* + 1. **二叉树的按层次遍历**

**●先遍历根结点，再遍历根结点的孩子结点，再遍历孩子结点的孩子结点，……。**

**具体如下：**

**(1)设置一个队列，将其置空，若树非空，先将根结点入队；**

**(2)重复下述操作，直至队列为空：**

**●队首结点出队，遍历该结点；**

**●若该结点有左子树，则将左子树的根结点入队；若该结点有右子树，则将右子树的根结点入队。**

**typedef struct**

**{**

**BTree \*s[MAXSIZE];**

**int front,rear;**

**}Sequence; //顺序队列类型**

**void ScanLevel(BTree \*t)**

**//按层次遍历二叉树（非递归）**

**{**

**Sequence que;// 定义队列**

**que.front=que.rear=0;//初始化队列**

**if(t!=NULL)//树非空，将根结点入队**

**{**

**que.rear++;**

**que.s[que.rear]=t;**

**}**

**printf("二叉树的层次结点是：");**

**while(que.front!=que.rear)//队列非空**

**{**

**que.front++;//队头出队**

**t=que.s[que.front];**

**printf("%c ",t🡪data); // 遍历结点**

**if(t🡪lchild!=NULL)//左孩子入队**

**{**

**que.rear++;**

**que.s[que.rear]=t🡪lchild;**

**}**

**if(t🡪rchild!=NULL)//右孩子入队**

**{**

**que.rear++;**

**que.s[que.rear]=t🡪rchild;**

**}**

**}**

**}**

**●根据按层次遍历的思想，创建二叉树;**

**由于二叉树的结点构成比较复杂，创建和遍历时必须有规律才能在机器上实施，这就想到以前讲过的完全二叉树，完全二叉树结点有对应的编号，然后按层次逐个输入各结点的编号和数据，直至结束。若输入非根结点，根据其编号（偶数为左子树结点，奇数为右子树结点），将其链接到其双亲结点的相应指针域上。**

