**实验报告六**

1. **实验内容**

Graph Connectivity

1. **问题描述**

由于本实验是英文题，我们增加描述部分。

总述：

这个问题涉及一个无向图G = <V, E>，最初G没有任何边。本实验的任务是编写一个程序，能够评估两个不同顶点之间的连通性，并维护插入和删除边的操作。程序应该响应一组命令来操作和查询图形。

输入描述：

* N(2≤N≤1000):第一行表示图中顶点的个数。
* Q(1≤Q≤20000):第二行为命令条数。

接下来的Q行定义了命令，它们有三种类型:

* I u v:在节点u和v之间插入一条边，保证在发出此命令时，这些节点之间没有现有的边。
* D u v:删除节点u和v之间已经存在的边。（该命令只在节点u和v之间存在边时下发。）
* Qu v:查询节点u和节点v之间的连通性。

输出描述：

* 对于每个查询命令(Q u v)，输出应为:
* Y表示两个顶点u和v相连通。
* N表示两个顶点u和v不相连通。

1. **数据结构设计与思路分析**
2. **本实验所采用的数据结构：**

本实验采用了图数据结构，代码中的图是用邻接表方式实现的，每个顶点的邻接顶点存储在一个 unordered\_set 中。

1. **问题分解与算法思路：**

**该大问题可分为三个子问题来实现：**

1、插入边:

insertEdge ()

功能：在图中插入边。

时间复杂度：平均时间复杂度是 O(1)，但最坏情况下可能达到 O(V)，其中 V 是顶点数。

空间复杂度：由于这个函数不增加额外的空间使用，所以其空间复杂度是 O(1)。

1. 删除边

deleteEdge ()函数

功能：在图中删除边。

时间复杂度：平均时间复杂度是 O(1)，但最坏情况下可能达到 O(V)，其中 V 是顶点数。

空间复杂度：由于这个函数不增加额外的空间使用，所以其空间复杂度是 O(1)。

1. 检查两个结点间是否存在路径

DFS ()函数

功能：用于检查从顶点 u 到顶点 v 是否存在路径。

时间复杂度：在最坏情况下，需要访问图中的每个顶点和每条边一次，因此时间复杂度是 O(V + E)，其中 V 是顶点数，E 是边数。

空间复杂度：主要由递归调用栈造成，最坏情况下与顶点数相同，因此空间复杂度是 O(V)。

1. 查询两个顶点是否连通

isConnected ()函数

功能：检查两个顶点是否连通。

时间复杂度：这个函数调用了 DFS() 函数，因此它的时间复杂度是 O(V + E)。

空间复杂度：由于 DFS 的空间复杂度是 O(V)，因此 isConnected ()函数的空间复杂度也是 O(V)。

1. **实验结果与分析**

本实验测试用例：

3

7

Q 1 2

I 1 2

I 2 3

Q 1 3

D 1 2

Q 1 3

Q 1 1

理论的输出结果：

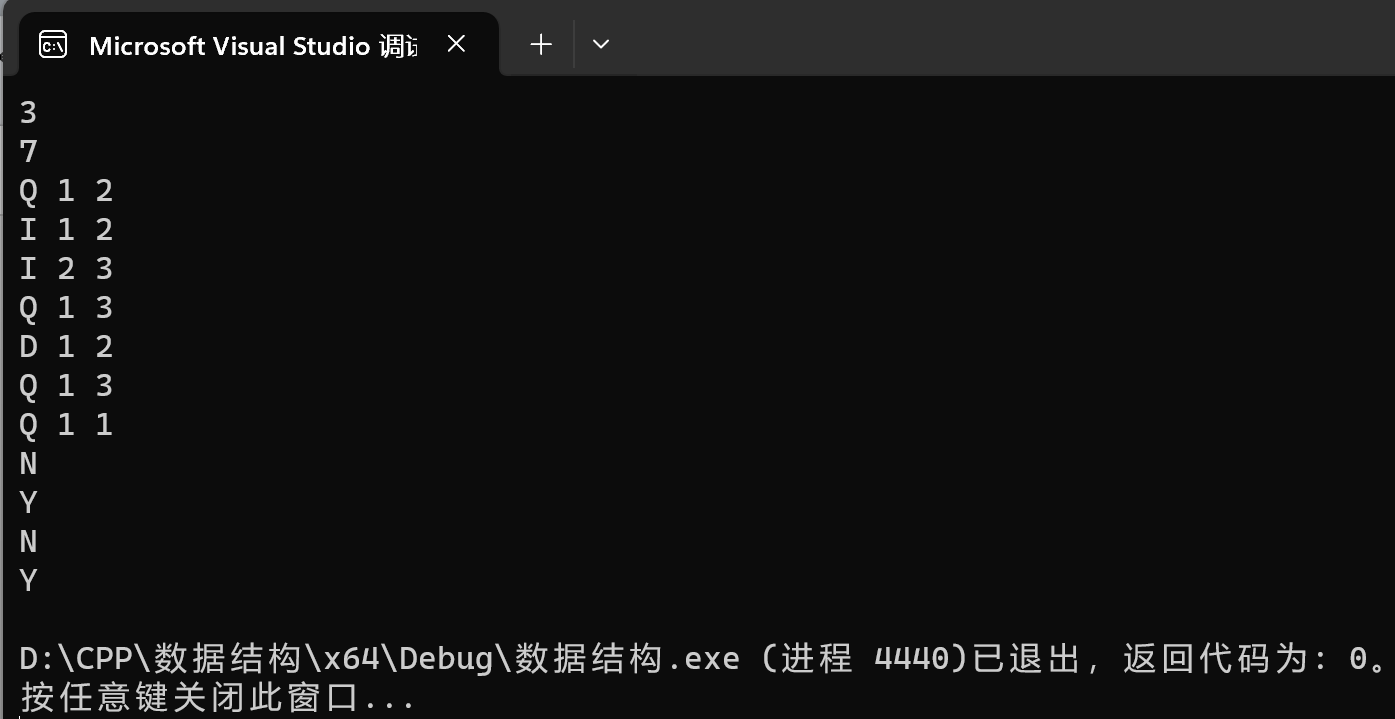
N

Y

N

Y

实际输出结果：



结果分析：

在这个示例中，最初，顶点1和2没有连接(因此是“N”)。插入边后，1和3连接起来(产生“Y”)，但一旦1和2之间的边被删除，1和3不再连接(产生“N”)。最后一个查询检查节点与自身的连通性，它总是“Y”。

1. **实验小结**

通过本次实验，我加深了对图论部分知识的理解，对连通性等基本概念有了更深的解读，同时回顾了图的存储方法（邻接表法），插入删除边等基本操作，深度优先搜索的实现方法，提高了解决图问题的能力。