# 北京邮电大学软件学院

# 2019-2020学年第1学期实验报告

**课程名称： 算法与数据结构**

**实验名称： 实验五**

**实验完成人：**

**姓名：**\_\_王衔飞\_\_\_**学号：**\_ \_\_**成绩：**\_\_\_\_\_\_\_\_

**指导教师：**\_\_\_\_\_\_贾红娓 \_李璐路\_\_\_\_\_\_\_\_

**日 期： 2019 年 12 月 1 日**

1. **实验目的**

树和图是两种应用极为广泛的数据结构，也是这门课程的重点。它们的特点在于非线性。 稀疏矩阵的十字链表存储结构也是图的一种存储结构。本章实验继续突出了数据结构加操作的程序设计观点，但根据这种结构的非线性特点，将操作进一步集中在遍历操作上，因为遍历操作是其它众多操作的基础。遍历逻辑的（或符号形式的）结构，访问动作可是任何操作。本次实验希望帮助学生熟悉各种存储结构的特征，以及如何应用图结构解决具体问题（即原理与应用的结合）。

1. **实验内容**

**必做内容**

* + - 1. **图遍历的演示**

**[问题描述]**

　　很多涉及图上操作的算法都是以图的遍历操作为基础的。试写一个程序，演示无向图的遍历操作。

**[基本要求]**

　　以邻接表为存储结构，实现连通无向图的深度优先和广度优先遍历。以用户指定的结点为起点，分别输出每种遍历下的结点访问序列和相应生成树的边集。

**[测试数据]**

　　由学生依据软件工程的测试技术自己确定。注意测试边界数据，如单个结点。

**[实现提示]**

　　设图的结点不超过30个，每个结点用一个编号表示（如果一个图有n个结点，则它们的编号分别为1,2,…,n）。通过输入图的全部边输入一个图，每条边为一个数对，可以对边的输入顺序做出某种限制。注意，生成树的边是有向边，端点顺序不能颠倒。

**选做内容**

1. 借助于栈类型（自己定义和实现）将深度优先遍历用非递归算法实现。
2. 以邻接多重表为存储结构建立深度优先生成树和广度优先生成树，再按凹入表或树形打印生成树。
3. 实现有向图的遍历操作。
4. **实验环境**

