

|  |
| --- |
| 数据结构  实验报告 |
|  |
| 姓名：刘明承  学号：919106840423  专业班级：19级计科一班 |



# 上机实验1

## 一元多项式的相加算法

实验目的：编写C++算法实现一元多项式加法

实验环境：PC计算机；Windows10操作系统；Dev-C++

实验内容及结果：

1. 存储方式：链表

typedef struct PolyNode

{

int Expon; 次数

int Coef; 系数

Polynomial Next; 连接下一项

} \*Polynomial;

1. 主要函数：
   1. Polynomial ReadPoly();

读入一个多项式，并采用链表存储。建立一个空表头head，循环输入多项式各项系数Coef与次数Expon，新建结点tmp将其保存，再按正确的顺序（即次数下降的顺序）将结点连接。返回指向链表的指针时要去掉空的表头。

* 1. Polynomial AddPoly( poly a, poly b ) ;

实现多项式加法 。设置两个指针m、n分别指向两个多项式a、b，新建一个多项式ans作为二者的和。先循环处理m和n都非空的情况直至有一方为空，每次利用Expon值较大者的结点信息为ans连接一个新的结点，并将对应指针后移；若二者Expon相等则将两指针同时后移。当其中一方为空，另一方非空时，直接将非空指针后面的内容复制给ans，直至m、n指针均为NULL，返回指针指向ans。

* 1. void PrintPoly( poly p );

打印多项式。比较复杂的是首项、次数为0或1、系数为0时的处理。用if-else语句处理即可。

1. 实验结果

A (x) = 7 + 3x + 9 x 8 + 5 x 17

B (x) = 8 x + 22 x 7 – 9 x 8

A (x)与 B (x)相加的和多项式为

C (x) = A (x) + B (x) = 7 + 11x + 22x 7 + 5 x 17



# 上机实验2

## 二叉树的遍历

实验目的：基于非递归算法，实现一个二叉树的先序、中序以及后序遍历。

实验环境：PC计算机；Windows10操作系统；Dev-C++

实验内容及结果：

1. 三种遍历方式对比

先序遍历总是先访问根节点，再访问左、右子树；中序则先访问左子树、再访问根节点、最后访问右子树；后序遍历先访问左、右子树，再访问根节点。

1. 存储方式：链表

typedef struct TreeNode \*BinTree; //树

struct TreeNode{

ElementType Data; 结点数据

BinTree Left; 左子树

BinTree Right; 右子树

};

1. 主要函数：
   1. BinTree BuildTree()

建立一颗链式存储的二叉树。

* 1. void PreTra( BinTree bt )

先序遍历。建立一个栈S，用指针T指向二叉树根。当T非空或S非空时，T不断向左子树移动，并将路径上的结点全部压栈，同时输出Data，直至T->Left==NULL。这时，若栈S非空，则将最顶上的元素出栈，将T指向其右子树，并继续进行上述的循环。

* 1. void MidTra( BinTree bt )

中序遍历。在先序遍历的基础上，输出为每次S出栈的元素的Data。

* 1. void LastTra( BinTree bt )

后序遍历。后序遍历较先、中序遍历复杂，需要增加一个指针LastVisited标记上一个出栈的结点。在指针T左子树为NULL的基础上，只有LastVisited也为NULL或T的右子树时，才将T->Data输出。

