

**数据结构课程实验报告**

**（4）**

**姓名：顾宬书**

**学号：2135061913**

**专业：通信工程**

**完成日期：2023.4.17**

**目录**

[目录 - 1 -](#_Toc434170429)

[1 设计要求 - 2 -](#_Toc434170430)

[2 程序功能框图 - 2 -](#_Toc434170431)

[3 数据结构说明 - 2 -](#_Toc434170432)

[4 重要算法核心代码 - 2 -](#_Toc434170433)

[5 测试运行界面 - 2 -](#_Toc434170434)

[6 完整源程序 - 3 -](#_Toc434170435)

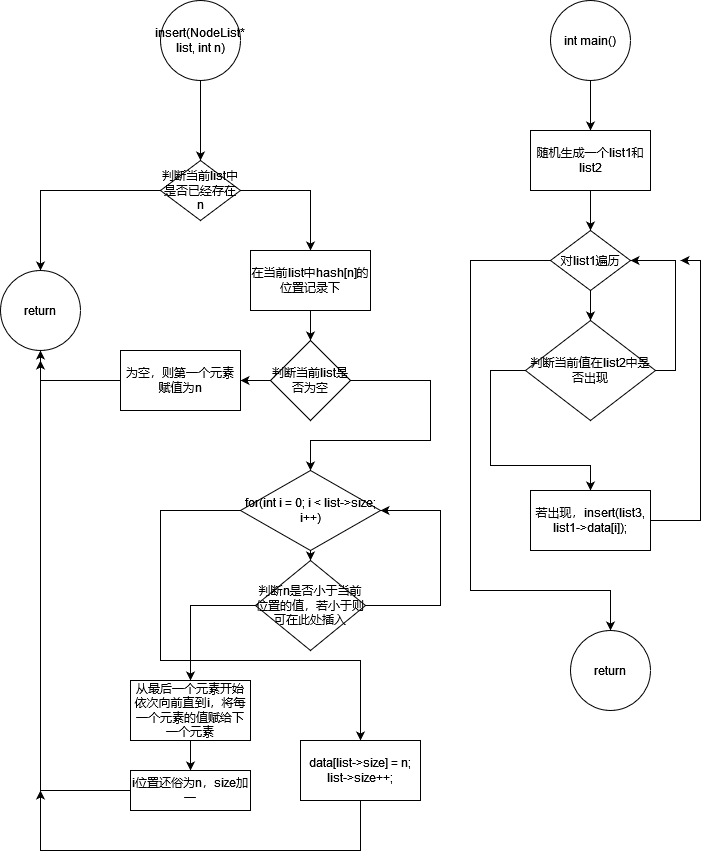
**数据结构实验四（综合练习）**

**1、链表**

【问题描述】

给定两个已升序排序的链表L1和L2，只使用基本的链表操作编写计算L1∩L2的过程。结果链表也要求是升序排序的。

【程序功能框图】



【数据结构说明】

NodeList结构中含有size，hash[]用于记录某一个值是否已经在其中存在，data[]存放数据。

【重要算法核心代码】

void insert(NodeList\* list, int n) {

if (list->hash[n] != 0) {

return;

}

list->hash[n] = n;

if (list->size == 0) {

list->size++;

list->data[0] = n;

return;

}

else {

for (int i = 0; i < list->size; i++) {

if (n < list->data[i]) {

for (int j = list->size; j > i; j--) {

list->data[j] = list->data[j - 1];

}

list->data[i] = n;

list->size++;

return;

}

}

list->data[list->size] = n;

list->size++;

return;

}

}

【源代码】

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

typedef struct \_nodeList {

int size;

int hash[0xFFFF];

int data[0xFFFF];

}NodeList;

void init(NodeList\* list) {

list->size = 0;

for (int i = 0; i < 0xFFFF; i++)

list->hash[i] = 0;

}

void insert(NodeList\* list, int n) {

if (list->hash[n] != 0) {

return;

}

list->hash[n] = n;

if (list->size == 0) {

list->size++;

list->data[0] = n;

return;

}

else {

for (int i = 0; i < list->size; i++) {

if (n < list->data[i]) {

for (int j = list->size; j > i; j--) {

list->data[j] = list->data[j - 1];

}

list->data[i] = n;

list->size++;

return;

}

}

list->data[list->size] = n;

list->size++;

return;

}

}

int main() {

NodeList\* list1 = (NodeList\*)malloc(sizeof(NodeList));

