《计算机实践》—数据结构
上机实验题和要求

南京航空航天大学

自动化学院

2022 年 10 月

实验 3. 树和图的实验

实验目的：

掌握二叉树的创建、遍历等操作实现；掌握图的邻接矩阵和邻接表创建及遍历等操作实现。

（以下实验 3.1、实验 3.2、实验 3.3 任选一节完成）

实验 3.1 树和图的基本实验

实验 3.1.1 二叉树的基本实验

1、实验内容

二叉树的创建、遍历及相关操作实现。

2、问题描述

编程实现二叉排序树的链式存储结构创建，并实现先序、中序、后序遍历（递归和非递归方法自选）操作，最后统计叶子结点的数目。

3、基本要求

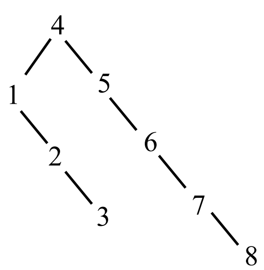
二叉树的创建方法可自行选择，确保生成二叉排序树。二叉树的高度不低于 4，结点数目不少于 8 个。结点使用不重复的正整数构建，按照要求输出相应的遍历序列及叶子结点数目。

4、输入输出要求

输入数据：自选方法输入结点个数及结点值，结点值使用不重复的正整数表示。

输出形式：输出正确的先序、中序、后序遍历序列，给出叶子结点数目。

5、测试用例

序号	输入	输出	说明
1	8 4 5 1 2 6 3 7 8  	先序遍历：4 1 2 3 5 6 7 8 中序遍历：1 2 3 4 5 6 7 8 后序遍历：3 2 1 8 7 6 5 4 叶结点数：2	一般情况

2	其他	其他	自行设计

6、选做内容：采取合适的方法，绘制出这棵二叉树。

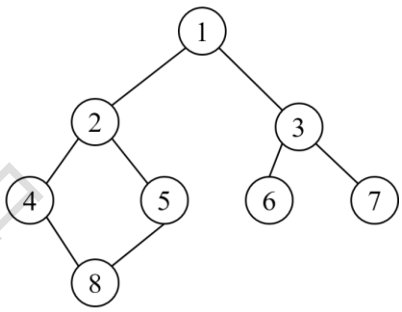
实验 3.1.2 图的基本实验

1、实验内容

图的创建、遍历操作实现。

2、问题描述

使用邻接矩阵存储图，实现深度优先遍历操作；使用邻接表存储图，实现广度优先遍历操作。



3、基本要求

根据题目要求，分别用邻接矩阵和邻接表创建右图，然后给出相应的深度优先遍历序列和广度优先遍历序列。

4、输入输出要求

输入数据：采用合适的方式创建上图，如先输入顶点数 n、边数 e，然后依次输入边信息；遍历时要求从键盘输入起始顶点。

输出形式：分别输出给定起始顶点的深度优先遍历序列和广度优先遍历序列。

5、测试用例

序号	输入		输出	说明
	图信息	起始顶点		
1	8 8 1 2 1 3 2 4 2 5 4 8 5 8 3 6 3 7 …	1	深度优先遍历：1 2 4 8 5 3 6 7 广度优先遍历：1 2 3 4 5 6 7 8	一般情况
2	其他		其他	自行设计

6、自行设计合理的测试用例和验证方法；选做内容：采取合适的方法，绘制出上图。

实验 3.2 树和图的应用

实验 3.2.1 二叉树的重构和遍历

1. 问题描述

根据给定的一颗二叉树的后序遍历和中序遍历结果，输出该二叉树的前序遍历和层次遍历结果。

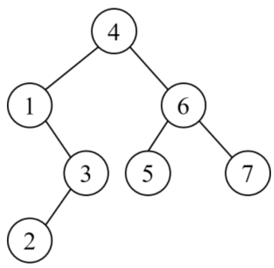
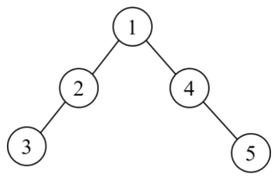
2. 实验要求

