湖南大學

数据结构

讨论课课前资料

题　　目：　第七次讨论课（第十四周）

学生姓名：　魏子铖

学生学号：　201726010308

专业班级：　软件 1703

完成时间：　2018.12.19

抽象数据结构的物理实现是数据结构的重点和难点，尤其是非线性结构。如何针对特定的物理数据结构实现方式，把抽象数据结构中的数据和数据关系存储到物理存储结构中，是数据结构学习难点，但它也是应用的前提。

一、二叉树实现之二叉链表

- 物理数据结构概述

二叉树一般使用链式存储结构，由二叉树的定义可知，二叉树的结点由一个数据元素和分别指向其左右孩子的 指针构成，即二叉树的链表结点中包含 3 个域，这种结点结构的二叉树存储结构称之为二叉链表。

- 物理数据结构定义

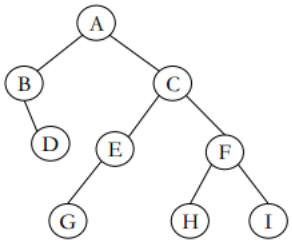
```
class Node{
    public:
            char data;
            Node *lt;
            Node *rt;
 }
class Btree:public Node{
private:
          Node *root;
}
```

- 如何存储

首先创建一个结点类用于储存元素值、左孩子指针和右孩子指针。并创建一个树类，成员是结点类型的指针，用来表示链表的头结点。创建可以根据先序遍历输入结点的顺序建立。

- 存储算法

这里采用顺序表示法（层次遍历）输入，如下图的树 ABC/DEF////G/HI



算法思想：根据输入的顺序确定二叉树，基于递归的思想，如果接收到"/"，则当前节点为空，如果不为空，则新建结点，将数据域存为结点值，并设置左子节点和右子节点递归。

代码如下：

```
Node* Btree::create(char a[])
{
    Node* rt = NULL;
    if(a[count] == '#')
    {
        return rt;
    }
    if(a[count] == '/')
    {
        count++;
        rt = NULL;
    }
    else
    {
        rt = new Node;
        rt->data = a[count];
        count++;
        rt->lt = create(a);
        rt->rt = create(a);
    }
    root = rt;
    return rt;
}
```

二、二叉树实现之左子节点右兄弟表示法

- 物理数据结构概述

二叉树也可使用顺序存储结构，即采用数组存储，因为要存储左子节点下标和右兄弟下标及数据，所以一般采用结构体数组表示。

- 物理数据结构定义

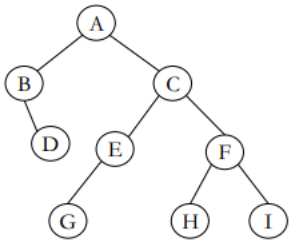
```
class BinNode
 {
    private:
        E it;//数据域
        int l;//最左儿子结点下标
        int r;//右邻兄弟结点下标
template<typename E>   class BinTree {
private:
    int size;//能存储最大的结点个数
    BinNode<E>* T;//存储树结点的数组
    int *D;//存储各结点的深度
    int height;//树的高度
}
```

- 如何存储

首先创建一个结点类用于储存元素值、左子结点下标和右兄弟结点下标。并创建一个树类，成员是结点类型的指针，用来表示数组的首地址。创建可以根据先序遍历输入结点的顺序建立。

- 存储算法

这里采用顺序表示法（层次遍历）输入，如下图的树 ABC/DEF////G/HI



算法思想：根据输入的顺序确定二叉树，首先把数组的数据域全部初始化为'0'，设一计数变量为 0，表示结构体数组的下标。接收的结点如果为'/'，则将数组当前元素的数据域设为'#'，当前元素的左子节点为计数变量乘 2+1，当前元素的右兄弟节点为计数变量+1，计数变量加一；如果接收的节点不为'/'，则在线性表相应的位置存储结点数据，当前元素的左子节点为计数变量乘 2+1，当前元素的右兄弟节点为计数变量+1，计数变量加一。

代码如下：

```
B_tree::B_tree(const char* a)
{
    arr = new Node[MAXSIZE];
    for(int j = 0;j < MAXSIZE;j ++)
    {
        arr[j].data = ' ';
        arr[j].left = 0;
        arr[j].right = 0;
    }
    int i = 1;
    while(a[i] != '#')
    {
        if(a[i] == '/')
        {
            arr[counts].data = '#';
            arr[counts].left = 2*counts+1;
            arr[counts].right = counts+1;
            counts++;
        }
        else
        {
            arr[counts].data = a[i];
            arr[counts].left = 2*counts+1;
```

```
                arr[counts].right = counts+1;
                counts++;
            }
            i++;
        }
    }
```

最后形成的数组：

下标	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
数据	A	B	C	#	D	E	F	#	#	#	#	G	#	H	I	#
左孩子	1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31
右兄弟	/	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

三、无向带权标号图的邻接矩阵表示法

- 物理数据结构概述

  邻接矩阵，是用一个二维数组存储，边使用矩阵来构建模型，这使得每一个顶点和其它顶点之间都有边的有无的表示的机会。若有边，则他们交点为 1 ，否则为 0。当然，如果是一副边有权值的图，交点存储的是他们边的权值。

- 物理数据结构定义

```
class Vertex{
public:
  char data;
};
class GraphM{
private:
  int numVertex, numEdge;
  int **matrix;
  bool *visited;
  bool type = true;
Vertex vexs[MAX_VERTEX_NUM];
 }
```