NodeList\* list2 = (NodeList\*)malloc(sizeof(NodeList));

NodeList\* list3 = (NodeList\*)malloc(sizeof(NodeList));

init(list1); init(list2); init(list3);

srand(time(NULL));

int n;

for (int i = 0; i < 0xFF; i++) {

n = rand() % 1000 + 1;

insert(list1, n);

}

printf("list1: ");

for (int i = 0; i < list1->size; i++) {

printf("%d ", list1->data[i]);

}

printf("\n");

for (int i = 0; i < 0xFF; i++) {

n = rand() % 1000 + 1;

insert(list2, n);

}

printf("list2: ");

for (int i = 0; i < list2->size; i++) {

printf("%d ", list2->data[i]);

}

printf("\n");

for (int i = 0; i < list1->size; i++) {

if (list2->hash[list1->data[i]] != 0)

insert(list3, list1->data[i]);

}

printf("list3: ");

for (int i = 0; i < list3->size; i++) {

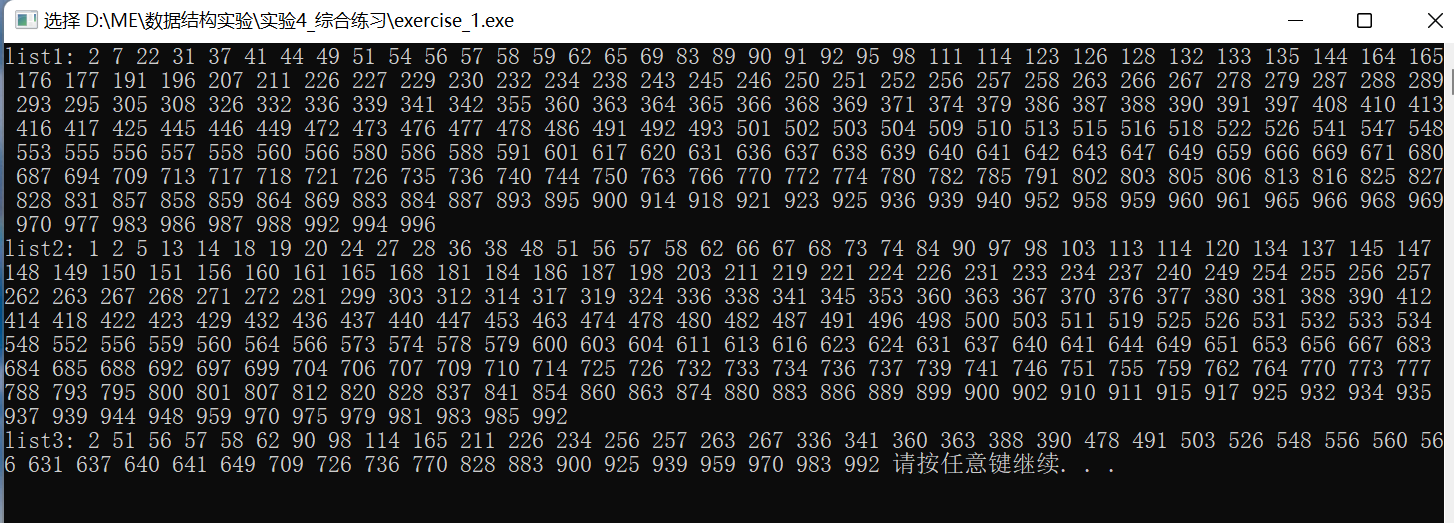
printf("%d ", list3->data[i]);

}

system("pause");

}

【测试运行界面】



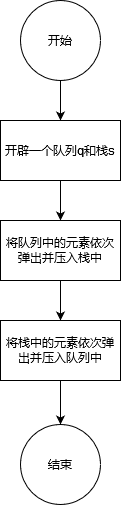
**2、链表/堆栈**

【问题描述】

用两种非递归方法以O(N)的时间复杂度，实现反转一个单链表。

* 做法1：用堆栈来保存链表的遍历，用pop倒序输出，每倒序输出一个值就构造新链表的一个结点。
* 做法2：在原链表上对结点指针重新赋值操作来进行链表的反转。

【程序功能框图】



【数据结构说明】

Stack结构中，data用于存放数据，size记录大小，top记录了当前栈顶的位置。

Queue结构中采用循环队列，data用于存放数据，head记录了当前队首的元素后一个位置，tail记录了队末的元素位置，size表示大小，isEmpty表示当前队列是否为空。