**例如：上述曾经讨论过的二叉树**

**用添加虚结点的办法补成一棵完全二叉树且对结点编号；**

**这里带@的结点是虚结点；除根结点外，左子树结点都是偶数编号，右子树结点都是奇数编号。**

**BTree \*CreateTree()**

**{**

**FILE \*fp;**

**BTree \*t,\*p,\*seq[MAXSIZE];**

**DataType x;**

**int i;**

**fp=fopen("p6-1.dat","r");**

**printf("输入二叉树的层次结点：\n");**

**fscanf(fp,"%d %c",&i,&x);**

**printf("%d %c\n",i,x);**

**if(x!='@')//建立根结点**

**{**

**t=(BTree \*)malloc(sizeof(BTree));**

**t🡪data=x;**

**t🡪lchild=NULL;**

**t🡪rchild=NULL;**

**}**

**else return NULL;**

**seq[i]=t;**

**fscanf(fp,"%d %c",&i,&x);**

**printf("%d %c\n",i,x);**

**while(x!='@')**

**{**

**p=(BTree \*)malloc(sizeof(BTree));**

**p🡪data=x;**

**p🡪lchild=NULL;**

**p🡪rchild=NULL;**

**seq[i]=p;**

**if(i%2==0)seq[i/2]🡪lchild=p;//建立左子树**

**else seq[i/2]🡪rchild=p;//建立右子树**

**fscanf(fp,"%d %c",&i,&x);**

**printf("%d %c\n",i,x);**

**}**

**fclose(fp);printf("\n");**

**return t;**

**}**

**程序p6-1.c 二叉树的层次建立和遍历**

**输入时注意结点的编号（@作为结束标志）**

**输入文件（编号和内容，@ 作为结束标志）：**

**1 A**

**2 B**

**3 C**

**5 D**

**6 E**

**10 F**

**13 G**

**0 @ //作为结束标志**

**#define MAXSIZE 100**

**#include<stdio.h>**

**typedef char DataType;**

**typedef struct node**

**{**

**DataType data;**

**struct node \*lchild,\*rchild;**

**}BTree;**

**typedef struct**

**{**

**BTree \*s[MAXSIZE];**

**int front,rear;**

**}Sequence;**

**BTree \*CreateTree()//二叉树的层次创建**

**{**

**FILE \*fp;**

**BTree \*t,\*p,\*seq[MAXSIZE];**

**DataType x;**

**int i;**

**fp=fopen("p6-1.dat","r");**

**printf("输入二叉树的层次结点：\n");**

**fscanf(fp,"%d %c",&i,&x);**

**printf("%d %c\n",i,x);**

**if(x!='@')**

**{**

**t=(BTree \*)malloc(sizeof(BTree));**

**t🡪data=x;**

**t🡪lchild=NULL;**

**t🡪rchild=NULL;**

**}**

**else return NULL;**

**seq[i]=t;**

**fscanf(fp,"%d %c",&i,&x);**

**printf("%d %c\n",i,x);**

**while(x!='@')**

**{**

**p=(BTree \*)malloc(sizeof(BTree));**

**p🡪data=x;**

**p🡪lchild=NULL;**

**p🡪rchild=NULL;**

**seq[i]=p;**

**if(i%2==0)seq[i/2]🡪lchild=p;**

**else seq[i/2]🡪rchild=p;**

**fscanf(fp,"%d %c",&i,&x);**

**printf("%d %c\n",i,x);**

**}**

**fclose(fp);printf("\n");**

**return t;**

**}**

**void ScanLevel(BTree \*t) //二叉树的层次遍历**

**{**

**Sequence que;**

**que.front=que.rear=0;**

**if(t!=NULL)**

**{**

**que.rear++;**

**que.s[que.rear]=t;**

**}**

**printf("二叉树的层次结点是：");**

**while(que.front!=que.rear)**

**{**

**que.front++;**

**t=que.s[que.front];**

**printf("%c ",t🡪data); // 遍历结点**

**if(t🡪lchild!=NULL)**

**{**

**que.rear++;**

**que.s[que.rear]=t🡪lchild;**

**}**

**if(t🡪rchild!=NULL)**

**{**

**que.rear++;**

**que.s[que.rear]=t🡪rchild;**

**}**

**}**

**}**

**void main()**

**{**

**BTree \*t;**

**t=CreateTree();**

**ScanLevel(t);**

**printf("\n");**

**}**

**6.4.5二叉树遍历的应用**

**通过遍历二叉树，可以输出树中的结点数据，统计和查找结点中的信息等。