Lab4 Sorting

2014050693 贺嘉允

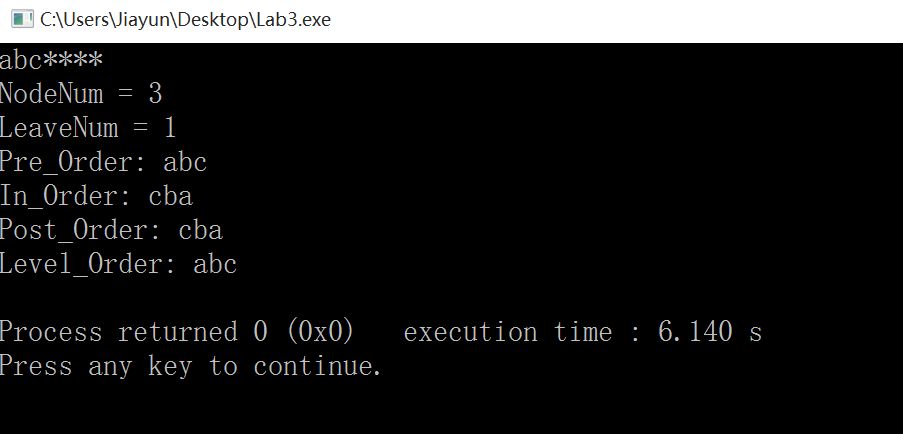
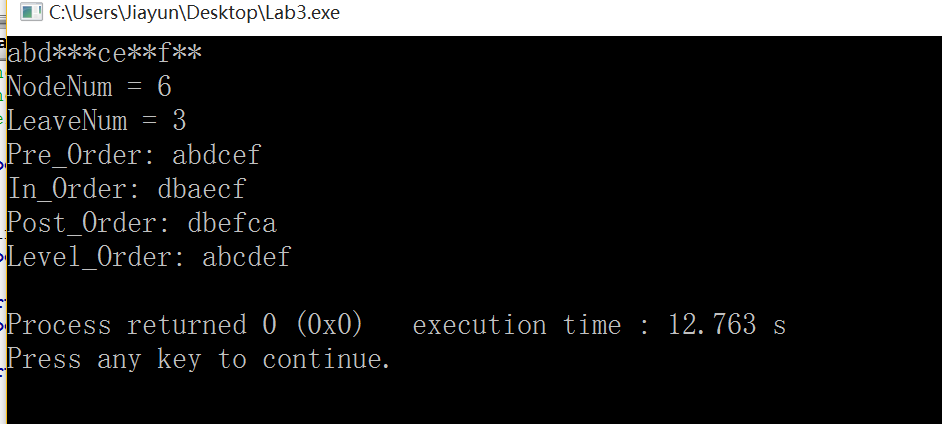
*Section 1: Introduction*

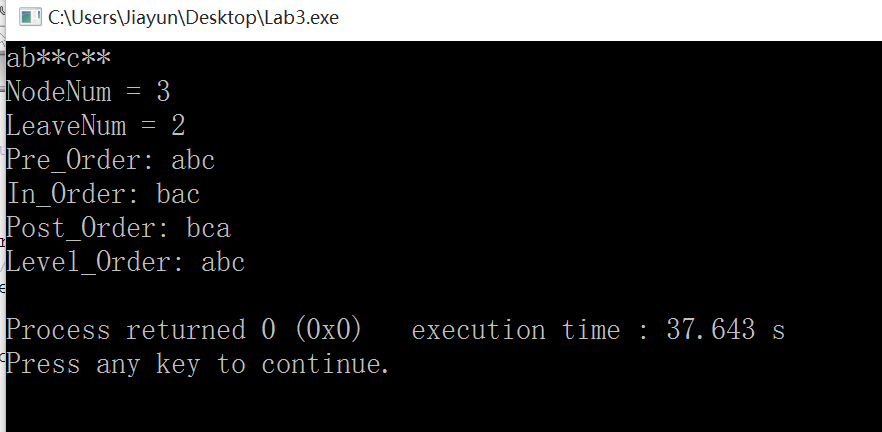
Assigned：Oct. 30th

Using linked representation as the storage structure of a binary tree, input the pre-order traversal sequence of a binary tree, where \* represents virtual node (i.e. the node does not exist, NULL link), for example ABD\*\*\*CE\*\*F\*\*. Create the binary tree and complete the following operations:

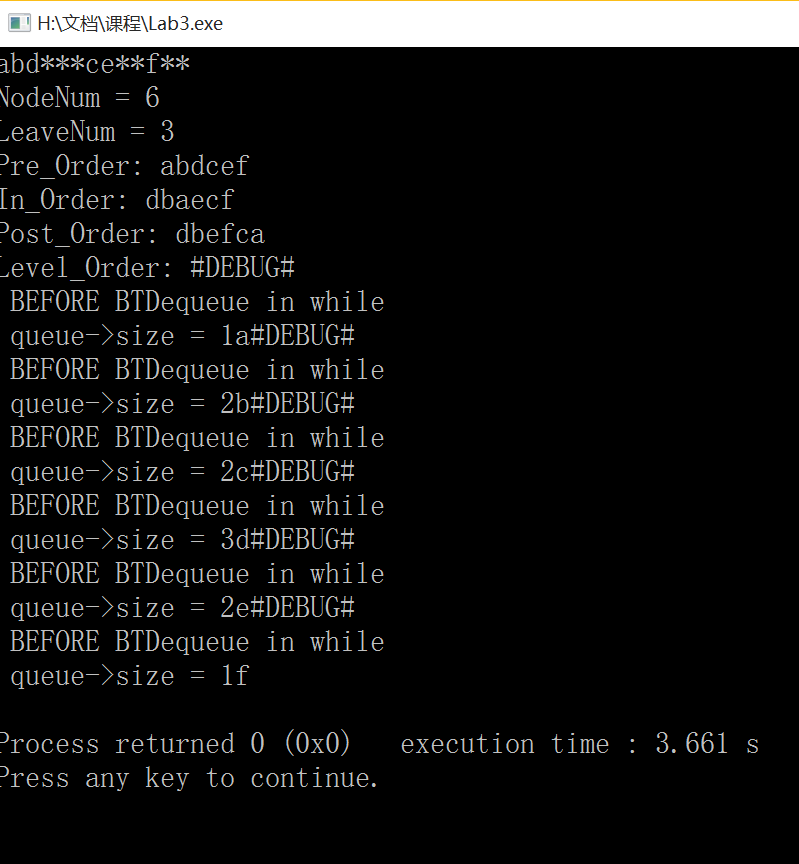
1. output the traversal sequences of first-order, in-order, post-order and breadth-first traversal of the tree

2. count the number of the leaves of the tree and the total number of the nodes in the tree *Section 2: Algorithms Specification*  
In the program, the data structure Queue was used to realize the level-order(Breadth-order) traversal of a binary tree. In the other traversals and in the counting of total nodes and leaves, the recursive method was used.  
*Section 3: Test results*





*if I set the DEBUG parameter to 1 to see the process in the queue inside the breadth-order traversal, the result with input “ abd\*\*\*ce\*\*f\*\*” is shown below:*

  
*Section 4: Analysis and Comments*  
The time complexity of the four traversal cases is O(n).  
  
*Section 5: Source code ( in C language)*  
#include<stdio.h>

#include<stdlib.h>

#define DEBUG 1 //for debugging in the level-traversal

typedef struct node{

char data;

struct node \*lchild,\*rchild;

}BinBinTree; //define the data type of the node

typedef BinBinTree \*BinTree;

struct QueueRecord;

typedef struct QueueRecord \*Queue;

struct QueueRecord

{

int capacity;

int front;

int rear;

int size;

BinTree \*Array;

};

//Functions Used//

//QUEUE//

Queue CreateQueue(int capacity);

int IsEmpty(Queue queue);

static int Succ( int value, Queue queue);

int IsFull(Queue queue);

void BTDequeue(Queue queue);

void BTEnqueue(Queue queue, BinTree T);

//Binary Tree Traversal//

BinTree CreatBinTree(void);

void Pre\_Order( BinTree T );

void In\_Order( BinTree T );

void Post\_Order( BinTree T );

void Level\_Order( BinTree T );