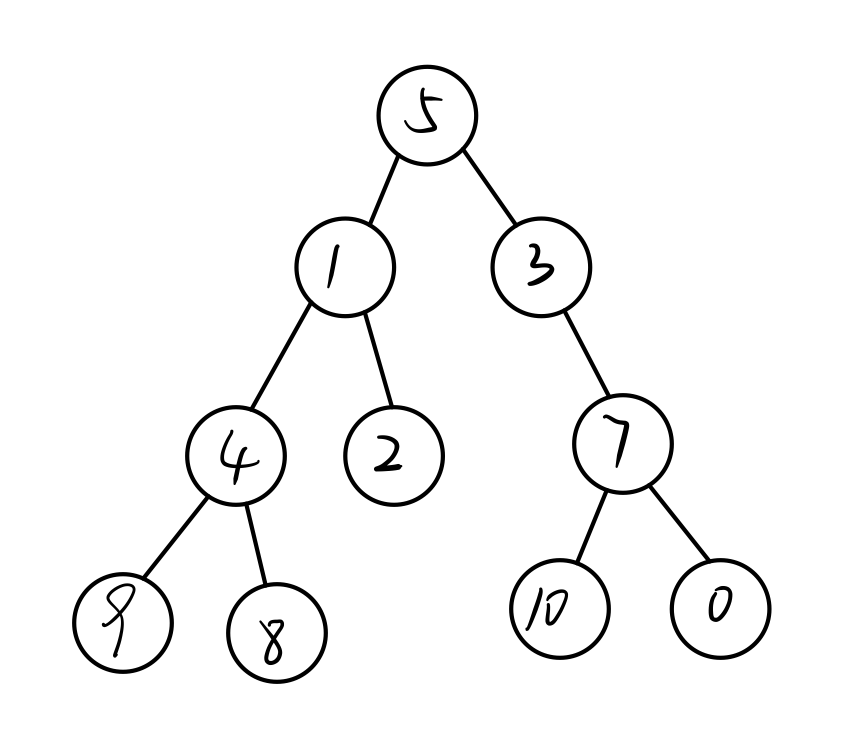
1. 实验结果

试构造一棵树，并对其进行遍历：

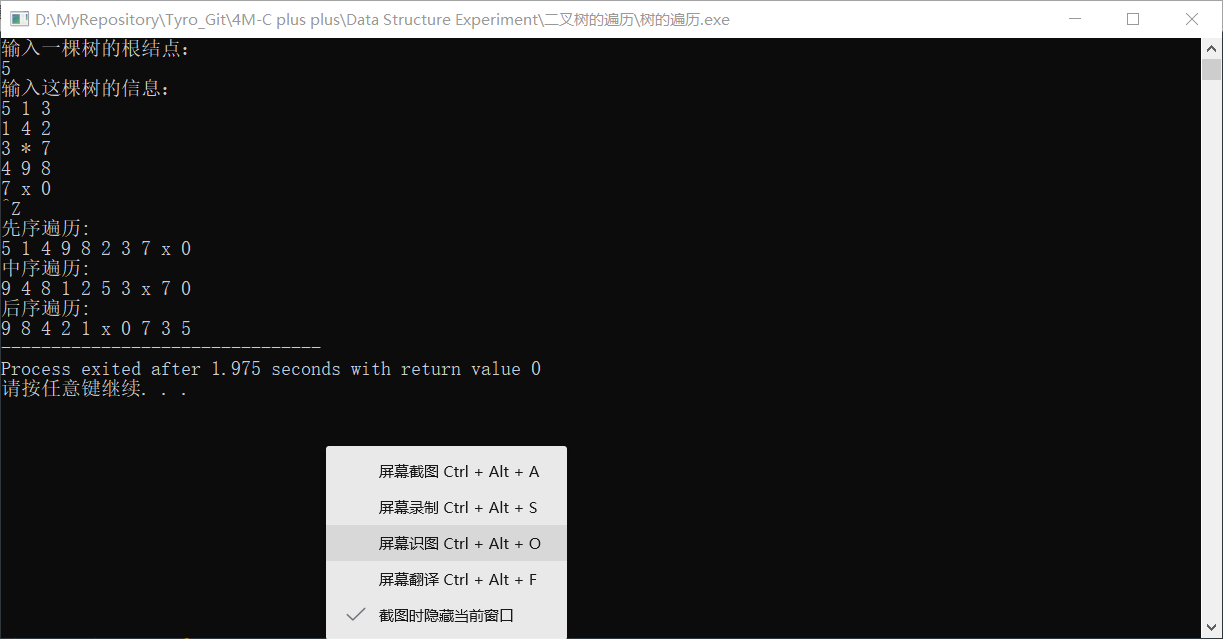
先序遍历5 1 4 9 8 2 3 7 x 0

中序遍历9 4 8 1 2 5 3 x 7 0

后序遍历9 8 4 2 1 x 0 7 3 5



程序结果：



# 上机实验3

## 图的遍历

实验目的：1）用非递归算法实现深度和者广度遍历一个无向图，并输出遍历结果，注意如果该图不连通，可能需要多次遍历。2）用非递归算法实现Kruskal算法和Prim算法。

实验环境：PC计算机；Windows10操作系统；Dev-C++

实验内容及结果：

1. 输入数据规模：对于 100\%100% 的数据：1<=N<=5000 1≤M≤2\*10^5。N、M分别为顶点数和边数。
2. 存储方式：

int Map[MAXN][MAXN]; 邻接矩阵—图

priority\_queue<Edge> Heap; Kruskal算法，最小堆维护边权

int N,M; 顶点\边数目

bool vis[MAXN]; 一个顶点是否被访问过

int dist[MAXN]; Prim算法，顶点到MST的最短距离

int parent[MAXN]; Prim算法，MST中顶点的父节点

int root[MAXN]; Kruskal算法，并查集

1. 主要函数：
   1. void InitMap();

