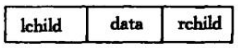
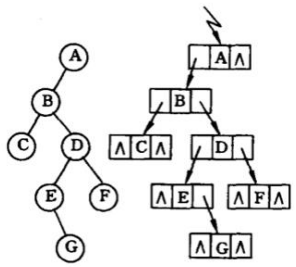
二叉树的二叉链表表示法

1.物理数据结构的概述

二叉树的链表中的结点至少包含 3 个域：数据域和左、右指针域。其中，data 域存放某结点的数据信息；lchild 与 rchild 分别存放指向左孩子和右孩子的指针，当左孩子或右孩子不存在时，相应指针域值为空（用符号∧或 NULL 表示）。利用这样的结点结构表示的二叉树的链式存储结构被称为二叉链表标识法。

结点结构	二叉链表
	

2.物理数据结构的定义

```
public class BinaryTreeNode {
    private int data;//数据
    private BinaryTreeNode leftChild;//左孩子
    private BinaryTreeNode rightChild;//右孩子
    }

public class BinaryTree {
    private BinaryTreeNode root;
    public void setRoot(BinaryTreeNode root)
    public BinaryTreeNode getRoot()
    public void clear(BinaryTreeNode node)
    public boolean isEmpty()
    public int heigh(BinaryTreeNode node)
    public void insert (BinaryTreeNode parent,BinaryTreeNode newNode)

}
```

