树 第 20 题

问题描述:

用二叉链表存储二叉树，判断二叉树是否为完全二叉树。

1：函数结构设计

函数名	IsBFT
函数正常输入	BitTree 型二叉树头结点指针
函数正常输出	是完全二叉树返回 TRUE=1，否则返回 FALSE=-1

2：测试样例设计

	输入	预测结果
一般正常情况	二叉树头结点指针	是完全二叉树返回 TRUE=1， 否则返回 FALSE=-1
异常情况	二叉树头指针为空	返回 FALSE=-1

3：伪代码描述

如果输入的二叉树头指针为空，返回 FALSE=-1
利用队列层序遍历二叉树
如果访问到非空节点，将其子节点加入队列，如果该节点没有子节点， 仍然将空节点 NULL 加入队列
如果第一次访问到空节点 NULL，且后面的节点也全部为空节点，则为完全二叉树，返回 TRUE=1，如果后面有非空节点，则不是完全二叉树，返回 FALSE=-1
如果恰好遍历完所有节点且没有访问非空节点，返回 TRUE=1

4：程序描述

```
// 树第 20 题

#include "stdafx.h"
#include <assert.h>
#include <iostream>
using namespace std;

#define TRUE              1
#define FALSE            -1

////////////////////////////////////////////////

// 线性表实现的队列，元素类型为 ElemType 型，用于树的创建时参数的输入

typedef char ElemType;

#define QUEUE_CHAR_SIZE    20

typedef struct QueueChar {
	ElemType dat[QUEUE_CHAR_SIZE];
	int front, rear, size;
} QueueChar;

int EnQueueC(QueueChar *q, char c);

// 从字符数组创建队列
QueueChar *CreateQueueC(ElemType *s)
{
	QueueChar *q = new QueueChar;
	assert(q != NULL);
	q->front = q->rear = q->size = 0;
	while(*s) {
		EnQueueC(q, *s++);
	}
	return q;
}
```

```
// 入队
int EnQueueC(QueueChar *q, ElemType c)
{
	if(q == NULL || q->size == QUEUE_CHAR_SIZE) {
		return FALSE;
	}
	q->dat[q->rear] = c;
	++q->size;
	q->rear = (q->rear + 1) % QUEUE_CHAR_SIZE;
	return TRUE;
}

// 出队
int DeQueueC(QueueChar *q, ElemType *c)
{
	if(q == NULL || q->size == 0) {
		return FALSE;
	}
	*c = q->dat[q->front];
	--q->size;
	q->front = (q->front + 1) % QUEUE_CHAR_SIZE;
	return TRUE;
}

// 判断是否为空
int QueueEmptyC(QueueChar *q)
{
	if(q == NULL || q->size == 0) {
		return TRUE;
	}
	return FALSE;
}


//////////////////////////////////////////////////

// 链表实现的队列，元素类型为 BiTree 型，用于判断是否为完全树时的层序遍历

typedef struct BiNode {
	ElemType dat;
	BiNode *lchild;
	BiNode *rchild;
} BiNode, *BiTree;
```

```
typedef struct LinkList
{
	BiTree dat;
	LinkList *next;
} LinkList;

typedef struct Queue
{
	LinkList *list;
	LinkList *rear;
} Queue;

// 创建空队列
Queue *CreateQueue(void)
{
	Queue *q = new Queue;
	assert(q != NULL);
	LinkList *head = new LinkList;
	assert(head != NULL);
	head->next = head;
	q->list = head;
	q->rear = head;
	return q;
}

// 入队
int EnQueue(Queue *q, BiTree dat)
{
	if(q == NULL) {
		return FALSE;
	}
	LinkList *New = new LinkList;
	New->dat = dat;
	New->next = q->list;
	q->rear->next = New;
	q->rear = New;
	return TRUE;
}

// 出队
int DeQueue(Queue *q, BiTree *dat)
{
	if((q == NULL) || (q->list == NULL) || (q->list == q->rear)) {
		return FALSE;
```

```
    }
    LinkList *lst = q->list->next;
    *dat = lst->dat;
    q->list->next = lst->next;
    if(q->rear == lst) {
        q->rear = q->list;
    }
    delete lst;
    return TRUE;
}

// 判断队列是否为空
int QueueEmpty(Queue *q)
{
    if((q == NULL) || (q->list == NULL) || (q->list == q->rear)) {
        return TRUE;
    }
    return FALSE;
}

////////////////////////////////////////////////////////////

// 前序遍历方式创建二叉树 递归函数
BiTree &CreateBiTree(BiTree &t, QueueChar *q)
{
    ElemType e;
    if(DeQueueC(q, &e) == FALSE || e == '#') {
        t = NULL;
    } else {
        t = new BiNode;
        assert(t != NULL);
        t->dat = e;
        t->lchild = CreateBiTree(t->lchild, q);
        t->rchild = CreateBiTree(t->rchild, q);
    }
    return t;
}

// 判断二叉树是否为完全二叉树 Full Binary Tree（利用层序遍历）
int IsFBT(BiTree t)
{
    if(t == NULL) {
        return FALSE;
    }
```

```
    Queue *q = CreateQueue();
    EnQueue(q, t);
    while(QueueEmpty(q) == FALSE) {
        DeQueue(q, &t);
        if(t != NULL) {
            EnQueue(q, t->lchild);          // 如果子节点为空，空节点 NULL 入队
            EnQueue(q, t->rchild);
        } else {          // 已经访问到第一个空节点，则对于完全二叉树，后面还未访问的节点都是空节点
            while(DeQueue(q, &t) == TRUE) {
                if(t != NULL) {
                    return FALSE;
                }
            }
            return TRUE;
        }
    }
    return TRUE;
}

int main(void)
{
    /*
    // 五个用于测试的二叉树

    // ----------------1----------------
            A
        B       C
      D   E   F   G
     # # # # # # # #

    // ----------------2----------------
            A
        B       C
      D   E   #   #
     # # # #

    // ----------------3----------------
            A
        B       C
      D   #   E   #
     # #     # #
```

```
    // ----------------4----------------
              A
        B           #
      D     E
     # # # #

    // ----------------5----------------
              A
        B           C
      #     D     #     #
           # #

    // ---------------- ----------------
    */

    ElemType c1[] = "ABD##E##CF##G##";
    ElemType c2[] = "ABD##E##C##";
    ElemType c3[] = "ABD###CE###";
    ElemType c4[] = "ABD##E###";
    ElemType c5[] = "AB#D##C##";
    ElemType *p[] = {c1, c2, c3, c4, c5};

    cout << IsFBT(NULL) << endl;  // 异常情况

    for(int i = 0; i < 5; ++i) {   // 正常情况，判断以上五种二叉树是否为完全二叉树
        BiTree t = CreateBiTree(t, CreateQueueC(p[i]));
        cout << IsFBT(t) << endl;
    }

    system("pause");
    return 0;
}
```

5：结果展示

```
-1                  // 异常情况：返回 FALSE=-1
1                   // 正常情况：1,2 为完全二叉树，返回 TRUE=1
1
-1                  // 正常情况：3,4,5 不是完全二叉树，返回 FALSE=-1
-1
-1
```