**

**如统计树中结点数、叶子数、树的深度，查找最大（小）结点的值等。**

**(1)求二叉树的结点数（中序法）**

**int NodeNumber=0;//统计结点个数计数变量**

**void InOrder(BTree \*bt)**

**{**

**if(bt!=NULL)**

**{ InOrder(bt🡪lchild);//遍历左子树**

**NodeNumber++;//遍历根结点（统计结点）**

**InOrder(bt🡪rchild);//遍历右子树**

**}**

**}**

**(2)二叉树的递归建立（先序法）**

**不用层次法而用先序、中序和后序法建立二叉树不一定要按虚补成完全二叉树，但要提供左右子树是否存在（如不存在则用@虚补以示容易判断）。**

**BTree \*CreateTree()//先序法建立**

**{**

**BTree \*t;**

**DataType x;**

**scanf("%c",&x);**

**if(x==’@’) return NULL;**

**else**

**{**

**t=(BTree \*)malloc(sizeof(BTree));**

**//建立根结点**

**t🡪data=x;**

**t🡪lchild=CreateTree();//建立左子树**

**t🡪rchild=CreateTree();//建立右子树**

**return t;**

**}**

**}**

**p6-2.c递归“先序”创建二叉树并用四种方法遍历（先序、中序、后序和按层次）。**

**这棵树不是完全二叉树，但二叉树建立的过程中每个结点要指明它的左右子树是否存在，若不存在补上一个空结点（@）为的是容易存储和判断。**

**输入（先序法）文件：AB@DF@@@CE@G@@@**

**#define MAXSIZE 100**

**#include<stdio.h>**

**typedef char DataType;**

**typedef struct node**

**{**

**DataType data;**

**struct node \*lchild,\*rchild;**

**}BTree;**

**FILE \*fp;**

**BTree \*CreatTree( )//先序递归创建**

**{**

**char x;**

**BTree \*t;**

**x=fgetc(fp);**

**if(x=='@')return NULL;**

**else**

**{**

**t=(BTree \*)malloc(sizeof(BTree));**

**t🡪data=x;**

**t🡪lchild=CreatTree();**

**t🡪rchild=CreatTree();**

**return t;**

**}**

**}**

**void PreOrder(BTree \*bt)//先序递归遍历**

**{**

**if(bt!=NULL)**

**{**

**printf("%3c",bt🡪data);**

**PreOrder(bt🡪lchild);**

**PreOrder(bt🡪rchild);**

**}**

**}**

**void PreOrder1(BTree \*bt)//先序非递归遍历**

**{**

**BTree \*s[100],\*p=bt;**

**int top=0;**

**while(p!=NULL||top>0)**

**{**

**while(p!=NULL)**

**{printf("%3c",p🡪data); s[++top]=p;**

**p=p🡪lchild;}**

**p=s[top--];p=p🡪rchild;**

**}**

**}**

**void InOrder(BTree \*bt)//中序递归遍历**

**{**

**if(bt!=NULL)**

**{**

**InOrder(bt🡪lchild);**

**printf("%3c",bt🡪data);**

**InOrder(bt🡪rchild);**

**}**

**}**

**void InOrder1(BTree \*bt)//中序非递归遍历**

**{**

**BTree \*s[MAXSIZE],\*p=bt;**

**int top=0;**

**while(p!=NULL||top>0)**

**{**

**while(p!=NULL){s[++top]=p;p=p🡪lchild;}**

**p=s[top--];**

**printf("%3c",p🡪data);p=p🡪rchild;**

**}**

**}**

**void PostOrder(BTree \*bt)//后序递归遍历**

**{**

**if(bt!=NULL)**

**{**

**PostOrder(bt🡪lchild);**

**PostOrder(bt🡪rchild);**

**printf("%3c",bt🡪data);**

**}**

**}**

**void PostOrder1(BTree \*bt)//后序非递归遍历**

**{**

**BTree \*s1[MAXSIZE],\*p=bt;**

**int s2[MAXSIZE],b,top=0;**

**do**

**{**

**while(p!=NULL){s1[top]=p;s2[top++]=0;**

**p=p🡪lchild;}**

**if(top>0)**

**{**

**b=s2[--top]; p=s1[top];**

**if(b==0){s1[top]=p;s2[top++]=1;**

**p=p🡪rchild;}**

**else {printf("%3c",p🡪data); p=NULL;}**

**}**

**}while(top>0);**

**}**

**void LevelOrder(BTree \*bt)//层次遍历**

**{**

**BTree \*q[MAXSIZE],\*p;**

**int front,rear;**

**q[1]=bt;front=rear=1;**

**while(front<=rear)**

**{**

**p=q[front];front++;printf("%3c",p🡪data);**

**if(p🡪lchild!=NULL)**

**{rear++;q[rear]=p🡪lchild;}**

**if(p🡪rchild!=NULL)**

**{rear++;q[rear]=p🡪rchild;}**

**}**

**}**

**void main()**

**{**

**BTree \*t;**

**fp=fopen("p6-2.dat","r");**

**printf("输入数据在文件中读出！\n\n");**

**t=CreatTree();**

**fclose(fp);**

**printf("先序递归法遍历： ");**

**PreOrder(t);printf("\n");**

**printf("先序非递归法遍历：");PreOrder1(t);**

**printf("\n\n");**

**printf("中序递归法遍历： ");InOrder(t);**

**printf("\n");**

**printf("中序非递归法遍历：");InOrder1(t);**

**printf("\n\n");**

**printf("后序递归法遍历： ");PostOrder(t);**

**printf("\n");**

**printf("后序非递归法遍历：");PostOrder1(t);**

**printf("\n\n");**

**printf("按层次遍历： ");LevelOrder(t);**

**printf("\n\n");**

**}**

**程序p6-3.c 家族树的建立和统计（层次建立和遍历）**

**（1）该家族的平均年龄；**

**（2）指定的某个家族成员的辈分；**

**（3）输出指定的某个辈分的所有家族成员。**

**按先序输入（其中0为虚结点）：**

**60 景奇(Jingqi)**

**58 宏恩(Hongen)**

**66 新民(Xinmin)**

**43 意海(Yihai)**

**0 XX**

**0 XX**

**0 XX**

**69 新主(Xinzhu)**

**86 意河(Yihe)**

**0 XX**

**0 XX**

**73 意江(Yijiang)**

**0 XX**

**0 XX**

**55 宏亮(Hongliang)**

**88 新诚(Xincheng)**

**55 意凌(Yiling)**

**0 XX**

**0 XX**

**0 XX**

**70 新胜(Xinsheng)**

**66 意山(Yishan)**

**0 XX**

**0 XX**

**66 意水(Yishui)**

**0 XX**

**0 XX**

**#define MAX 100**

**#include<stdio.h>**

**typedef struct**

**{**

**int age;**

**char name[20];**

**}DataType;**

**typedef struct node**

**{**

**DataType data;**

**struct node \*lchild,\*rchild;**

**}BTree;**

**FILE \*fp;**

**int TotalAge,TotalNumber;**

**BTree \*CreateTree();**

**void Traversing(BTree \*t);**

**void PreOrder(BTree \*t);**

**void NodeLevel(BTree \*t,int h,DataType x);**

**void OutPutLevelNode(BTree \*t,int h);**

**void visitstring(BTree \*t,int h);**

**void main()**

**{**

**BTree \*root;**

**DataType x;**

**int h;**

**fp=fopen("p6-3.dat","r");**

**root=CreateTree();**

**fclose(fp);**

**TotalAge=0;**

**TotalNumber=0;**

**PreOrder(root);**

**if(TotalNumber==0)**

**{**

**printf("\n家族为空！\n");**

**exit(0);**

**}**

**printf("一、家族的平均年龄：%.2f\n",**

**(float)TotalAge/TotalNumber);**

**printf("\n二、输入要查找的家族成员的年龄：");**

**scanf("%d",&x.age);**

**printf("年龄为%d的的家族成员：\n姓名 年龄 辈份\n",x.age);**

**NodeLevel(root,1,x);**

**printf("\n三、输入辈份：");**

**scanf("%d",&h);**

**printf("辈份为%d的所有家族的成员：\n姓名 年龄 辈份\n",h);**

**if(h>0)OutPutLevelNode(root,h);**

**printf("\n");**

**}**

**BTree \*CreateTree()//层次创建二叉树**

**{**

**BTree \*root,\*s;**

**DataType pp;**

**root=NULL;**

**fscanf(fp,"%d%s",&pp.age,pp.name);**

**if(pp.age!