【重要算法核心代码】

void insertQueue(Queue\* q, int n) {

if (!q->isEmpty && q->head == q->tail)

return;

else {

if (q->isEmpty == 1)

q->isEmpty = 0;

q->data[q->head] = n;

q->head = (q->head + 1) % 0xFF;

q->size++;

}

}

int popQueue(Queue\* q) {

if (q->isEmpty)

return;

else {

int pop;

pop = q->data[q->tail];

q->tail = (q->tail + 1) % 0xFF;

q->size--;

if (q->size == 0)

q->isEmpty = 1;

return pop;

}

}

【源代码】

#include<stdio.h>

#include<stdlib.h>

typedef struct \_stack {

int data[0xFF];

int size;

int top;

}Stack;

typedef struct \_queue {

int data[0xFF];

int head;

int tail;

int size;

int isEmpty;

}Queue;

void initStack(Stack\* stack) {

stack->top = -1;

}

void push(Stack\* s, int n) {

s->top++;

s->data[s->top] = n;

s->size++;

}

int pop(Stack\* s) {

int pop = s->data[s->top];

s->top--;

return pop;

}

void initQueue(Queue\* q) {

q->head = 0;

q->tail = 0;

q->size = 0;

q->isEmpty = 1;

}

void insertQueue(Queue\* q, int n) {

if (!q->isEmpty && q->head == q->tail)

return;

else {

if (q->isEmpty == 1)

q->isEmpty = 0;

q->data[q->head] = n;

q->head = (q->head + 1) % 0xFF;

q->size++;

}

}

int popQueue(Queue\* q) {

if (q->isEmpty)

return;

else {

int pop;

pop = q->data[q->tail];

q->tail = (q->tail + 1) % 0xFF;

q->size--;

if (q->size == 0)

q->isEmpty = 1;

return pop;

}

}

int main() {

Queue\* q = (Queue\*)malloc(sizeof(Queue));

initQueue(q);

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

initStack(s);

for (int i = 0; i < 99; i++) {

insertQueue(q, i);

}

while (!q->isEmpty) {

push(s, popQueue(q));

}

while (s->top != -1) {

insertQueue(q, pop(s));

}

for (int i = q->tail; i != q->head; i = (i + 1) % 0xFF) {

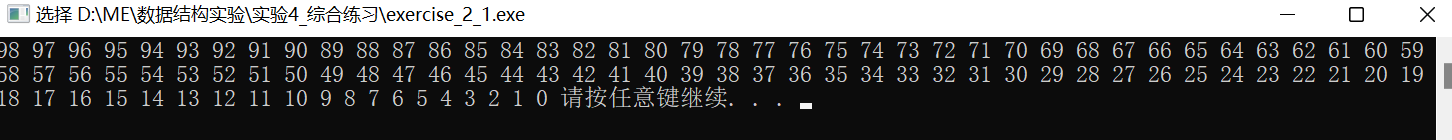
printf("%d ", q->data[i]);

}

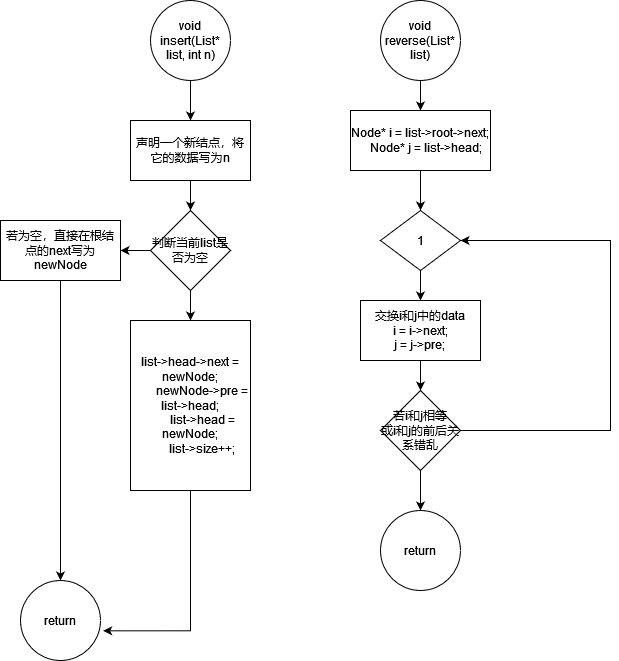
system("pause");

