分可以分开构建，请同学们在实验过程中务必将工作文件夹根目录设置为 BinaryTree 或 graph，否则将无法进行项目的正常构建！
使用 VSCode 中的 CMake 插件进行构建和调试的步骤如下：

1. 在 VSCode 中打开文件夹 BinaryTree 或 graph，使得资源管理器的文件组织显示如下：

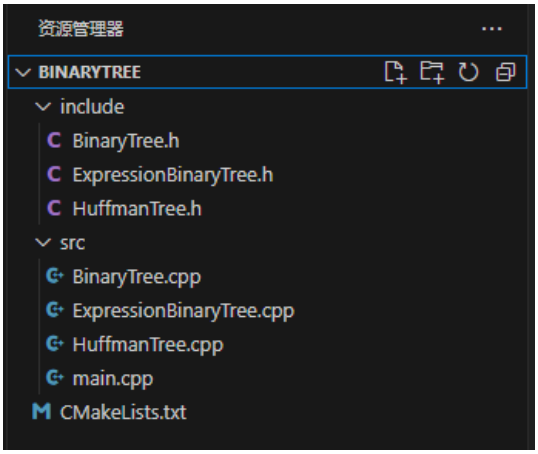

图 3: 打开 BinaryTree 文件夹时的文件组织

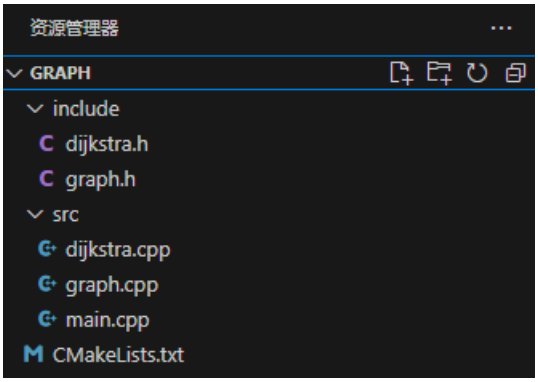

图 4: 打开 BinaryTree 文件夹时的文件组织

2. 点击左侧的 CMake 图标，进行 CMake 的配置；

3. 点击配置的图标：选择相应的工具包（mingw32 的那个）

图 5: 点击配置图标

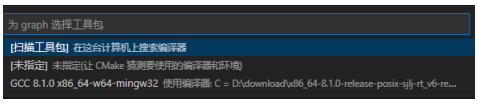

图 6: 选择工具包

4. 点击生成，即可生成该文件组织架构下的测试程序的二进制文件：

图 7: 生成测试程序的二进制文件

5. 在需要调试的地方打上断点，然后点击调试：即可进入调试状态：

图 8: 调试二进制文件生成

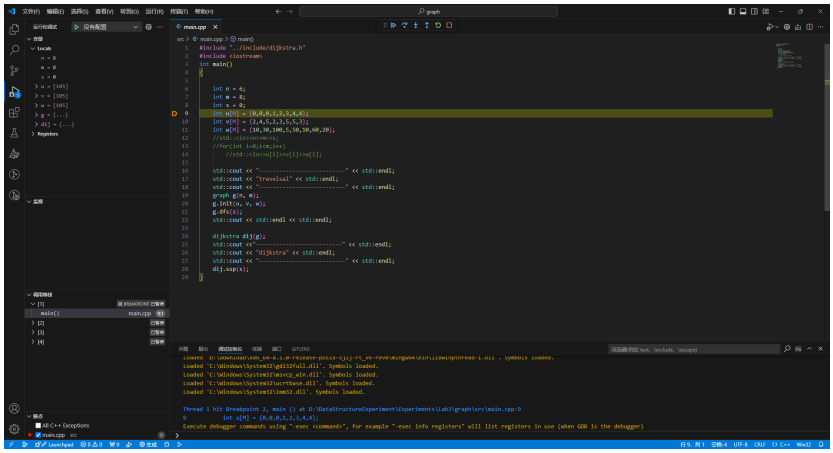

图 9: