**数据结构（C++）**

**实验报告**

**学 号: 2021902610**

**姓 名: 刘文越**

**院 系: 信息工程学院**

**专 业: 计算机科学与技术**

**指导教师: 慕晨**

**日 期: 2022.12.29**

目录

[实验一 线性表的设计与实现 3](#_Toc120821247)

[1. 实验内容与要求 3](#_Toc120821248)

[2. 实验环境 3](#_Toc120821249)

[3. 实验算法伪代码 3](#_Toc120821250)

[4. 实验结果 3](#_Toc120821251)

[实验二 二叉树的设计与实现 4](#_Toc120821252)

[1. 实验内容与要求 4](#_Toc120821253)

[2. 实验环境 4](#_Toc120821254)

[3. 实验算法伪代码 4](#_Toc120821255)

[4. 实验结果 4](#_Toc120821256)

[实验三 图的设计与实现 5](#_Toc120821257)

[1. 实验内容与要求 5](#_Toc120821258)

[2. 实验环境 5](#_Toc120821259)

[3. 实验算法伪代码 5](#_Toc120821260)

[4. 实验结果 5](#_Toc120821261)

[实验心得 6](#_Toc120821262)

[1. 实验支撑的毕业要求指标点 6](#_Toc120821263)

[2. 实验心得 6](#_Toc120821264)

# 实验一 线性表的设计与实现

## 实验内容与要求

1. 理解线性表的概念
2. 设计一个线性表，分别用顺序存储结构和链式存储结构实现，完成线性表的构造、查找、插入、删除、输出等基本操作。
3. 掌握两种存储结构的优缺点以及在实际应用中如何选择存储方式。

## 实验环境

1）硬件环境：

笔记本电脑

* 处理器：英特尔 Core i7-10870H @ 2.20GHz 八核
* 主 板：微星 MS-17K2
* 内 存：32 GB
* 硬 盘：三星 MZVLB2T0HMLB-00000
* 显 卡：Nvidia GeForce RTX 3070 Laptop GPU

2）软件环境

* Windows 10 系统
* Visual Studio Code

## 实验算法（伪代码或者活动图）

|  |
| --- |
| 顺序存储insert算法 |
| **Input:** Array length ***len***, Array ***data***[1…len], index ***pos***, Inserted element ***x***  **Output:** None   1. **for** *i*←*len* **to** *pos* 2. *data*[1]←*data*[i-1] 3. *data*[pos-1]←*x* 4. **return** None |

|  |
| --- |
| 顺序存储remove算法 |
| **Input:** Array length ***len***, Array ***data***[1…len], index ***pos***  **Output:** Deleted element   1. *tmp*←*data*[pos-1] 2. **for** *i*←*pos* **to** *length-1* 3. *data*[i-1]←*data*[i] 4. *data*[pos-1]←*x* |

|  |
| --- |
| 链式存储insert算法 |
| **Struct:** Node {  Element data, //The data of this node.  Element p // Its value is the field of next node.  }  **Input:**  Parameter whose value is Node's filed ***head***,  index ***pos***,  Inserted element ***x***  **Output:** None   1. *now*←*head* 2. *cnt←0* 3. **while** *now* != 0 and *cnt*<*i*-1 **do** 4. *now←now*->*p* // ”->” means the element of the field it pointed 5. *cnt*←*cnt*+1 6. **end while** 7. **if** *now*=0 **then** 8. **return** Wrong Position of Insert 9. **else** 10. *tmp*←new Node field 11. *tmp*->data←*x* 12. *tmp*->*p*←*now*->*p* 13. *now*->*p*←*tmp* 14. **end if** 15. **return** None |

|  |
| --- |
| 链式存储remove算法 |
| **Struct:** Node {  Element data, //The data of this node.  Element p // Its value is the field of next node.  }  **Input:**  Parameter whose value is Node's filed ***head***,  index ***pos***,  **Output:** Deleted element   1. *now*←*head* 2. *cnt←0* 3. **while** *now*!=0 and *cnt* < *i*-1 **do** 4. *now*←*now*->p 5. cnt←cnt+1 6. **end while** 7. **if** *now*=0 **then** 8. **return** Wrong Position of Delete 9. **else** 10. *p*←*now*->*p* 11. *tmp*←*p*->*data* 12. *now*->*p*←*p*->*p* 13. delete the field that p pointed 14. **return** tmp 15. **end if** |