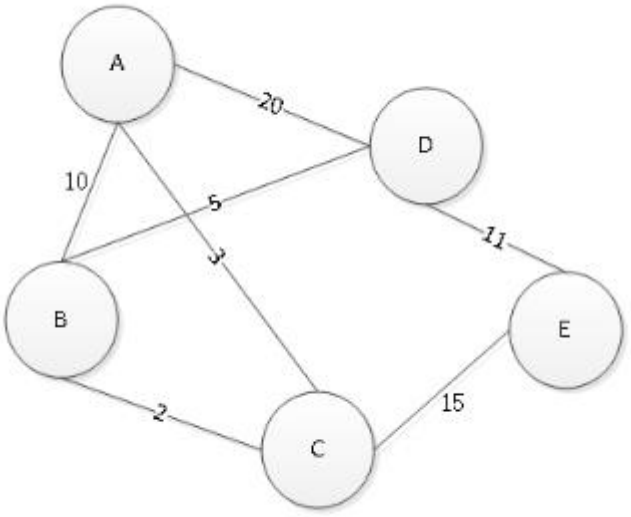
- 如何存储

无向图的边的矩阵一定是一个对称矩阵，因为无向图只关心边是否存在，而不关心方向，V0 和 V1 有边，那么 V1 和 V0 也有边。因为这里不研究有圈图，所以主对角线都是 0，输入 V0 和 V1 边的关系后，就不必输入 V1 和 V0 的关系了。

- 存储算法

算法思想：先输入顶点的大小，并保存在一维数组中，之后输入顶点的编号和权值并保存在二维数组中，由于对称性，再调整另一个方向的值。

例如下图



此图的关系为

A B 10

B D 5

A D 20

A C 3

C B 2

C E 15

```
D E 11
Graph::CreateGraph()
{
    matrix=(int **)new int*[numVertex];
    for(int i=0;i<numVertex;i++)
        matrix[i]=new int[numVertex];
    for(int i=0;i<numVertex;i++)
        for(int j=0;j<numVertex;j++)
            matrix[i][j]=0;
}
Graph::setEdge(int i,int j,int w)
{
    matrix[i][j]=w;
    matrix[j][i]=w;
     numEdge+=2;
}
```

最后的邻接矩阵为

```
0 10 3 20 0
10 0 2 5 0
3 2 0 0 15
20 5 0 0 11
0 0 15 11 0
```

四、无向带权标号图的邻接表表示法

- 物理数据结构概述

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。

- 物理数据结构定义

```
class edge
{
   int vert;
   int weight;
}
class Graph
{
   List<Edge>** vertex;
   int numVertex;
   int numEdge;
   int *mark;
}
```

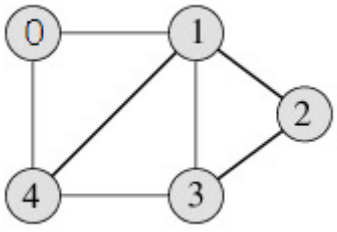
- 如何存储

每个结点包含两个域，一个域存图顶点的信息，一个域是指针域，存储该顶点形成的链表的信息。而邻接表是一个以链表为元素的数组，这个数组包含所有顶点的元素，其中第 i 个元素存储一个指针,指针指向顶点 vi 的边构成的链表，当 vi 指向 vj 有一条边的时候，vj 也有一条指向 vi 的边了。因此要分别为 vi 和 vj 指向的链表增加一个结点。

- 存储算法

先创造一组指针数组，每一个指针指向一条链表，之后输入一组边的顶点与权值，在两个顶点表示的链表各添加一个结点，填入顶点名称与权值。

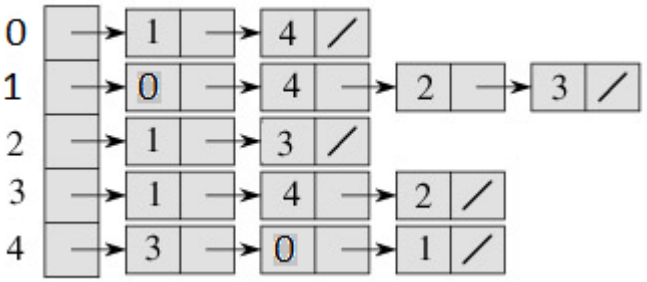
例如下图



代码如下

```
Graph::init()
{
    vertex = (LList<Edge>**) new LList<Edge>*[numVertex];
```

```
    for (i=0; i<numVertex; i++)
        vertex[i] = new LList<Edge>[numVertex];
}
void GraphI::setEdge(int i, int j, int weight) {
    Edge currEdge(j, weight);
    if (isEdge(i, j)) { // Edge already exists in graph
      vertex[i]->remove();
      vertex[i]->insert(currEdge);
    }
    else { // Keep neighbors sorted by vertex index
      numEdge++;
      for (vertex[i]->moveToStart();
            vertex[i]->currPos() < vertex[i]->length();
            vertex[i]->next()) {
        Edge temp = vertex[i]->getValue();
        if (temp.vertex() > j) break;
      }
      vertex[i]->insert(currEdge);
    }
}
```



以上内容参考:

Clifford A.Shaffer Data Structures and Algorithm Analysis in C++ Third Edition,248-253,Oct,2013.

https://mooc1-2.chaoxing.com/mycourse/studentstudy?chapterId=126320603&courseId=201712353&clazzid=4427432&enc=9b7d6fd5da9c2c3485915e1a5da4a74a.

https://zh.wikipedia.org/wiki/%E9%82%BB%E6%8E%A5%E8%A1%A8.