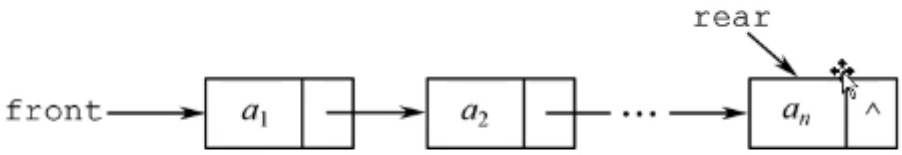
> C语言督学营 ， 学习笔记，龙哥永远的神

@[toc]

第七次直播

队列的链式存储

- 队列的链式表示称为**链队列**，它实际上是一个同时带有`队头指针`和`队尾指针`的单链表。头指针指向队头结点，尾指针指向队尾结点，即单链表的最后一个结点。



链表队列的定义

3047289

```
typedef int ElemType;
typedef struct LinkNode{
        ElemType data;
        struct LinkNode *next;
}LinkNode;  //链表结点的结构体
typedef struct{
    LinkNode *front,*rear;//链表头 链表尾
}LinkQueue;//先进先出

LinkQueue Q;
```

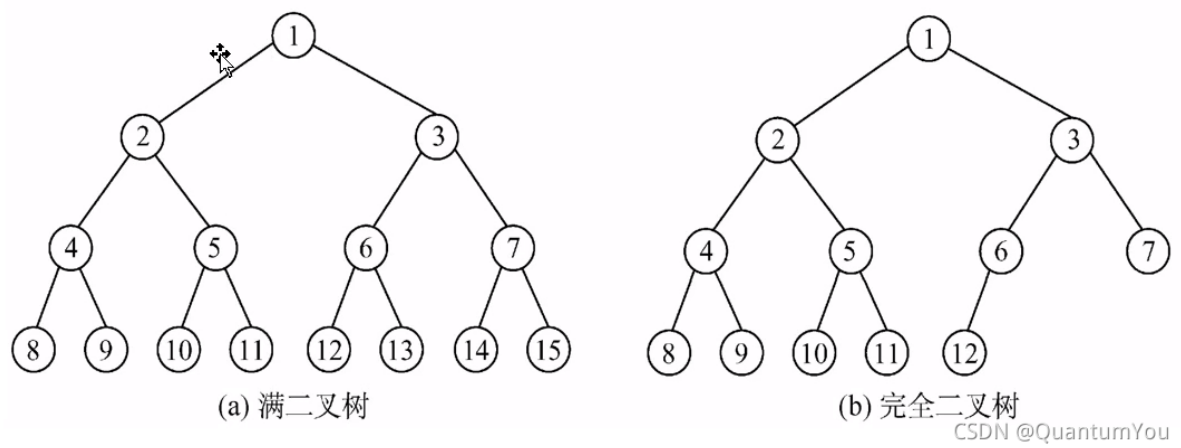
相对于原有编写的链表增加了尾指针

二叉树

- **树的定义**：树是n（n≥0）个节点的有限集。当n=0时，称为空树。在任意一棵非空树中应满足：1）有且仅有一个特定的称为根的结点。2）当n>1时，其余节点可分为m（m>0）个互不相交的有限集T1，T2...，Tm，其中每个集合本身又是一棵树，并且称为根的子树。
- **二叉树的定义**：

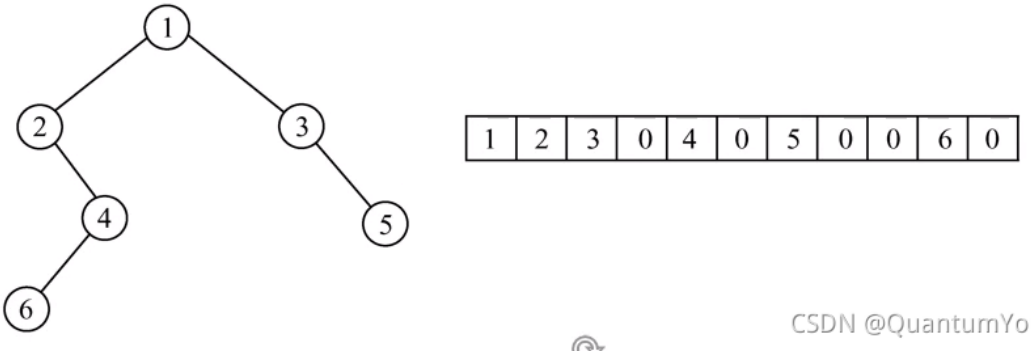
二叉树是另一种树形结构，其特点是每个结点至多只有两棵子树（即二叉树中不存在度大于2的结点），并且二叉树的子树有左右之分，其次序不能任意颠倒。
与树相似，二叉树也以递归的形式定义。二叉树是n（n ≥ 0）个结点的有限集合：
① 或者为空二叉树，即n = 0。
② 或者由一个根结点和两个互不相交的被称为根的左子树和右子树组成。左子树和右子树又分别是一棵二叉树。