## 实验结果

**顺序存储：**

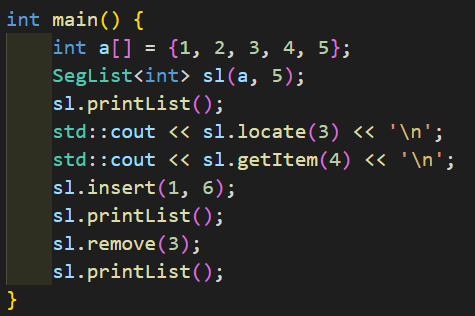
在主函数中我们使用一个int数组初始化，然后打印线性表

接着我们打印3在线性表中的索引、打印线性表中索引为4的元素

接着我们在索引1后插入新元素6，并打印线性表

接着我们移除索引为3的元素，并打印线性表

主函数如图所示：



在控制台中对应的输出结果为：



**链式存储：**

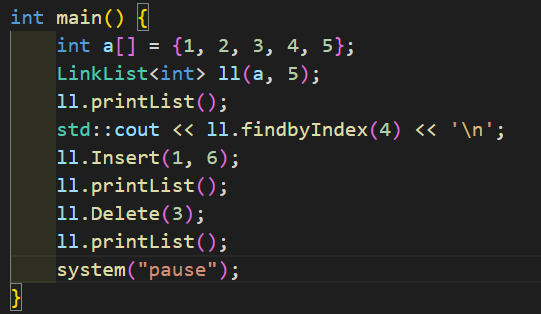
在主函数中我们使用一个int数组初始化，然后打印线性表

接着我打印线性表中索引为4的元素

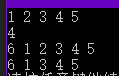
接着我们在索引1前插入新元素6，并打印线性表

接着我们移除索引为3的元素，并打印线性表

主函数如图所示：



控制台中对应的输出结果为：



# 实验二 二叉树的设计与实现

## 实验内容与要求

1. 理解二叉树的基本结构
2. 编程实现二叉树的构造、前序、中序、后序、层序遍历等基本功能
3. 选作：编程实现二叉树按值查找、输出叶子结点、求深度等操作。
4. 选作：二叉树的非递归遍历

## 实验环境

1）硬件环境：

笔记本电脑

* 处理器：英特尔 Core i7-10870H @ 2.20GHz 八核
* 主 板：微星 MS-17K2
* 内 存：32 GB
* 硬 盘：三星 MZVLB2T0HMLB-00000
* 显 卡：Nvidia GeForce RTX 3070 Laptop GPU

2）软件环境

* Windows 10 系统
* Visual Studio Code

## 实验算法（伪代码或者活动图）

|  |
| --- |
| 二叉树构造算法createBiTree |
| **Struct:** Node {  Element data, //The data of this node.  Element lCld, // Its value is the field of left node.  Element rCld, // Its value is the field of right node.  }  **Input:**  Parameter whose value is Node's filed ***node***,  Data ***curNodeData***,  **Output: None**   1. *node*←new field of struct Node 2. *node*->*data*←*curNodeData* 3. createBiTree(*node*->*lCld*) 4. createBiTree(*node*->r*Cld*) |

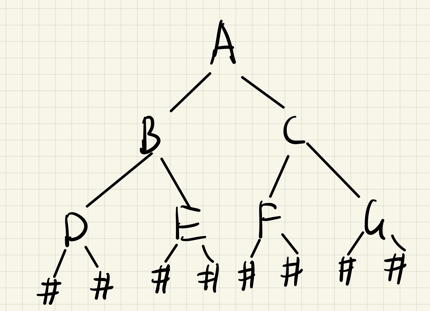
|  |
| --- |
| 二叉树前序遍历算法preOrder |
| **Struct:** Node {  Element data, //The data of this node.  Element lCld, // Its value is the field of left node.  Element rCld, // Its value is the field of right node.  }  **Input:**  Parameter whose value is Node's filed ***node***,  **Output:** Preorder traversal of binary trees   1. **if** *node* != 0 **then** 2. output the value of *node*->*data* 3. preOrder(*node*->*lCld*) 4. preOrder(*node*->r*Cld*) 5. **end if** |