以邻接矩阵方式建立图，将数据存在Map[MAXN][MAXN]中。输入数据规模较大故用scanf/printf输入输出。输入数据中可能在两个顶点间建立多条直接边，存储时只取其中最小正值。

* 1. void DFS();

非递归方式实现dfs，与树的遍历中先序遍历相似。建立堆栈S，将vis[]全部置为false。首先任选一个结点入栈，不失一般性，将结点1入栈，并将vis[1]标记为true；在S不为空的前提下，不断循环，将栈顶元素弹出，判断其邻接点是否都被访问过，若有结点i满足，则将i入栈，并将vis[i]置为true；直到S.empty()为true。

* 1. void BFS();

建立队列Q，将vis数组全部置为false。任选一个结点入队，不失一般性，将结点1入栈队并将vis[1]标记为true；在Q不为空的前提下，不断循环，将d队头元素弹出，判断其邻接点是否都被访问过，若有结点i满足，则将i入队，并将vis[i]置为true；直到Q.empty()为true。

* 1. void Prim();

将dist[]置为INF，边权和sum=0，生成树中顶点数tot=0。将1视为最小生成树的根（不失一般性），令dist[1]=0，parent[1]=-1。接下来每一次循环，在MST以外的所有结点中找到距离MST最近的（即对应边权值最小），将tot和sum值更新，再将所有点的dist值更新：已经收入MST中的置为0；与MST相邻的置为邻接矩阵中对应的值（取最小）；其他都为INF。若找不到这样的点，则跳出循环，判断tot是否为N。若为则说明找到了最小生成树，否则图不连通，不存在最小生成树。

* + 1. void Kruskal();