//Binary Tree Count//

void Node\_Num(BinTree T, int \*num);

void Leave\_Num(BinTree T, int \*num);

Queue CreateQueue(int capacity)

{

Queue queue;

queue = (Queue)malloc(sizeof(Queue));

if (queue == NULL)

{

printf("out of space!!!\n");

}

queue->Array = (BinTree \*)malloc(capacity \* sizeof(BinTree));

if (queue->Array == NULL)

{

printf("out of space!!!\n");

}

queue->size = 0;

queue->rear = 0;

queue->front = 1;

return queue;

}

int IsEmpty(Queue queue)

{

return queue->size == 0;

}

static int Succ( int value, Queue queue)

{

if(++value == queue->capacity)

value = 0;

return value;

}

int IsFull(Queue queue)

{

return (queue->size >= queue->capacity);

}

void BTDequeue(Queue queue)

{

if(IsEmpty(queue))

{

printf("Error!");

exit(1);

}

else

{

printf("%c", queue->Array[queue->front]->data);

queue->front++;

queue->size--;

// printf("size after Dequeue = %d\n ", queue->size);

}

}

void BTEnqueue(Queue queue, BinTree T)

{

if(IsFull(queue))

{

printf("Error!");

exit(1);

}

queue->size++;

queue->rear++;

// printf("The Enqueue element is %c ", T->data);

queue->Array[queue->rear] = T;

// printf("size after Enqueue = %d\n ", queue->size);

}

/\*=Create a binary tree recursively based on pre-order traversal sequence=\*/

BinTree CreatBinTree(void)

{

BinTree T;

char ch;

if((ch=getchar())=='\*')

return(NULL); /\*input is ¡®\*¡¯£¬return null vector\*/

else{

T= (BinBinTree \*)malloc(sizeof(BinBinTree)); /\*create a node\*/

T->data=ch;

T->lchild=CreatBinTree(); /\*create the left subtree\*/

T->rchild=CreatBinTree(); /\*create the right subtree\*/

return(T);

}

}

void Pre\_Order( BinTree T )

{

if(T)

{

printf("%c", T->data);

Pre\_Order(T->lchild );

Pre\_Order(T->rchild );

}

}

void In\_Order( BinTree T )

{

if(T)

{

In\_Order( T->lchild );

printf("%c", T->data);

In\_Order( T->rchild );

}

}

void Post\_Order( BinTree T )

{

if(T)

{

Post\_Order( T->lchild );

Post\_Order( T->rchild );

printf("%c", T->data);

}

}

void Level\_Order( BinTree T )

{

Queue queue = CreateQueue(20);

BinTree Temp;

BTEnqueue(queue, T);

while(queue->size!= 0)

{

Temp = queue->Array[queue->front];

if(DEBUG)

{

printf("#DEBUG#\n BEFORE BTDequeue in while \n ");

printf("queue->size = %d ", queue->size);

}

BTDequeue(queue);

if(Temp->lchild)

BTEnqueue(queue,Temp->lchild);

if(Temp->rchild)

BTEnqueue(queue,Temp->rchild);

}

}

void Node\_Num(BinTree T, int \*num)

{

if(T)

{

(\*num)++;

Node\_Num(T->lchild, num);

Node\_Num(T->rchild, num);

} else

return;

}

void Leave\_Num(BinTree T, int \*num)

{

if(!T)

return;

if((T->lchild==NULL)&&(T->rchild==NULL))

(\*num)++;

else

{

Leave\_Num(T->lchild, num);

Leave\_Num(T->rchild, num);

}

}

int main()

{

int num = 0;

BinTree T = CreatBinTree();

Node\_Num(T, &num);

printf("NodeNum = %d\n",num);

num = 0; //reset num for counting leaves

Leave\_Num(T, &num);

printf("LeaveNum = %d\n",num);

printf("Pre\_Order: ");

Pre\_Order(T);

printf("\n");

printf("In\_Order: ");

In\_Order(T);

printf("\n");

printf("Post\_Order: ");

Post\_Order(T);

printf("\n");

printf("Level\_Order: ");

Level\_Order(T);

printf("\n");

return 0;

}