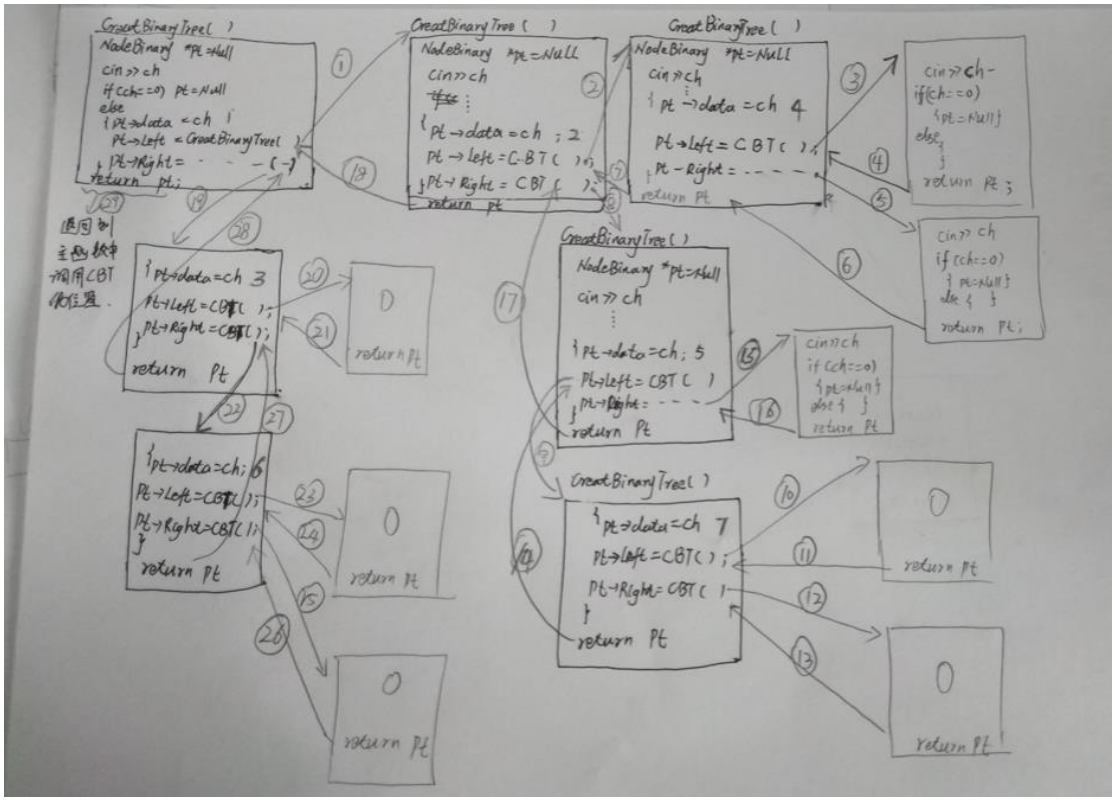
3.如何存

二叉链表的节点值存储二叉树中的节点值，二叉链表指向其两个子节点的指针存贮二叉树根节点与子节点之间的关系。这样使用链式结构节存储下了二叉树的所有数据与关系。

4.如何构建（构建算法）

4.1 举例

假设该二叉树前序遍历为 1 2 4 # # 5 7 # # # 3 # 6 # #(#代表空节点)构建此二叉树的步骤如下。



4.2 算法描述

第一步：输入值，当值不为#时,然后将该值赋给第一个节点的 pt->data，若为#则直接返回 NULL。

第二步：然后设置它的左子节点为递归调用 Creat ()函数的返回值。

第三步：然后设置它的右子节点为递归调用 Creat ()函数的返回值。

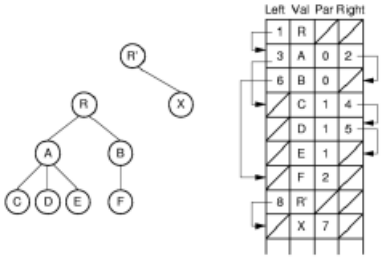
第四步：操作执行完毕，返回该节点地址

第五步：重复执行一到四步，直至所有节点色湖之完毕，构建完成。

二叉树的左子结点/右兄弟结点表示法

1.物理数据结构的概述

每个结点存储节点的值以及指向父结点、最左子节点，右兄弟节点的下标。



2.物理数据结构的定义

```
class BinNode{
    private:
        char data;
        int parent;
        int lc;
        int rightbro;
}
class BinaryTree {
    private
        BinaryTreeNode root;
    Public
        void setRoot(BinaryTreeNode root)
        BinaryTreeNode getRoot()
        void clear(BinaryTreeNode node)
        bool isEmpty()
        int heigh(BinaryTreeNode node)
        void insert (BinaryTreeNode parent,BinaryTreeNode newNode)
        }
```

3.如何存

二叉树的节点值存储二叉树中的节点值，节点的存储其亲节点、左子节点、右兄弟节点的下标，代表他们的关系。这样构建起了二叉树中的数据关系。完成了二叉树的存储。

4.如何构建（构建算法）

4.1 举例

假设该二叉树前序遍历为 1 2 4 # # # 3 # #(#代表空节点)构建此二叉树的步骤如下。
第一步：创建一个 4 行的矩阵 BIN，用于存储 node
第二步：新建一个 node，存储 1，该 node 为 BIN[0]
第三步：新建一个 node，存储 2，该 node 为 BIN[1]
第四步：新建一个 node，存储 4，该 node 为 BIN[2]
第五步：返回-1
第六步：BIN[2]的左子节点下标设为-1
第七步：设置 BIN[2]的父节点下标为 1
第八步：返回 2，设置 BIN[1]的左子节点下标为 2
第九步：新建 Node 返回-1
第十步：设置 BIN[2]的右兄弟节点为-1
第十一步：返回 1，设置 BIN[0]的左子节点下标为 1
第十二步：新建一个 node，存储 4，该 node 为 BIN[3]
第十三步：返回-1
第十四步：设置 BIN[3]的左子节点下标为-1
第十五步：设置 BIN[3]的右兄弟节点下标为-1

第十六步：设置 BIN[3]的父节点为 0

第十七步：返回 3，设置 BIN[1]的右兄弟节点下标为 3

第十八步：设置 BIN[0]的父节点与右兄弟节点下标为-1

第十九步：构建完成

4.2 算法描述

1、设置该节点的的值，若为#则返回-1;

2、递归调用 creat 函数，将该函数的返回值设为它左子节点的值;

3、若左子节点的下标为-1，则不执行后序操作。否则设置递归调用 creat 函数，将该函数的返回值设为它左子节点的右兄弟结点的下标;

4、，设置其父节点下标为调用这个 creat 函数的元素的下标;

5、返回该节点的下标;

6、重复 1-4 步，直至所有节点建立完毕，完成该树的建立。

无向带权标号图的邻接表表示法

1.物理数据结构的概述

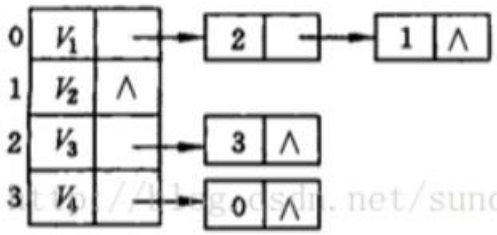
邻接表(Adjacency List)是图的一种顺序存储与链式存储结合的存储方法。邻接表表示法类似于树的孩子链表表示法。就是对于图 G 中的每个顶点 vi，将所有邻接于 vi 的顶点 vj 链成一个单链表，这个单链表就称为顶点 vi 的邻接表，再将所有点的邻接表表头放到数组中，就构成了图的邻接表。在邻接表表示中有两种结点结构。