（1）输入说明：第一行给出正整数 N（<=30），表示二叉树中结点的个数。随后两行，分别给出 N 个整数（用空格隔开），对应后序遍历和中序遍历的结果。输入数据应能保证正确对应一颗二叉树。

（2）输出说明：在新的一行中，输出“PreOrder: ” 以及该二叉树的前序遍历结果；另起一行，输出“LvlOrder: ” 以及该二叉树的层次遍历结果。数字中间用一个空格隔开，行末不得有多余的空格。

（3）使用链式存储实现。

3. 测试用例

序号	输入	输出	说明
1	7 2 3 1 5 7 6 4 1 2 3 4 5 6 7	PreOrder: 4 1 3 2 6 5 7 LvlOrder: 4 1 6 3 4 7 2 二叉树的形状（可选）： 	一般情况
2	5 3 2 5 4 1 3 2 1 4 5	PreOrder: 1 2 3 4 5 LvlOrder: 1 2 4 3 5 二叉树的形状（可选）： 	单边张开
3	略	略	自定义

4. 提示

（1）重构二叉树

可以将给定的后序遍历和中序遍历分别存放在两个辅助数组 int PostOrder[Max]和 int InOrder[Max]中，通过给定的后序遍历和中序遍历重构二叉树。

后序遍历的最后一个结点为根结点，然后在中序遍历序列中找出根结点的位置（记为 p，数组下标从 0 开始使用），该根结点左边的子序列 InOrderp[0]~ InOrderp[p-1]是其左子树的中序遍历结果，右边的子序列 InOrderp[p+1]~ InOrderp[N-1]是右子树的中序遍历结果。

同理可以推理出 PostOrder[0]~ PostOrder[p-1]为根结点左子树的后续遍历结果；PostOrder[p]~ PostOrder[N-1] 为根结点右子树的后续遍历结果。

在构建二叉树的过程中，必须注意递归终止的条件。

```
/*根据中序和后序数组中的 N 个结点重构二叉树（参考程序）。使用教材 P66 二叉树的
二叉链表定义，data 可定义为 int*/
BinTree BuildTree(int InOrder[], int PostOrder[], int N)
{
    BinTree T;//等价于 BiTNode * T;
    int p;    //用于记录当前根结点的位置。

    if(N==0) return NULL;   //递归终止条件:空树。

    T=( BiTNode *)malloc(sizeof(BiTNode));
    T->data= PostOrder[N-1];   //从后序遍历结果中找到根节点。
    T->lchild=NULL;
    T->rchild=NULL;

    //在中序遍历的结果 InOrder 中找到根结点存储位置，即下标值，需要自行完成
    for(p=0; p<N; p++)
    {  …  }

    //递归重构左子树
    T->lchild=BuildTree(InOrder, PostOrder, p);
    //递归重构右子树
    T->rchild=BuildTree(InOrder+p+1, PostOrder+p, N-p-1);
}
```

（2）根据要求输出二叉树的前序遍历和层次遍历结果。

5. 自行设计合理的交互界面

6. 选作内容

采取合适的方法，绘制出这棵二叉树。

实验 3.2.2 列出图的所有连通分量

1. 问题描述

给定一个具有 n 个顶点、e 条边的无向图 G，使用深度优先遍历（DFS）和广度优先遍历（BFS）的方法，分别给出图 G 的所有连通分量（即最大连通子图）。假设顶点从 0~n-1 编号。遍历时候，总是从编号最小的顶点出发，按照编号递增的顺序访问邻接顶点。

2. 基本要求

（1）输入说明：输入第一行给出 2 个整数：图的顶点数 n（$0<n\le 10$）和边数 e。随后 e 行，每行给出一条边的两个顶点（用空格分开）。

（2）输出说明：按照“$\{v_1, v_2, \dots, v_k\}$”的格式，每行输出一个连通分量对应的顶点集合。先输出 DFS 结果，再输出 BFS 结果。