}

【测试运行界面】



【程序功能框图】



【数据结构说明】

Node结点存放了int类型的data，下一个指针，以及上一个指针。

List中存放了root指针，指向了根结点，根节点不存放任何数据。Head指针指向的是最后一个被加入的node

【重要算法核心代码】

void reverse(List\* list) {

Node\* i = list->root->next;

Node\* j = list->head;

int temp;

while (1) {

temp = i->data;

i->data = j->data;

j->data = temp;

i = i->next;

j = j->pre;

if (i == j)

return;

if (i->pre == j || j->next == i)

return;

}

}

【源代码】

#include<stdio.h>

#include<stdlib.h>

typedef struct \_node {

int data;

struct \_node\* next;

struct \_node\* pre;

}Node;

typedef struct \_list {

Node\* head;

Node\* root;

int isEmpty;

int size;

}List;

void initList(List\* list) {

list->size = 0;

list->isEmpty = 1;

list->head = NULL;

list->root = (Node\*)malloc(sizeof(Node));

list->root->data = NULL;

list->root->next = NULL;

list->root->pre = NULL;

}

void insert(List\* list, int n) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->next = NULL;

newNode->pre = NULL;

newNode->data = n;

if (list->isEmpty) {

list->root->next = newNode;

newNode->pre = list->root;

list->head = newNode;

list->size++;

list->isEmpty = 0;

}

else {

list->head->next = newNode;

newNode->pre = list->head;

list->head = newNode;

list->size++;

}

}

void reverse(List\* list) {

Node\* i = list->root->next;

Node\* j = list->head;

int temp;

while (1) {

temp = i->data;

i->data = j->data;

j->data = temp;

i = i->next;

j = j->pre;

if (i == j)

return;

if (i->pre == j || j->next == i)

return;

}

}

int main() {

List\* list = (List\*)malloc(sizeof(List));

initList(list);

for (int i = 0; i < 10; i++)

insert(list, i);

reverse(list);

for (Node\* ptr = list->root->next; ptr != NULL; ptr = ptr->next)

printf("%d ", ptr->data);

system("pause");

}

【测试运行界面】



**3、队列**

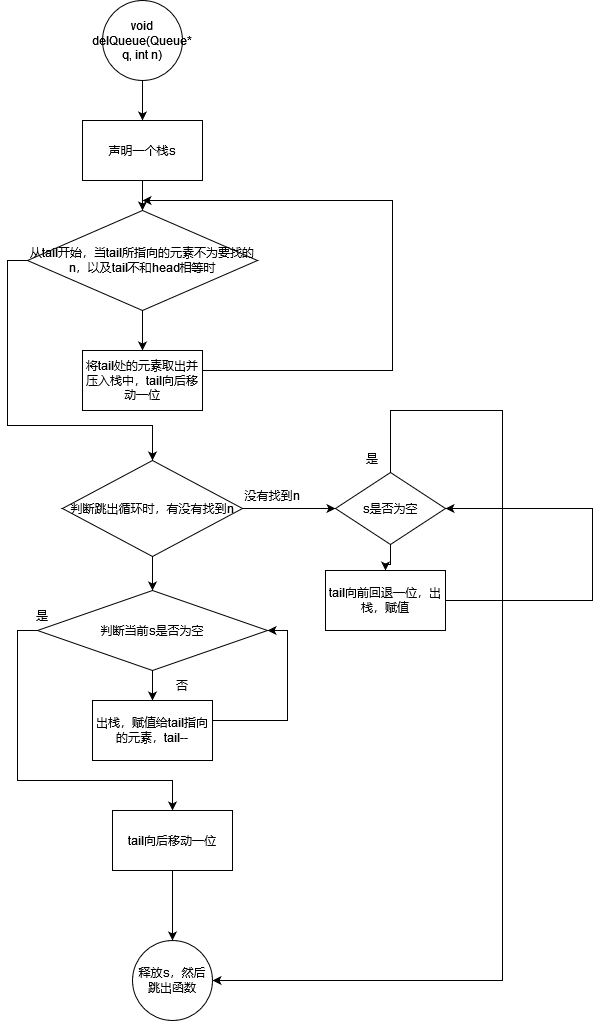
【问题描述】

假设以数组Q[m]存放循环队列中的元素，同时以rear和length 分别指示环形队列中的队尾位置和队列中所含元素的个数：

1）求队列中第一个元素的实际位置。

2）给出该循环队列的队空条件和队满条件，并写出相应的插入(enqueue)和删除(delqueue)元素的操作。

【程序功能框图】



【数据结构说明】

Queue结构中data数组存放int类型数据，tail指向了队尾的值，head指向了最后一个加入队列中的元素的后一个地址位置，head指向的位置值没有意义。Size是大小。