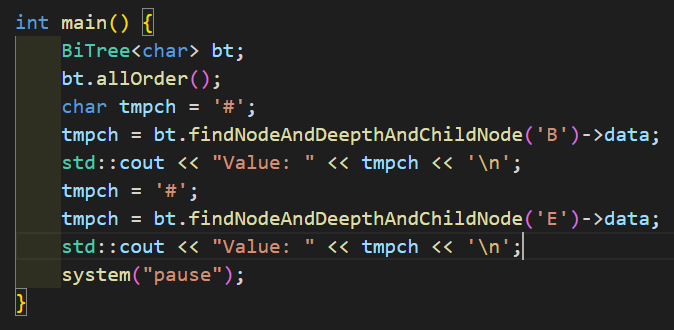
## 实验结果

输入内容为：

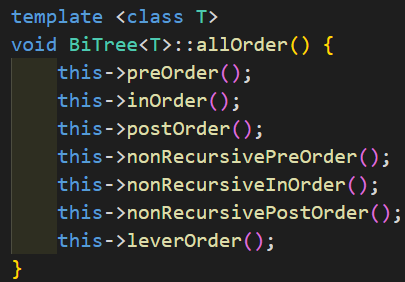
ABD##E##CF##G##

二叉树如图：

  
main函数内代码为：

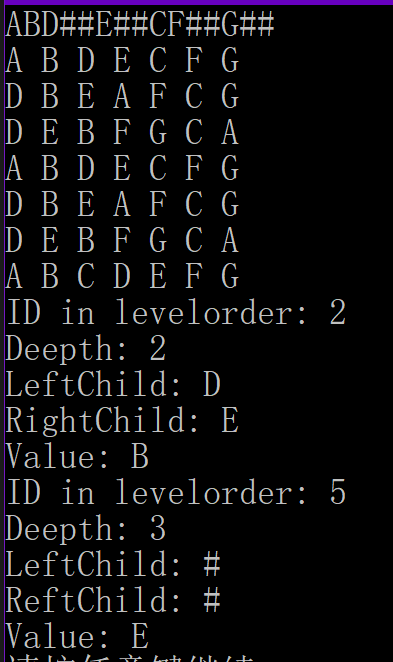


其中allOrder函数体为：



即调用所有实现的二叉树遍历方式

运行结果如图：



# 实验三 图的设计与实现

## 实验内容与要求

1. 理解邻接矩阵和邻接表的概念
2. 任选一种存储方式实现以下操作：图的构造、深度优先遍历以及广度优先遍历
3. 选作：最小生成树和最短路径。

## 实验环境

1）硬件环境：

笔记本电脑

* 处理器：英特尔 Core i7-10870H @ 2.20GHz 八核
* 主 板：微星 MS-17K2
* 内 存：32 GB
* 硬 盘：三星 MZVLB2T0HMLB-00000
* 显 卡：Nvidia GeForce RTX 3070 Laptop GPU

2）软件环境

* Windows 10 系统
* Visual Studio Code

## 实验算法（伪代码或者活动图）

|  |
| --- |
| 图的创建 |
| **Input:**  The number of nodes ***nodeNum,***  The number of edges ***edgeNum,***  The data array of nodes ***nodeArray*[1…*nodeNum*]**  The data of input edge ***u*[1…*edgeNum*], *v*[1…*edgeNum*], *w*[1…*edgeNum*],**  The nodeArray ***adjList*[1…*nodeNum*]**  The current point of processing ***now***  **Output: None**   1. **for** *i*←0 **to** *nodeNum*-1 2. *adjList*[*i*].*node*←*i* 3. *adjList*[*i*].*first*←0 4. **for** *i*←0 **to** *edge*-1 5. *now*->*idx*←*v* 6. *now*->*weight*←*w* 7. *now*->*next*←*adjList*[*u*].*first* 8. *adjList*[*u*].*first*←*now* 9. *now*->*idx*←*u* 10. *now*->*weight*←*w* 11. *now*->*next*←*adjList*[*v*].*first* 12. *adjList*[*v*].*first*←*now* |

|  |
| --- |
| 图的深度优先遍历 |
| **Input:**  The number of nodes ***nodeNum,***  The array used to mark whether it has been used ***st*[1…*nodeNum*],**  The nodeArray ***adjList*[1…*nodeNum*],**  The stack ***stack,***  The start pos ***s***  The current pos ***p***  **Output: Depth-first traversal of the graph**   1. *st*[*s*]←**true** 2. *stack*.push(s) 3. **while** *stack* is not empty **do** 4. *s*←*stack*.top 5. *p*←*adjList*[*s*].*first* 6. **while** *p*!=0 **and** *st*[*p*->*idx*]!=0 **do** 7. *p*←*p*->*next* 8. end while 9. **if** *p*!=0 **then** 10. *s*←*p*->*idx* 11. Output the value of *adjList*[*s*].*node* 12. *st*[*s*] = **true** 13. *stack*.push(*s*) 14. **else** 15. *stack*.pop 16. **end if** 17. **end while** |