满二叉树与完全二叉树

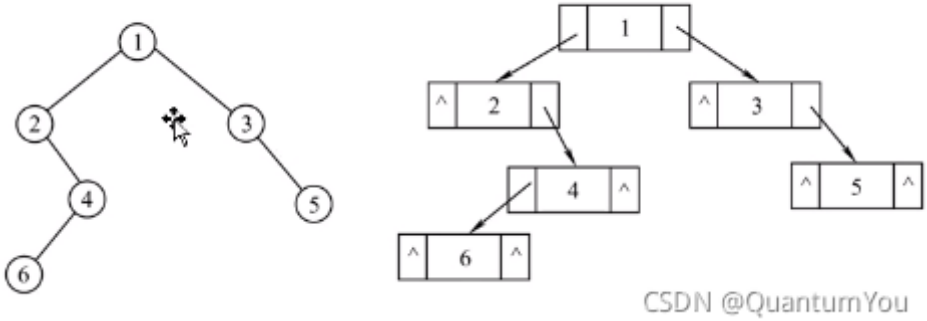


二叉树的存储

顺序存储



链式存储



二叉树的建立

定义

```
typedef char BiElemType;
typedef struct BiTNode{
    BiElemType c;//c就是书籍上的data
    struct BiTNode *lchild;
    struct BiTNode *rchild;
}BiTNode,*BiTree;
```


- 二叉树的建树（层次建树，思考为什么借助队列建树），前序、中序、后序遍历、中序非递归遍历、层次遍历

calloc

> calloc
>
> *语法:*
>
> ```
> #include <stdlib.h>
> void *calloc( size_t num, size_t size );
> ```
>
> 功能： 函数返回一个指向*num* 数组空间，每一数组元素的大小为*size*。如果错误发生返回NULL。

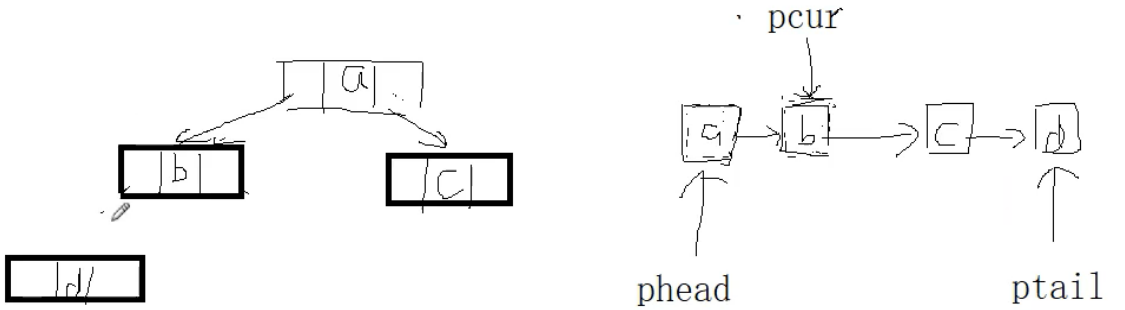
- `calloc`申请空间并对空间进行初始化，赋值为0（填的是二进制的零），好处在于省的 `malloc` 之后需要赋值为 `null`

参考链接

```
BiTree pnew;
    int i,j,pos;
    char c;
    BiTree tree=NULL;//树根
    ptag_t phead=NULL,ptail=NULL,listpnew,pcur;//phead就是队列头，ptail就是队列尾
    //abcdefghij
    while(scanf("%c",&c)!=EOF)
    {
        if(c=='\n')
        {
            break;
        }
        pnew=(BiTree)calloc(1,sizeof(BiTNode));//calloc申请空间并对空间进行初始化，赋值为0
        pnew->c=c;//数据放进去
        listpnew=(ptag_t)calloc(1,sizeof(tag_t));//给队列结点申请空间
        listpnew->p=pnew;
        if(NULL==tree)
        {
            tree=pnew;//树的根
            phead=listpnew;//队列头
```

```
            ptail=listpnew;//队列尾
            pcur=listpnew;
            continue;
        }else{
            ptail->pnext=listpnew;//新结点放入链表，通过尾插法
            ptail=listpnew;//ptail指向队列尾部
        }//pcur始终指向要插入的结点的位置
        if(NULL==pcur->p->lchild)//如何把新结点放入树
        {
            pcur->p->lchild=pnew;//把新结点放到要插入结点的左边
        }else if(NULL==pcur->p->rchild)
        {
            pcur->p->rchild=pnew;//把新结点放到要插入结点的右边
            pcur=pcur->pnext;//左右都放了结点后，pcur指向队列的下一个
        }
```