【重要算法核心代码】

void enQueue(Queue\* q, int n) {

if (q->size < SIZE) {

q->data[q->head] = n;

q->head = (q->head + 1) % SIZE;

q->size++;

}

}

void delQueue(Queue\* q, int n) {

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

initStack(s);

while (q->data[q->tail] != n && q->tail != (q->head)) {

push(s, q->data[q->tail]);

q->tail = (q->tail + 1) % SIZE;

}

if (q->data[q->tail] == n) {

q->size--;

while (s->size > 0) {

q->data[q->tail] = pop(s);

q->tail = (q->tail - 1) % SIZE;

}

q->tail = (q->tail + 1) % SIZE;

}

else {

while (s->size > 0) {

q->tail = (q->tail - 1) % SIZE;

q->data[q->tail] = pop(s);

}

}

//q->size--;

free(s);

}

【源代码】

#include<stdio.h>

#include<stdlib.h>

#define SIZE 0xFF

typedef struct \_queue {

int tail;

int head;

int size;

int data[SIZE];

}Queue;

typedef struct \_stack {

int data[0xFF];

int size;

int top;

}Stack;

void initStack(Stack\* stack) {

stack->top = -1;

stack->size = 0;

}

void push(Stack\* s, int n) {

s->top++;

s->data[s->top] = n;

s->size++;

}

int pop(Stack\* s) {

int pop = s->data[s->top];

s->top--;

s->size--;

return pop;

}

void initQueue(Queue\* q) {

q->tail = 0;

q->head = 0;

q->size = 0;

}

int isEmpty(Queue\* q) {

return (q->size == 0);

}

void enQueue(Queue\* q, int n) {

if (q->size < SIZE) {

q->data[q->head] = n;

q->head = (q->head + 1) % SIZE;

q->size++;

}

}

void delQueue(Queue\* q, int n) {

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

initStack(s);

while (q->data[q->tail] != n && q->tail != (q->head)) {

push(s, q->data[q->tail]);

q->tail = (q->tail + 1) % SIZE;

}

if (q->data[q->tail] == n) {

q->size--;

while (s->size > 0) {

q->data[q->tail] = pop(s);

q->tail = (q->tail - 1) % SIZE;

}

q->tail = (q->tail + 1) % SIZE;

}

else {

while (s->size > 0) {

q->tail = (q->tail - 1) % SIZE;

q->data[q->tail] = pop(s);

}

}

//q->size--;

free(s);

}

int main() {

Queue\* q = (Queue\*)malloc(sizeof(Queue));

initQueue(q);

for (int i = 0; i <= 128; i++)

enQueue(q, i);

delQueue(q, 9);

delQueue(q, 127);

delQueue(q, 128);

delQueue(q, 256);

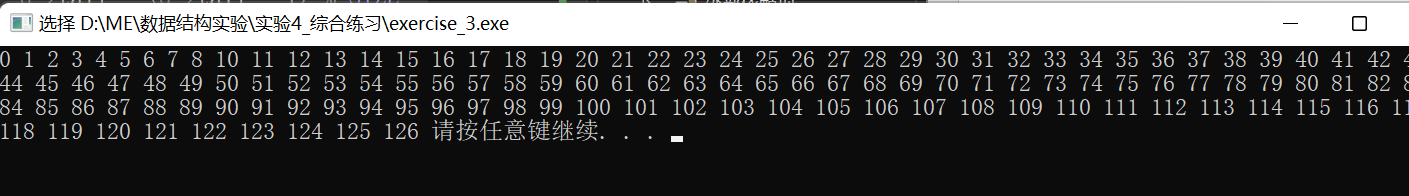
for (int tail = q->tail; tail != q->head; tail = (tail + 1) % SIZE)

printf("%d ", q->data[tail]);

system("pause");