OS：macOS 10.15 IDE：CLion 2019.2 Compiler：GCC8.2

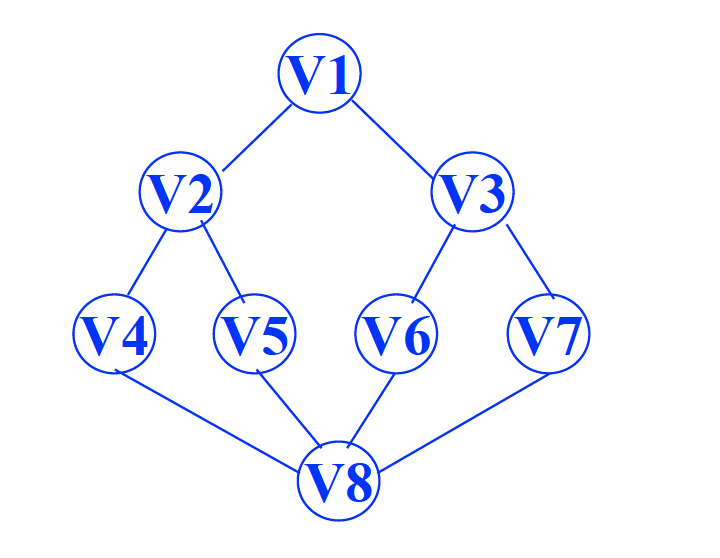
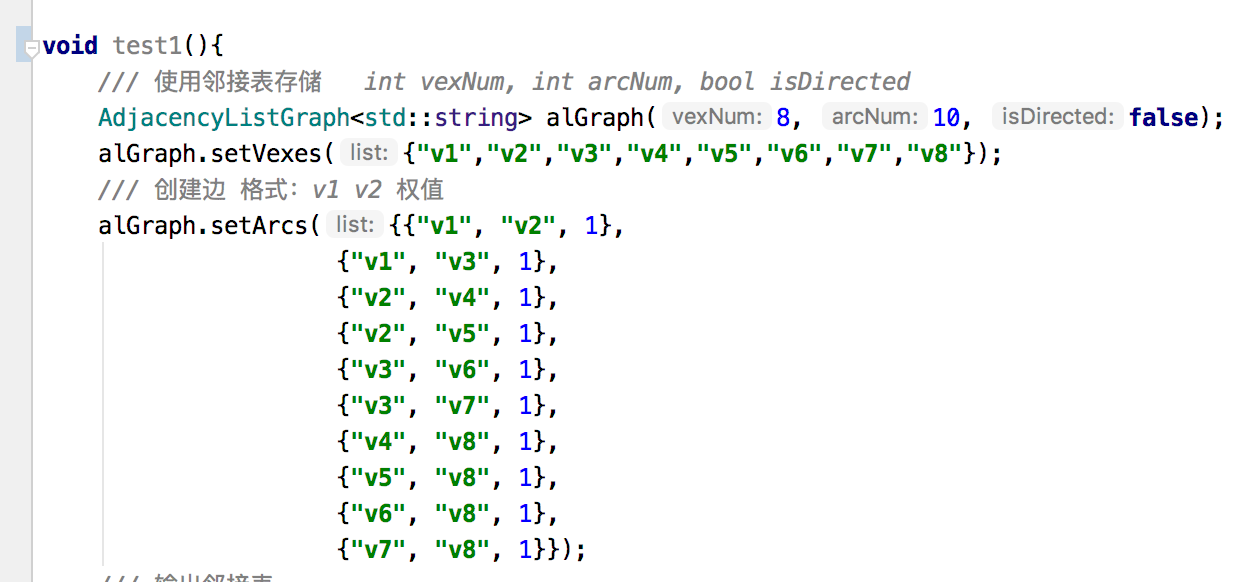
1. **实验结果**

完成了所有任务，以及选做1题和3题（2还没学）。并分别测试了无向图和有向图的DFS（依赖递归）、DFS、BFS，以及不连通图、单个节点等特殊测试数据，测试数据及截图见附件。

1. **附录**

**测试一：**

8顶点10边的无向图，以v1为起点，测试数据图片及创建图代码：

输出结果：

"/Users/xianfei/Documents/GitHub/SSE18\_Homework/Data Struct Programming/Experiment 5/cmake-build-debug/Graph"

邻接表：

0 v1 -> 1 1 -> 2 1 -> ^

1 v2 -> 0 1 -> 3 1 -> 4 1 -> ^

2 v3 -> 0 1 -> 5 1 -> 6 1 -> ^

3 v4 -> 1 1 -> 7 1 -> ^

4 v5 -> 1 1 -> 7 1 -> ^

5 v6 -> 2 1 -> 7 1 -> ^

6 v7 -> 2 1 -> 7 1 -> ^

7 v8 -> 3 1 -> 4 1 -> 5 1 -> 6 1 -> ^

深度优先搜索:

v1-v2

v2-v4

v4-v8

v8-v5

v8-v6

v6-v3

v3-v7

v1 v2 v4 v8 v5 v6 v3 v7

深度优先搜索(不递归):

v1 v2 v4 v8 v5 v7 v3 v6

宽度优先搜索:

v1-v2

v1-v3

v2-v4

v2-v5

v3-v6

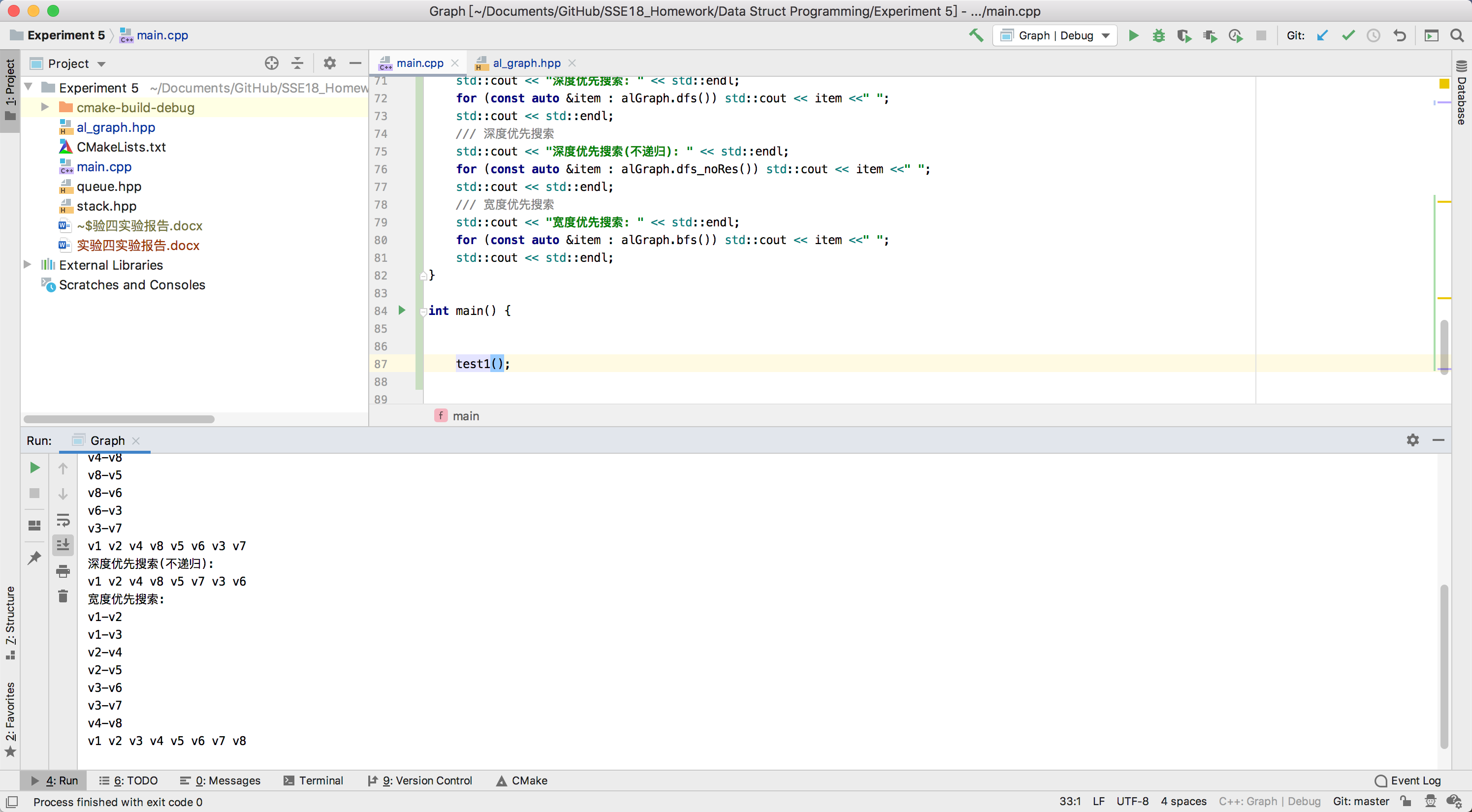
v3-v7

v4-v8

v1 v2 v3 v4 v5 v6 v7 v8

Process finished with exit code 0

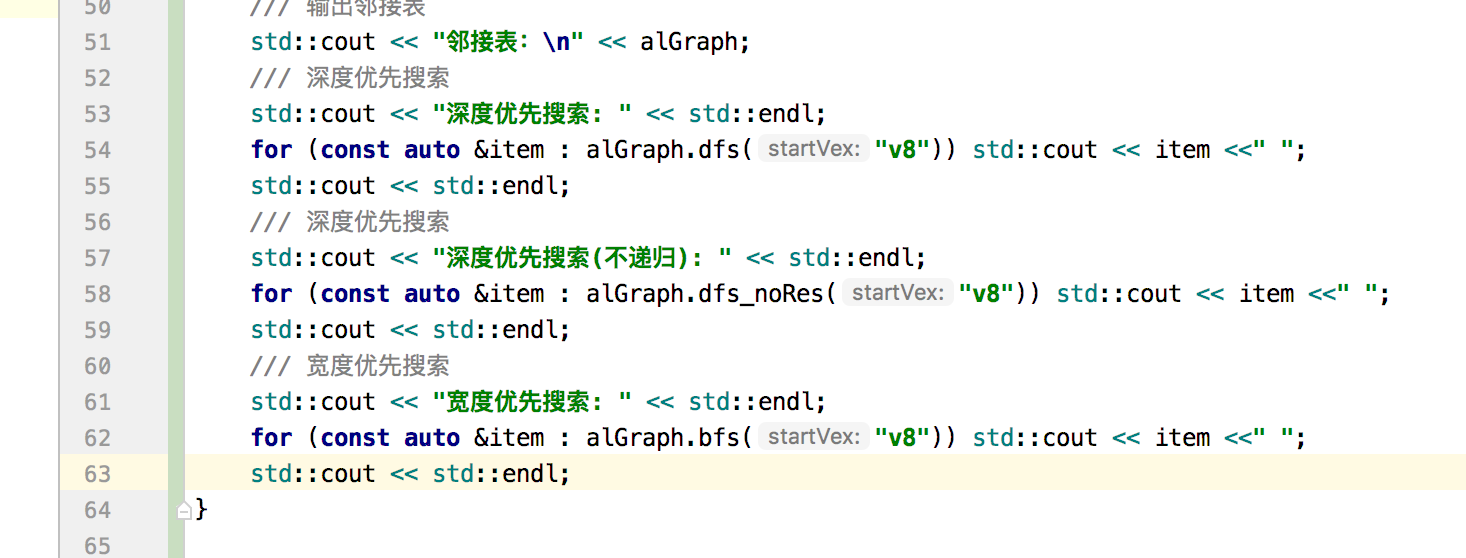
截图：



分析：非递归与递归方法的DFS输出结果不同，但是都符合DFS特征。

**测试二：**

与测试一数据相同，将起点设为v8，代码如图



输出：

"/Users/xianfei/Documents/GitHub/SSE18\_Homework/Data Struct Programming/Experiment 5/cmake-build-debug/Graph"

邻接表：

0 v1 -> 1 1 -> 2 1 -> ^

1 v2 -> 0 1 -> 3 1 -> 4 1 -> ^

2 v3 -> 0 1 -> 5 1 -> 6 1 -> ^

3 v4 -> 1 1 -> 7 1 -> ^

4 v5 -> 1 1 -> 7 1 -> ^

5 v6 -> 2 1 -> 7 1 -> ^

6 v7 -> 2 1 -> 7 1 -> ^

7 v8 -> 3 1 -> 4 1 -> 5 1 -> 6 1 -> ^

深度优先搜索:

v8-v4

v4-v2

v2-v1

v1-v3

v3-v6

v3-v7

v2-v5

v8 v4 v2 v1 v3 v6 v7 v5

深度优先搜索(不递归):

v8 v4 v2 v1 v3 v6 v7 v5

宽度优先搜索:

v8-v4

v8-v5

v8-v6

v8-v7

v4-v2

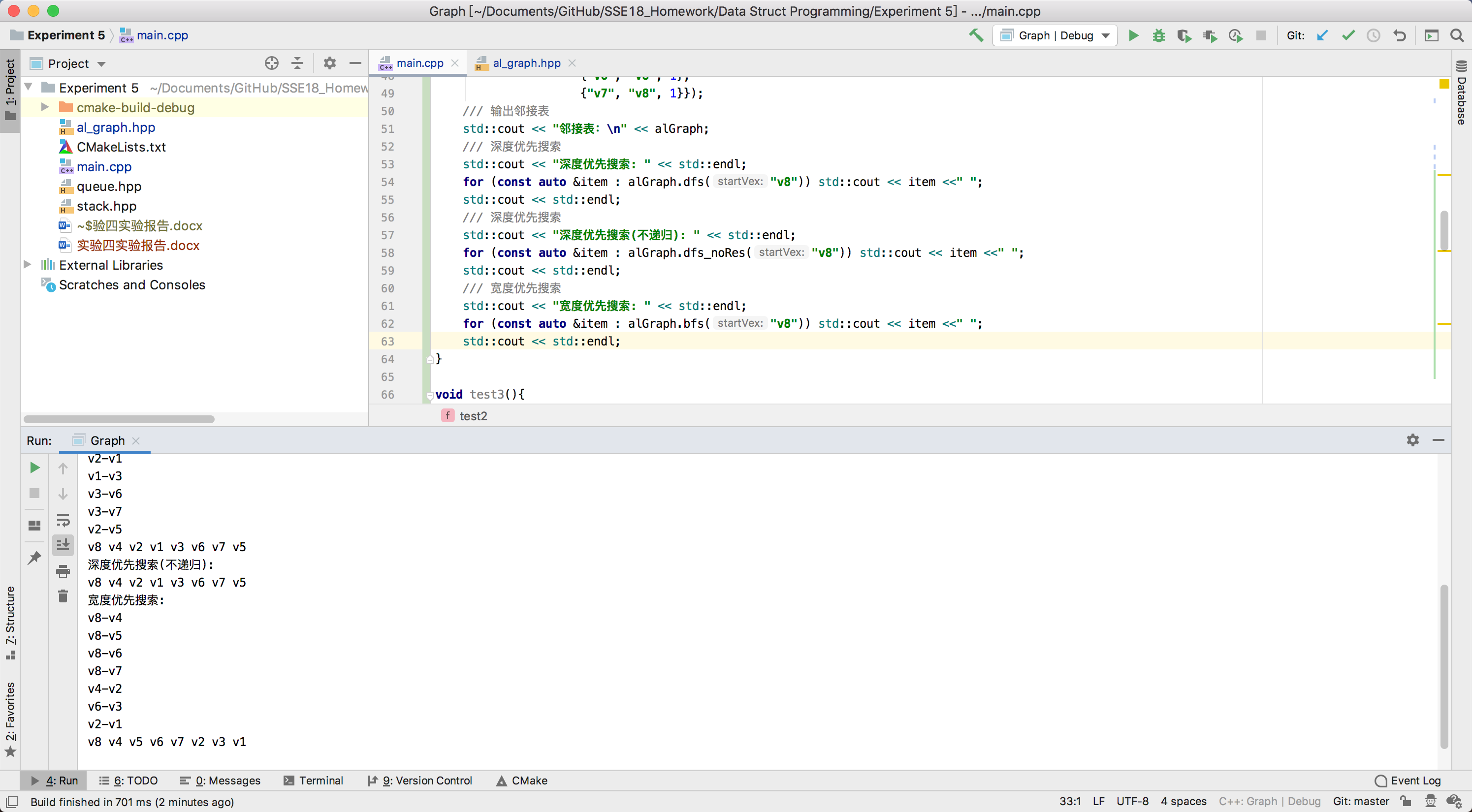
v6-v3

v2-v1

v8 v4 v5 v6 v7 v2 v3 v1

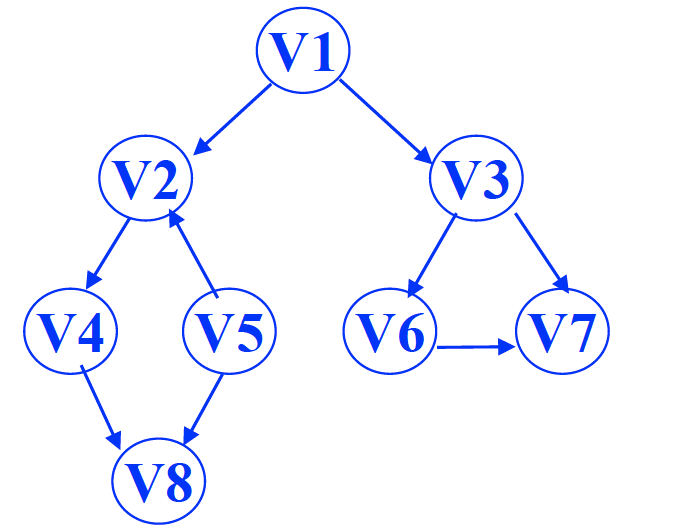
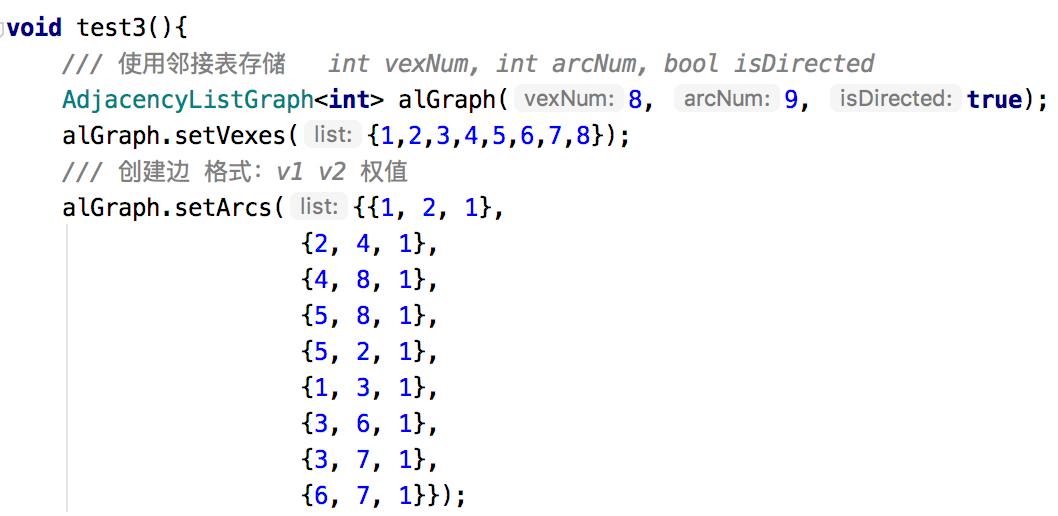
Process finished with exit code 0

截图：



**测试三：**

8顶点9边的有向不连通图，以v1为起点，测试数据图片及创建图代码：

输出：

"/Users/xianfei/Documents/GitHub/SSE18\_Homework/Data Struct Programming/Experiment 5/cmake-build-debug/Graph"

邻接表：

0 1 -> 1 1 -> 2 1 -> ^

1 2 -> 3 1 -> ^

2 3 -> 5 1 -> 6 1 -> ^

3 4 -> 7 1 -> ^

4 5 -> 7 1 -> 1 1 -> ^

5 6 -> 6 1 -> ^

6 7 -> ^

7 8 -> ^

深度优先搜索:

1-2

2-4

4-8

1-3

3-6

6-7

1 2 4 8 3 6 7 5

深度优先搜索(不递归):

1 2 4 8 3 6 7 5

宽度优先搜索:

1-2

1-3

2-4

3-6

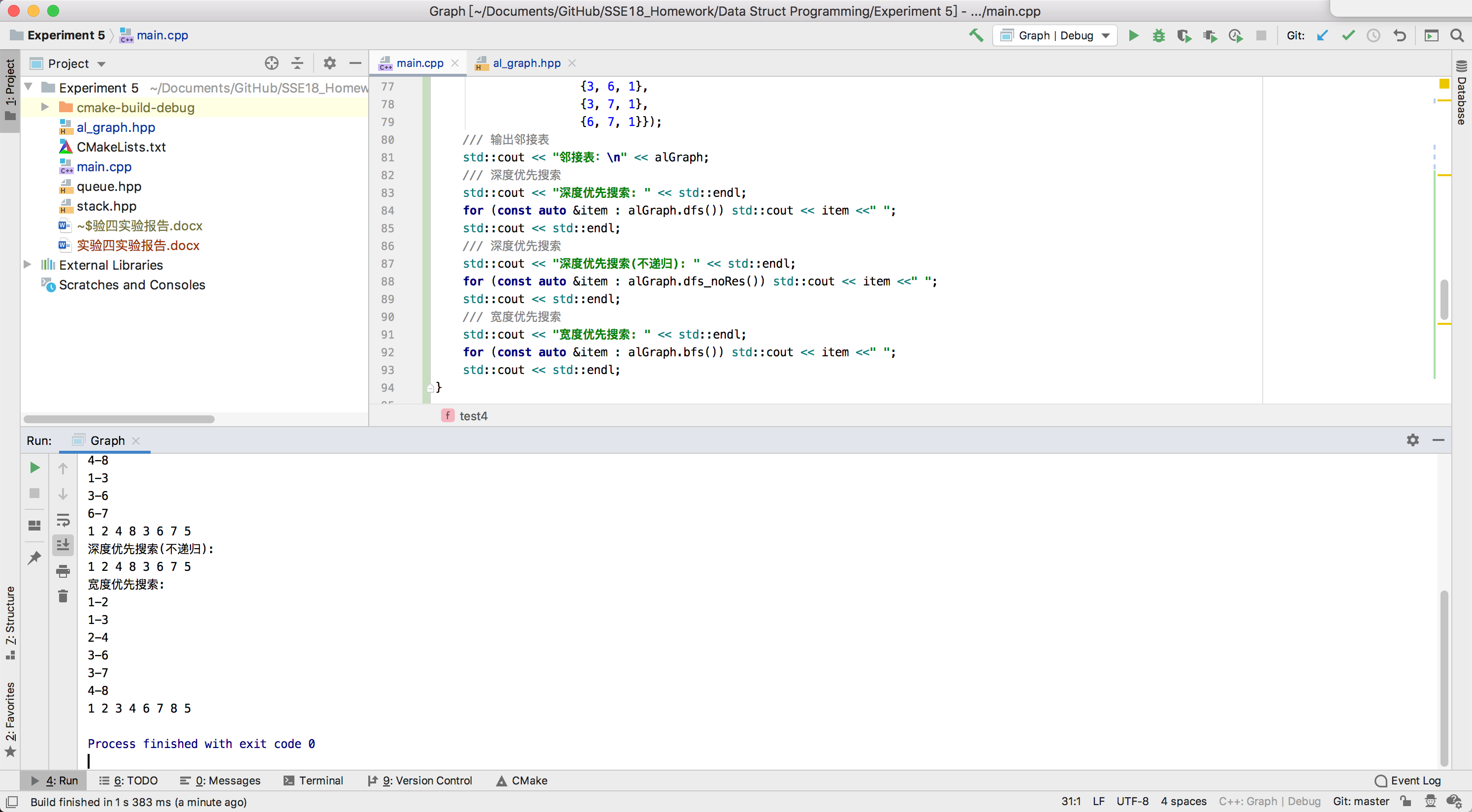
3-7

4-8

1 2 3 4 6 7 8 5

Process finished with exit code 0

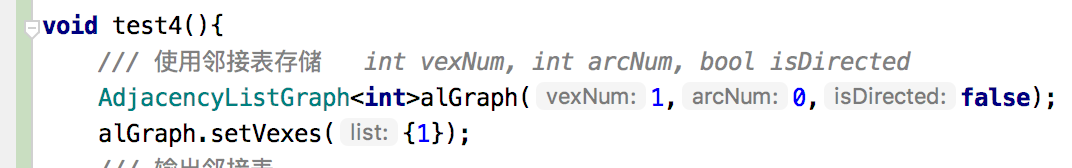
截图：



分析：由于该图从v1出发进行遍历时不是连通图，v5为一个独立的点，所以生成树边集中不包含5顶点所在的边。

**测试四：**

只有一个顶点的图，测试数据创建图代码：



输出：

"/Users/xianfei/Documents/GitHub/SSE18\_Homework/Data Struct Programming/Experiment 5/cmake-build-debug/Graph"

邻接表：

0 1 -> ^

深度优先搜索:

1

深度优先搜索(不递归):