层序遍历建树的过程：(注意这队列不带头节点)

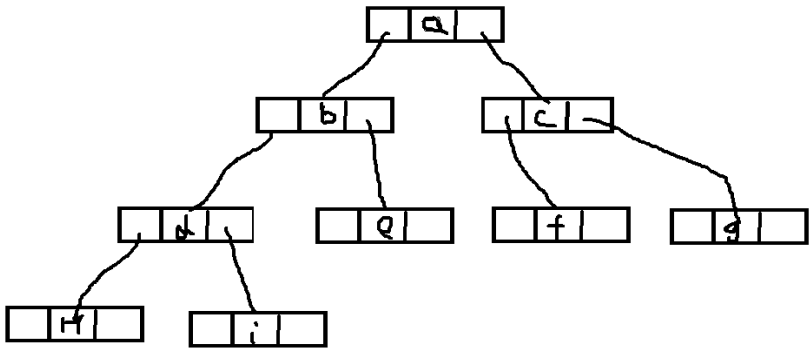


- **注意**：`pcur=pcur->pnext;` 代表指向下一层

可以通过监视窗口理解层序遍历的建树过程如下

- 前序遍历递归打印相对于（非递归）比较简单

龙哥的前序遍历的后方法



龙哥前序遍历方法过程

主体思想：孩子一次在父母旁边

1.abc

2.abdec

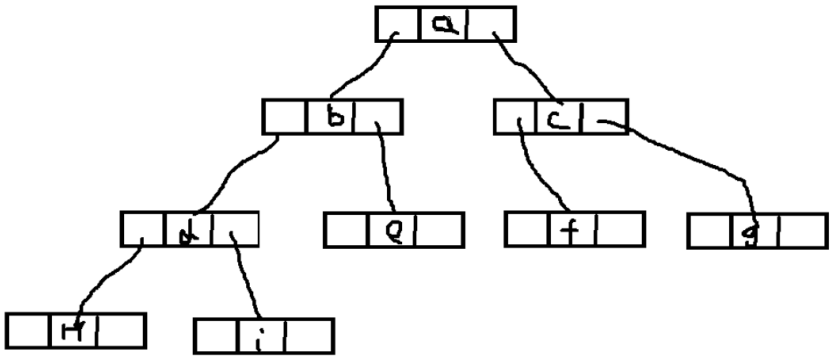
3.abdhiec

4.abdhiecfg

第八次直播

二叉树中序-后序-层序遍历

- 中序遍历



龙哥前序遍历方法过程

主体思想：孩子一次在父母旁边

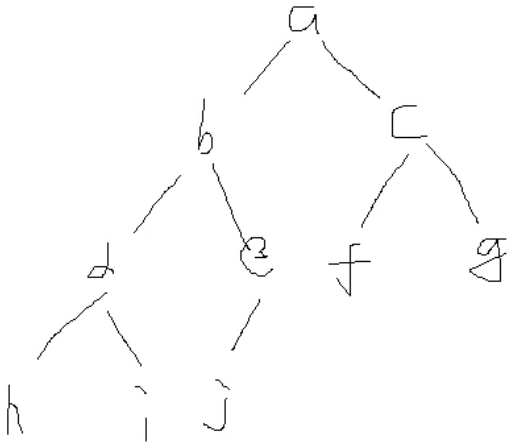
1.bac
2.dbeac
3.hdibeac
4.hdibeafcg

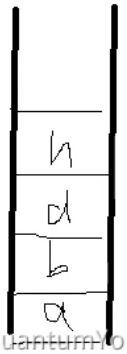
- 后序遍历同理

中序遍历非递归写法 （非重点）

- 主体思路为不断压栈的过程找到了最左边的左孩子

```
void InOrder2(BiTree T)
{
    SqStack S;
    InitStack(S);BiTree p=T;
    while(p||!StackEmpty(S))//逻辑或||
    {
        if(p)
        {
            Push(S,p);
            p=p->lchild;
        }else{
            Pop(S,p);putchar(p->c);
            p=p->rchild;
        }
    }
}
```