3. 测试用例

序号	输入	输出	说明
1	8 6 0 7 0 1 2 0 4 1 2 4 3 5	DFS: {0 1 4 2 7} {3 5} {6} BFS: {0 1 2 7 4} {3 5} {6}	遍历方法不同，结果有不同顺序
2	10 9 9 6 6 1 4 6 1 4 8 2 3 7 8 5 2 3 2 5	DFS: {0} {1 4 6 9} {2 3 7 5 8} BFS: {0} {1 4 6 9} {2 3 5 8 7}	第 1 个顶点是单独的连通分量。N 为最大值。

4. 提示

（1）对于一个图，从任意一个顶点出发，无论是 DFS 还是 BFS 遍历，完成一次遍历后访问过的顶点集合是相同的，只是访问的顺序可能不同。即，从编号为 0 的顶点开始，每完成一次遍历就能输出一个连通分量的所有顶点。通过一个循环结构，就能列出图的所有连通分量。

（2）由于题目中图的顶点个数较少，邻接矩阵表示方法是首选。从矩阵的下标 0 开始搜索邻接点，可以保证“按编号递增的顺序访问邻接点”。注意输出的格式要求。

5. 选作内容

使用邻接表来存储图，注意满足“按编号递增的顺序访问邻接点”这一要求。

实验 3.3 管道铺设施工的最佳方案问题

1. 问题描述

需要在某个城市 n 个居民小区之间铺设煤气管道，则在这 n 个居民小区之间只需要铺设 n-1 条管道即可。假设任意两个小区之间都可以铺设管道，但由于地理环境不同，所需要的费用也不尽相同。选择最优的方案能使总投资尽可能小，这个问题即为求无向网的最小生成树。

2. 基本要求

在可能假设的 m 条管道中，选取 n-1 条管道，使得既能连通 n 个小区，又能使总投资最小。每条管道的费用以网中该边的权值形式给出，网的存储采用**邻接矩阵**或**邻接表**的结构。

3. 测试数据

使用下图给出的无向网图的数据作为程序的输入，求出最佳铺设方案。右侧是给出的参考解。

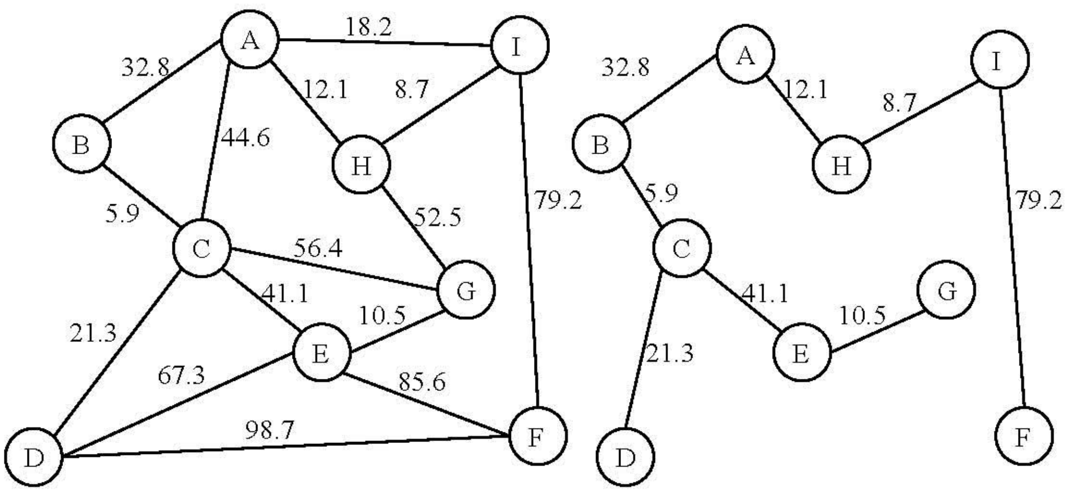


图 1.3.1 小区煤气管道铺设网及其参考解

4. 输入输出:

从键盘或文件读入上图中的无向网，以顶点对（i, j）的形式输出最小生成树的边。文件读入的使用方法可参考（https://cplusplus.com/reference/cstdio/fscanf/）。

5. 提示

连通图 G 的生成树，是 G 的包含其全部 n 个顶点的一个极小连通子图。它必定包含且仅包含 G 的(n-1)条边。在一个连通网图（边带有权值）的所有生成树中，各边的代价之和最小的那棵生成树，称为该连通网的**最小代价生成树（Minimum-cost Spanning Tree）**，简称最小生成树。

（1）MST 性质

构造最小生成树算法有多种，它们一般利用了最小生成树的一种简称为 **MST 的性质**：设 G = <V，E>是一个连通的带权图，其中每条边(v_i, v_j)上带有权 W(v_i, v_j)；集合 U 是顶点集 V 的一个非空真子集，则构建生成树时需要一条边连通顶点集合 U 和 V-U；若有一条边(u_i, v_j)∈E 是一条具有最小权值的边，其中 u_i∈U，v_j∈V-U，那么一定存在一棵包含边(u_i, v_j)的最小生成树。

普里姆（Prim）算法和克鲁斯卡尔（Kruskal）算法是两种利用 MST 性质构造最小生成树的算法。

（2）普里姆（Prim）算法

Prim 算法的构造过程如下：假设 G = <V，E>是一个连通的带权图，TE 是图 G 的最小生成树中对应边的集合。

1）取任意一个图 G 的顶点 u_0∈V，使得顶点集合 U={ u_0}，此时 TE={}；

2）在所有满足 u_i∈U，v_j∈V-U 的(u_i, v_j)∈E 中，找到一条权值最小的边(u_i, v_j)，则这条权值最小的边(u_i, v_j)即为最小生成树的一条边。将边(u_i, v_j)并入集合 TE，同时将顶点 v_j 并入顶点集合 U。

3）重复步骤 2)，直到 U=V 为止，即获得图 G 的最小生成树，此时 TE 有 n-1 条边。

为了实现 Prim 算法，可以设置一个结构体类型的辅助数组 CloseEdge，用于记录从顶点集 U 到顶点集 V-U 的最小权值边。

```
//辅助数组的类型定义，用于记录从顶点集(V-U)中各顶点 vj 到顶点集 U 的最小权值边
typedef struct closeedge
{
   VertexType adjvex; //最小权值边在集合 U 中的邻接顶点
   EdgeType LwstCost; //最小权值。当顶点 vj 并入到 U 之后，其值为 0。
                      //权值边不存在时，LwstCost 可设置为无穷大（如
```

```
65535）
}CloseEdge[MaxVNum];
```

对于每一个顶点 v_j∈V-U，它在辅助数组中存在一个对应的 CloseEdge[j] 分量，其包含两个成员变量：adjvex 和 LwstCost，如 CloseEdge 的定义所示。其中，CloseEdge[j].LwstCost = Min{cost(u_i, v_j)| u_i∈U }。

```
/*Prim 算法描述，使用教材 P81 图的邻接矩阵表示 MGraph 的定义，边的权值类型可设为 float。*/
Void MinSpanTree_Prim(MGraph *G, VertexType u)
{
/*使用邻接矩阵存储图 G，从顶点 u 开始，生成图 G 的最小生成树 T，输出 T 对应的各条边。*/
   k=LocateVex(G, u);   //获得顶点 u 在数组 vexs 中的下标 k，需自行实现

   /* 对 V-U 中的每个顶点 vj，初始化 CloseEdge[j]*/
   for(j=0; j<G->n; j++)
   {
      if (j!=k)
      {
         CloseEdge[j].adjvex=u;
         CloseEdge[j].LwstCost=G->edges[k][j]};
      }
   }
   CloseEdge[j].LwstCost=0;   //最初，U={u0}

   /*依次生成最小生成树的 n-1 条边，将对应 n-1 个顶点添加到集合 U*/
   for(i=1; i<G->n; i++)
   {
   /*找出最小生成树 T 的下一个节点：即 G 中的下标为 k 的顶点，其对应的最小权值边 CloseEdge[k].LwstCost 值最小。需自行实现*/
      k=Min(CloseEdge);
      u0= CloseEdge[k].adjvex;    //所找到的最小权值边的顶点 u0∈U
      v0=G->vexs[k];              //所找到的最小权值边的另一顶点 v0∈U-V
      visit(u0, v0);              //输出所找到的最小权值边，需自行实现

      CloseEdge[k].LwstCost=0;   //G 中下表为 k 的顶点并入 U

      /*U 增加一个顶点后，更新 U-V 中顶点的最小权值边*/
      for(j=0; j<G->n; j++)
      {
      if (G->[k][j]< CloseEdge[j].LwstCost)
         CloseEdge[j]={G->vexs[k], G->edges[k][j]};//结构体变量赋值
      }
   }
}
```