邻接表是一个包含 |V| 个元素的顺序表

每个元素包含两域，一存储图的顶点信息，另一个是指针，指向该顶点的边构成链表

这个链表通过存储顶点的邻接来示对应边，链表中的每个结点存储顶邻接在顺序链表中的每个结点存储顶邻接在顺序的位置

如果要存储边的权值，则链表中结点增加一个数据域。



2.物理数据结构的定义

```
Class Node {
    VRType adj; //点的下标
    InfoType *info;     //附加信息指针
}
class Graph {
private:
  void operator=(const Graph&) {}        // Protect assignment
  Graph(constGraph&) {}                  // Protect copy constructor
public:
```

```
    Graph(){}                  // Default constructor 默认构造函数
    virtual~Graph() {} // Basedestructor  析构函数

    // Initialize a graph of nvertices  初始化一有 n 个顶点的图
    virtual voidInit(int n) =0;

    // Return: the number of verticesand edges  返回图的顶点数、边数
    virtual intn() =0;
    virtual inte() =0;

    // Return v's first neighbor  返回顶点 v 的第一个邻居
    virtual intfirst(int v) =0;

    // Return v's next neighbor  返回在 w 点之后的邻居（与物理存储中的存放位置有关）
    virtual intnext(int v, int w) =0;

    //设置图的类型（有向图或无向图）
virtualvoid setType(bool flag)=0;
//获取图的类型
virtualbool getType()=0;
//找到(包含实际信息的）顶点在图中的位置
    virtual intlocateVex(VertexType u) =0;
    //返回某个顶点的值(实际信息)
    virtualVertexType getVex(int v)=0;
    //给某个顶点赋值
virtualvoid putVex(int v,VertexType value) =0;

    // Set the weight for an edge  为边(v1,v2)设置权值
    virtual voidsetEdge(int v1, int v2, int wght) =0;

    // Delete an edge 删除边(v1,v2)
    virtual voiddelEdge(int v1, int v2) =0;

    // Determine if an edge is in the graph  判断边(i,j)是否在图中
    virtual boolisEdge(int i, int j) =0;

    // Return an edge's weight  返回边的权值
    virtual intweight(int v1, int v2) =0;

    // Get and Set the mark value fora vertex  取得和设置顶点的标志位
    virtual intgetMark(int v) =0;
    virtual voidsetMark(int v, int val) =0;
};
```

3.如何存

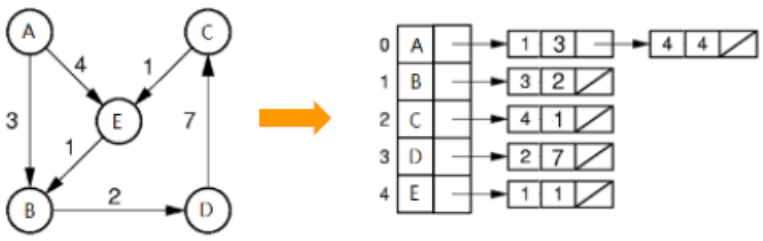
邻接表的 node 分为两部分，数据域与指针域。

每个邻接表第一列的数据域存储该点的数据，然后它指向的节点的数据域以该点为起点的弧的终点的下标以及权值，指针域存贮第二条弧的有关信息，直至把所有的从该点出发弧都存完了。

4.如何构建（构建算法）

4.1 举例

构建该图



第一步：新建一个 5 行的 node 数组 map
第二步：将第 0-4 行的数据域分别填上 A，B，C，D，E 存储该节点的值
第三步：找到以 A 为起点的弧
第四步：新建弧结点，数据域存储 1，3，分别代表指向下标与权值，指向下一节点
第五步：新建弧结点，数据域存储 4，3，分别代表指向下标与权值，指向空
第六步：以 A 为起点的弧链表构建完毕，将数组 map[0]的指针指向它
第七步：新建弧结点，数据域存储 3，2，分别代表指向下标与权值，指向空
第八步：以 B 为起点的弧链表构建完毕，将数组 map[1]的指针指向它
第九步：新建弧结点，数据域存储 4，1，分别代表指向下标与权值，指向空
第十步：以 C 为起点的弧链表构建完毕，将数组 map[2]的指针指向它
第十一步：新建弧结点，数据域存储 2，7，分别代表指向下标与权值，指向空
第十二步：以 D 为起点的弧链表构建完毕，将数组 map[3]的指针指向它
第十三步：新建弧结点，数据域存储 1，1，分别代表指向下标与权值，指向空
第十四步：以 E 为起点的弧链表构建完毕，将数组 map[4]的指针指向它
第十五步：图构建完毕