=0)**

**{**

**s=(BTree \*)malloc(sizeof(BTree));**

**s🡪data=pp;**

**s🡪lchild=CreateTree();**

**s🡪rchild=CreateTree();**

**root=s;**

**}**

**return root;**

**}**

**void Traversing(BTree \*t)//年龄求和**

**{**

**TotalAge+=t🡪data.age;**

**TotalNumber++;**

**}**

**void visitstring(BTree \*t,int h)**

**//输出姓名和年龄**

**{**

**printf("%s%6d%6d\n",**

**t🡪data.name,t🡪data.age,h);**

**}**

**void PreOrder(BTree \*t)//先序查找**

**{**

**if(t!=NULL)**

**{**

**Traversing(t);**

**PreOrder(t🡪lchild);**

**PreOrder(t🡪rchild);**

**}**

**}**

**void NodeLevel(BTree \*t,int h,DataType x)**

**{//层次遍历统计年龄**

**if(t!=NULL)**

**{**

**if(t🡪data.age==x.age)visitstring(t,h);**

**NodeLevel(t🡪lchild,h+1,x);**

**NodeLevel(t🡪rchild,h+1,x);**

**}**

**}**

**void OutPutLevelNode(BTree \*t,int h)**

**{//层次遍历输出**

**BTree \*s[MAX];**

**int i,begin,last,level,num;**

**begin=1;last=1;level=1; num=1;s[last]=t;**

**while(level<h)**

**{**

**for(i=1;i<=num;i++)**

**{**

**if(s[begin]🡪lchild!=NULL)**

**{**

**last++;**

**s[last]=s[begin]🡪lchild;**

**}**

**if(s[begin]🡪rchild!=NULL)**

**{**

**last++;**

**s[last]=s[begin]🡪rchild;**

**}**

**begin++;**

**}**

**num=last-begin+1;**

**level++;**

**}**

**for(i=begin;i<=last;i++)**

**visitstring(s[i],h);**

**}**

* 1. **线索二叉树**

**●二叉树的指针域，有许多处于空状态，可以加**

**以利用，保存遍历时该结点的直接前驱或直接后继结点的地址，这些地址称为线索，带有线索的二叉树称为线索二叉树。**

**处理这样的树既便利又可以提高效率。**

A

C

B

E

D

G

F

**●给二叉树的结点添加线索的过程，称为二叉树的线索化。这样的二叉树称为线索二叉树。**

**线索化的方法是：若结点左（右）子针为空，将其改为直接指向该结点前驱（后继）的指针，称为左（右）线索。**

**如“先序”线索二叉树：A，B，D，F，C，E，G**

A

C

B

E

D

G

F

**如“中序”线索二叉树：B,F,D,A,E,G,C**

A

C

B

E

D

G

F

**如“后序”线索二叉树：F,D,B,G,E,C,A**

A

C

B

E

D

G

F

**●在一个线索二叉树中，为了区别每个结点的左右指针域所存放的是孩子指针还是线索，必须在结点的结构中增加两个线索标志域，左、右线索标志域（ltag和rtag），这两个域分别用0和1表示是孩子指针还是线索。结构为：**

Lchild

Lchild

Ltag

Data

Rtag

**Rchild**

**定义如下：**

**typedef struct BThrnode**

**{**

**int ltag,rtag;**

**DataType data;**

**struct BThrnode \*lchild,\*rchild;**

**}BThrTree;**

**BThrTree TTreeNode；**

**例“中序”线索二叉树的存储结构：**

**B，F，D，A，E，G，C**

0

0

A

1

C

0

0

B

1

0

E

1

0

D

1

1

F

1

1

G

1

**建立中序线索化的算法：**

**void InThread(TTreeNode \*HT)**

**{//对以树根指针为HT的二叉树加中序线索**

**static TTreeNode \*pre=NULL;**

**//指向前驱结点的指针**

**if(HT!=NULL)**

**{**

**if(HT🡪ltag==0)InThread(HT🡪lchild);**

**//当左子树非空给予加中序线索**

**if(pre!=NULL&&pre🡪rtag==1)**

**pre🡪rchild=HT;**

**//给前驱结点加后继线索**

**if(HT🡪lchild==NULL)**

**{//给当前结点加前驱线索**

**HT🡪ltag=1;HT🡪lchild=pre;**

**}**

**if(HT🡪rchild==NULL)HT🡪rtag=1;**

**//给右指针域为空的结点加右线索标志**

**pre=HT;//把刚访问过的结点置为前驱结点**

**if(HT🡪rtag==0)InThread(HT🡪rchild);**

**//当右子树非空给予加中序线索**

**}**

**}**

**●中序线索二叉树进行遍历算法就变得简单：**

**void ThInorder(TTreeNode \*HT)//HT为树根指针**

**{**

**if(HT!=NULL)**

**{**

**while(HT🡪ltag==0)HT=HT🡪lchild;**

**//查找出中序遍历中的第一个结点**

**do{**

**printf(“%c “,HT🡪data);**

**HT=InorderNext(HT);**

**//查找出HT结点的中序后继结点**

**}while(HT!=NULL);**

**}**

**}**