- 树的前序遍历就是深度优先遍历

层序遍历

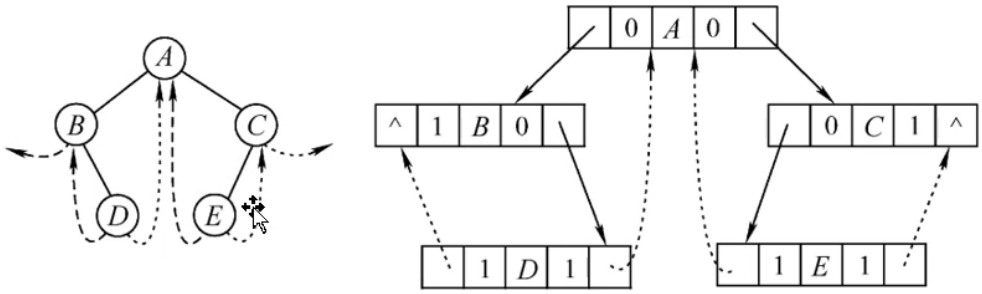
- 层序遍历需要用到辅组队列

```
void LevelOrder(BiTree T)
{
    LinkQueue Q;
    InitQueue(Q);
    BiTree p;
    EnQueue(Q,T);//树根入队
    while(!IsEmpty(Q))
    {
        DeQueue(Q,p);
        putchar(p->c);
        if(p->lchild!=NULL)
            EnQueue(Q,p->lchild);
        if(p->rchild!=NULL)
            EnQueue(Q,p->rchild);
    }
}
```

线索二叉树

- 一般出小题不可能出大题
- 以中序线索二叉树的建立为例。附设指针pre指向刚刚访问过的结点，指针p指向正在访问的结点，即pre指向p的前驱。在中序遍历的过程中，检查p的左指针是否为空，若为空就将它指向pre;检查pre的右指针是否为空，若为空就将它指向p

```
typedef char ElemType;
typedef struct ThreadNode{
    ElemType data;
    struct ThreadNode *lchild,*rchild;
    int ltag,rtag;//相对于二叉树多的
}ThreadNode,*ThreadTree;
```

方法，先把树画出来，然后再线索化

$$ltag\begin{cases}0, & lchild域指示结点的左孩子\\ 1, & lchild域指示结点的前驱\end{cases}$$

$$rtag\begin{cases}0, & rchild域指示结点的右孩子\\ 1, & rchild域指示结点的后继\end{cases}$$

线索二叉树的建立代码

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
//作者 王道训练营 龙哥
typedef char ElemType;
typedef struct ThreadNode{
    ElemType data;
    struct ThreadNode *lchild,*rchild;
    int ltag,rtag;
}ThreadNode,*ThreadTree;
//手工建线索树，总计5个结点
void BulidThreadTree(ThreadTree &T)
{
    ThreadTree arr[5];
    int i;
    for(i=0;i<5;i++)
    {
```

```
        arr[i]=(ThreadTree)malloc(sizeof(ThreadNode));
        memset(arr[i],0,sizeof(ThreadNode));
        arr[i]->data='A'+i;
    }
    arr[0]->lchild=arr[1];
    arr[0]->rchild=arr[2];
    arr[1]->rchild=arr[3];
    arr[2]->lchild=arr[4];
    T=arr[0];
}
void InThread(ThreadTree &p,ThreadTree &pre)
{
    if(p!=NULL){
        InThread(p->lchild,pre);
        if(p->lchild==NULL){//左边为NULL
            p->lchild=pre;
            p->ltag=1;
        }
        if(pre!=NULL&&pre->rchild==NULL){
            //pre节点右孩子为NULL，就让其指向后继节点
            pre->rchild=p;
            pre->rtag=1;
        }
        pre=p;
        InThread(p->rchild,pre);
    }
}
void CreateInThread(ThreadTree T)
{
    ThreadTree pre=NULL;
    if(T!=NULL){
        InThread(T,pre);
        pre->rchild=NULL;
        pre->rtag=1;
    }
}
//中序序列下的第一个结点
ThreadNode *Firstnode(ThreadNode *p)
{
    while(p->ltag==0)
        p=p->lchild;
    return p;
}
//p在中序序列下的后继结点

int main()
{
    ThreadTree T;
    ThreadTree p;
    BulidThreadTree(T);
    CreateInThread(T);//构建线索二叉树
    p=Firstnode(T);
    printf("最左下结点值为 %c\n",p->data);
    system("pause");
}
```

层序遍历的分析过程