4.2 算法描述

1、新建一个 node 数组，存储入所有的节点的值
2、将以某个节点为起点的弧构建弧结点，并用链表连接起来
3、将 node 数组的每一行，指向以它为起点的弧所形成的链表
4、构建完毕

无向带权标号图的邻接矩阵表示法

1.物理数据结构的概述

–也称为邻接矩阵或二维数组表示法
–图的顶点元素是一个 |V| 的顺序表
–图的相邻矩阵是一个 |V| ×|V| 矩阵
–如果从 vi 到 vj 存在一条边 ，则对第 i 行的第 j 个 元素做标记 ，否则不做标记
–如果矩阵中的元素要存储边权值，则阵的每个元素必须足够大（存储权值），或者存储一个指向权值的指针

2.物理数据结构的定义

```
template<typename T>
class map_matirx:public Graph
{
public:
    map_matirx(int num);
    ~map_matirx();
    int n();
    int e();
    int first(int v);
    int next(int v, int w);
    bool setPoint(int v, T val);
    bool setEdge(int v1, int v2, int wght);
    bool delEdge(int v1, int v2);
    bool isEdge(int i, int j);
    int weight(int v1, int v2);
    bool setMark(int v, bool val);
    bool getMark(int v);
    int findPoint(T v);
    T getPoint(int v);
private:
    int numPoint, numEdge;
    T* point;
    bool* mark;
    int** matirx;
};
```

3.如何存

一维数组存储每个顶点的信息。

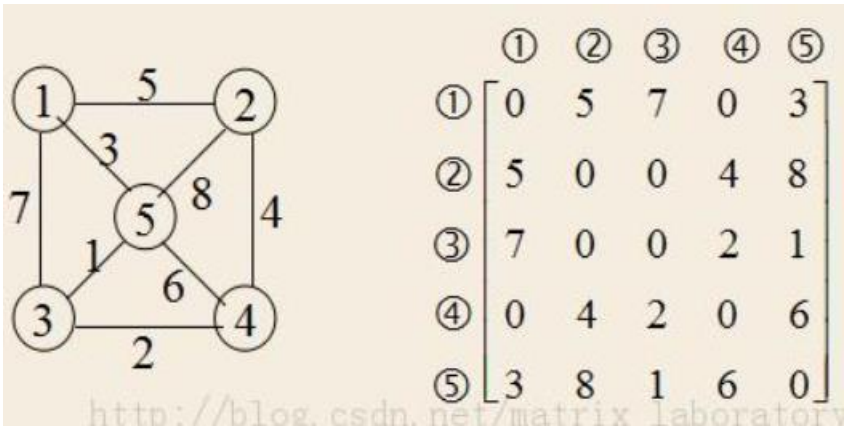
二维数组行和列分别代表同下标的节点。某行某列的值代表这两个点之间的权值，如果为 0（或无穷，看自己定义）代表这两个点之间未相连。

这样就存完了一个图的所有数据与数据间的关系。

4.如何构建（构建算法）

4.1 举例

构建左边的图



第一步：创建一个 5 行的一位数组以及一个 5*5 的二维数组

第二步：将所有的点的信息依次存入一维数组

第三步：将与 1 相连的顶点，在二维矩阵第一行中的相应的列数填入权值，未连接的存入 0

第四步：重复第三步，完成所有边的存储

第五步：完成

4.2 算法描述

1、创建一个 5 行的一位数组以及一个 5*5 的二维数组

2、将所有的点的信息依次存入一维数组

3、将每个点连接的边的权值，存入相应的二维数组中的位置

4、完成

参考文献

[1] buaa_shang，https://blog.csdn.net/buaa_shang/article/details/10399923 [DB/OL]

[2] Nolan kang，https://blog.csdn.net/qq_43739582/article/details/89400268[DB/OL]