（3）克鲁斯卡尔（Kruskal）算法

Kruskal 算法的构造过程如下：假设 G = <V，E>是一个连通的带权图，将图 G 中的边按照权值从小到大排序。

1）初始状态时，图的最小生成树是一个只有 n 个顶点而无边的非连通图 T=(V, TE)，其中树中对应边的集合 TE={}；

2）在图 G 的边集合 E 中，选择权值最小的边，若该边依附的两个顶点分别属于 T 的不同的连通分量上，则将此边并入 TE，否则舍去此边，继续选择下一条权值最小的边，进行上述判断。

3）重复步骤 2)，直到 T 中所有顶点都属于同一连通分量，嗯嗯。即获得图 G 的最小生成树，此时 TE 有 n-1 条边。

为了实现 Kruskal 算法，可以设置一个结构体辅助数组 Edge，用于存储图的边信息；另外定义一个辅助数组 VexSet，用于存放各个顶点所属的连通分量。它们的定义如下：

```
//辅助数组 Edge 的类型定义
typedef struct edge
{
   VertexType Head; //边的一端顶点
   VertexType Tail; //边的另一端顶点。
   EdgeType EdgeCost;// 边的权重。。
}Edge[MaxEdgeNum];

/*辅助数组 VexSet 的类型定义。初始时，VexSet[i]=i，表示各个顶点各自属于不同的连通分量。*/
int VexSet[MaxVexNum]; 嗯
```



```
/*Kruskal 算法描述，使用教材 P81 图的邻接矩阵表示 MGraph 的定义，边的权值类型可设为 float。*/
Void MinSpanTree_Kruskal(MGraph *G)
{//使用邻接矩阵存储图 G，生成图 G 的最小生成树 T，输出 T 对应的各条边。
   Sort(&Edge);   //对数组 Edge 中的元素按照权值从小到大排序，需自行实现

   /*初始化 VexSet*/
   for(i=0; i<G->n; i++)
      VexSet[i]=i;

   /*依次检查数组 Edge 中的边，是否为最小生成树的边*/
   for(i=0; i<G->e; i++)
   {
      //分别获得当前边邻接的两个顶点的下标。
```

```
        k1=LocateVex(G, Edge[i].Head);
        k2=LocateVex(G, Edge[i].Tail);


        //若该边的两个顶点分别属于不同的连通分量，则并入集合 TE
        SetOne=VexSet[k1];
        SetTwo=VexSet[k2];
        if(SetOne!= SetTwo)
        {
            //输出所找到的边，需自行实现
            visit(Edge[i].Head, Edge[i].Tail);

            //合并该边两个顶点对应的连通分量。
            for (j=0; j<G->n; j++)
                if(VexSet[j]==SetTwo)
                    VexSet[j]=SetOne;
        }
    }
}
```

6. 选做内容

使用图的邻接矩阵和邻接表两种存储结构来实现问题求解。