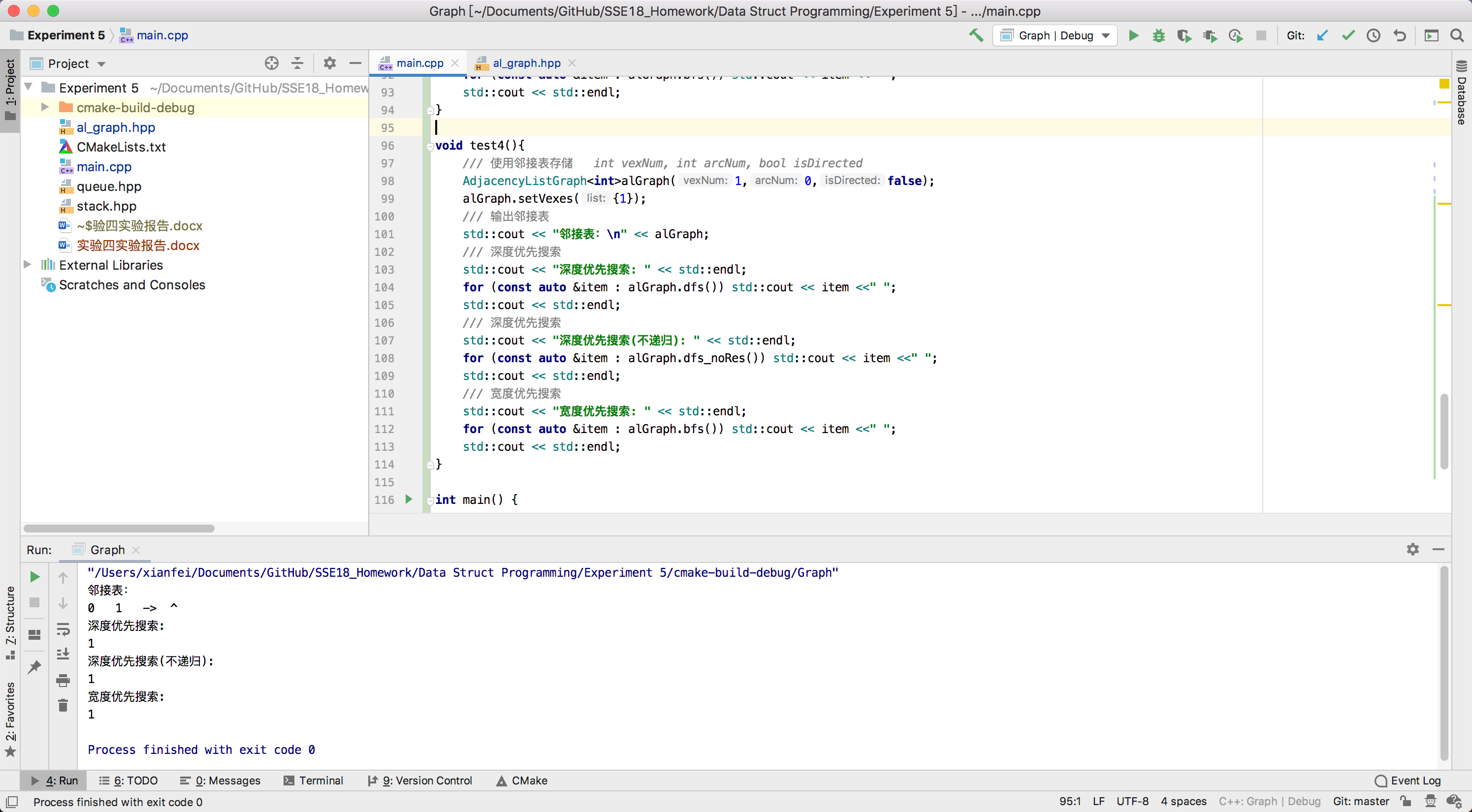
1

宽度优先搜索:

1

Process finished with exit code 0

截图：



**心得：**

不同的DFS、BFS算法可能会导致输出的结果不一样，但是都符合预期，因为选择同级顶点时选择方法可能不一样。从哪个点开始遍历可以通过改变for循环的初始值，也可以使用使用 (i+开始顶点)%\_顶点数 的方法来改变是从哪个顶点开始遍历。该测试代码中dfs()、bfs()方法返回std::vector数组的同时打印生成树边集。这两个方法不传入指定的开始顶点的话默认从第一个插入的顶点开始遍历。其中queue.hpp和stack.hpp为之前实验中自己写的队列和栈数据结构，与STL中的std::queue和std::stack的区别在于之前自己写的pop操作会返回值，其他的与STL相同。