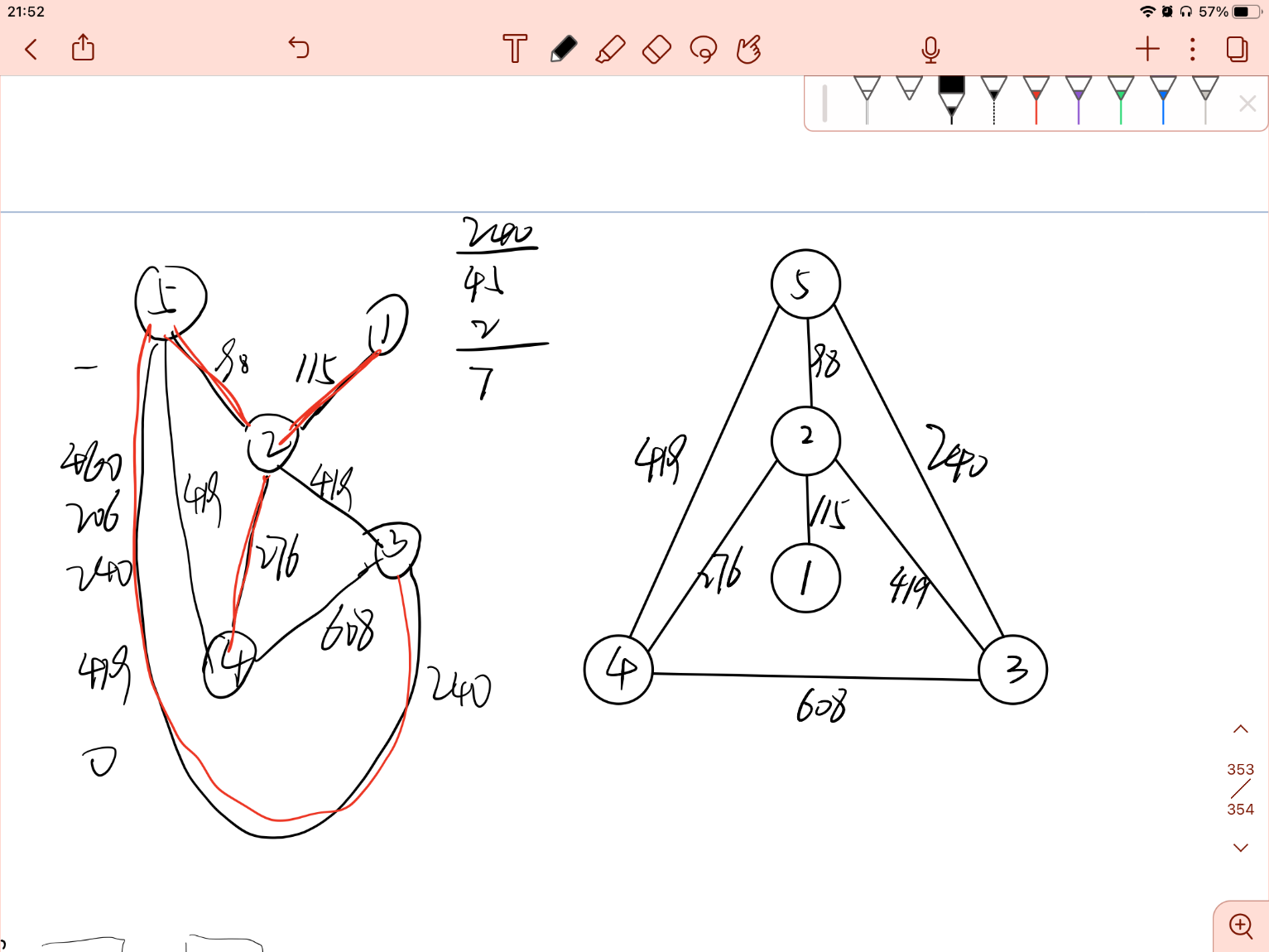
用root[]实现并查集，若两个顶点之间联通，则通过getRoot()二者得到结果相同，否则不同。将dist[]置为INF，边权和sum=0，MST中收入边数tot=0。在堆不空（仍有未利用边）且tot不足N-1的条件下进行循环，堆弹出权值最小的边e，若其两个顶点e.x、e.y属于一个集合，则删除这条边，否则将这条边添加到MST中，更新tot、sum和其中一点的root值。跳出循环后，若tot不足N-1，说明图不连通，不存在最小生成树，否则就找到了最小生成树。

* + 1. int getRoot( int );

root[i]中存储的是结点i最近一个连接的结点。令i=root[i]，直到root[i]==i。将这个值保存下来，如果两个结点的getRoot()值相等，说明他们已经联通。

1. 实验结果

构造如下所示连通图：



程序输出结果：