## 实验结果

输入内容为：

9 11

0 1 99

1 3 99

3 7 99

7 8 99

1 4 99

4 7 00

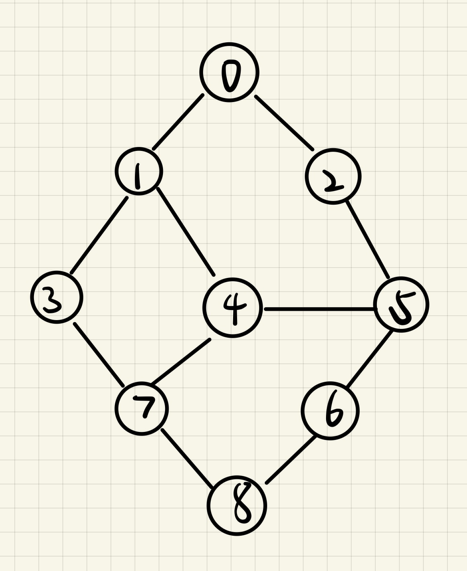
4 5 99

0 2 99

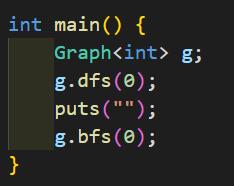
2 5 99

5 6 99

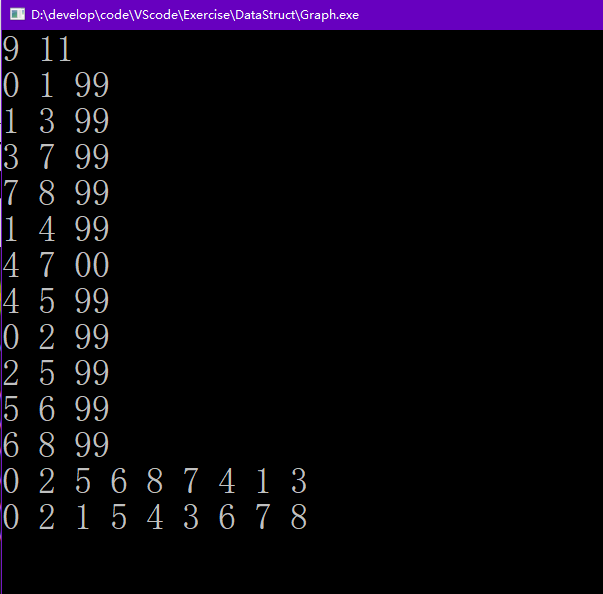
6 8 99

建立一个如图所示的无向图 

以下是main函数截图，建立图，并分别从0点开始深度优先遍历和广度优先遍历。



运行截图如下：



# 实验心得

## 实验支撑的毕业要求指标点

数据结构课程实验协助数据结构课程支撑以下四项毕业要求指标点：

指标点1-3：能够将数学、自然科学、工程科学和计算机科学与技术相关知识和建模求解方法用于推演、分析计算机领域的复杂工程问题；

指标点2-3：能够认识到计算机领域的工程问题有多种解决方案可以选择，并结合文献查阅及研究，比较、寻求可替代的解决方案；

指标点5-1：了解计算机专业常用的仪器设备、开发工具、信息检索工具、工程工具和模拟软件的使用原理和方法，并理解其局限性；

指标点6-2：能够分析和评价计算机工程实践和复杂工程问题解决方案对社会、健康、安全、法律以及文化的影响，具备在工程实践中考虑上述制约因素的意识。

## 实验心得

1. 通过实验我学习到了线性表、二叉树、无向图等基本的数据结构，巩固了数据结构的构造方式，锻炼了处理代码的能力。而学习数据结构，可以依据他们的特性来存储各种特定数据，实现数据的增、删、查、改等功能。
2. 例如图的dfs和bfs，两种图的遍历方式采用不同的遍历策略，dfs会优先深度往后寻点，而bfs则会优先同层点。因此，DFS比较适合判断图中是否有环，寻找两个节点之间的路径，有向无环图（DAG）的拓扑排序，寻找所有强连通片（SCC），无向图中寻找割点和桥等；BFS则比较适合判断二分图，以及用于实现寻找最小生成树（MST），如在BFS基础上的Kruskal算法。还有寻找最短路径问题（如Dijkstra算法）。
3. 在软件开发的过程中，我们应该更加注重算法和存储方式的选择。d对于一个问题的求解，或者是一个项目的实现，我们应该优先去分析问题或者项目数据的特点，或是实现方式过程中的特殊点，作为依据去选择适当的数据结构去存储数据和求解问题，以此达成更加优秀的效率。
4. 在软件开发过程中，我们也要注意空间复杂度和时间复杂度的计算，选择适当的算法去优化控制程序的时空复杂度，在处理内存时要有开辟有释放，避免内存溢出或内存泄漏等情况的发生，造成程序崩溃。