}

【测试运行界面】



**4、二叉树**

【问题描述】

判断一个二叉树是不是二叉搜索树。

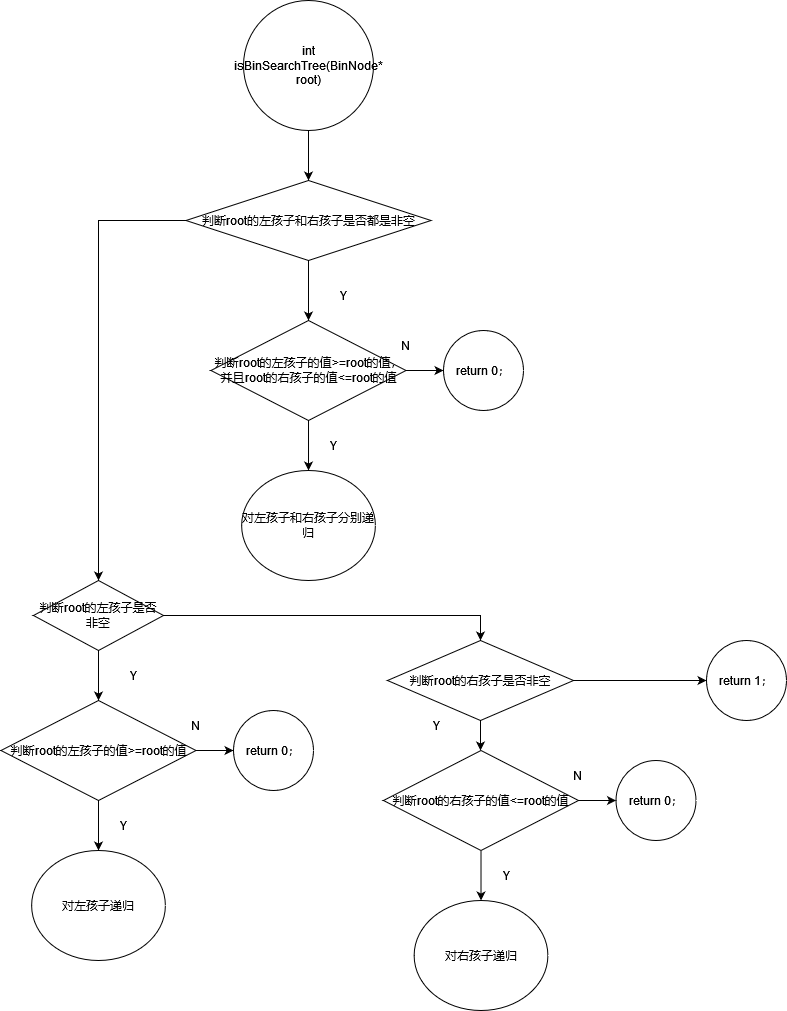
二叉搜索树必须满足：

1)或者是空树

2)如果不是空树，则对于任意一个节点

* 左子树（如果有）中所有节点都小于该节点
* 右子树（如果有）中所有节点都大于该节点
* 左子树（如果有）也是二叉搜索树
* 右子树（如果有）也是二叉搜索树

【程序功能框图】



【数据结构说明】

Node有data存放int类型数据，lchild是指向左子节点的指针，rchilld是指向右子节点的指针。

【重要算法核心代码】

int isBinSearchTree(BinNode\* root) {

if (root->lchild != NULL && root->rchild != NULL) {

if (root->data <= root->rchild->data && root->data >= root->lchild->data) {

return isBinSearchTree(root->lchild) \* isBinSearchTree(root->rchild);

}

else

return 0;

}

else if (root->rchild != NULL) {

if (root->data <= root->rchild->data) {

return isBinSearchTree(root->rchild);

}

else

return 0;

}

else if (root->lchild != NULL) {

if (root->data >= root->lchild->data) {

return isBinSearchTree(root->lchild);

}

else

return 0;

}

else if (root->lchild == NULL && root->rchild == NULL)

return 1;

}

【源代码】

#include<stdio.h>

#include<stdlib.h>

typedef struct \_BinNode {

int data;

struct \_BinNode\* lchild;

struct \_BinNode\* rchild;

}BinNode;

BinNode\* init() {

return (BinNode\*)malloc(sizeof(BinNode));

}

int isBinSearchTree(BinNode\* root) {

if (root->lchild != NULL && root->rchild != NULL) {

if (root->data <= root->rchild->data && root->data >= root->lchild->data) {

return isBinSearchTree(root->lchild) \* isBinSearchTree(root->rchild);

}

else

return 0;

}

else if (root->rchild != NULL) {

if (root->data <= root->rchild->data) {

return isBinSearchTree(root->rchild);

}

else

return 0;

}

else if (root->lchild != NULL) {

if (root->data >= root->lchild->data) {

return isBinSearchTree(root->lchild);

}

else

return 0;

}

else if (root->lchild == NULL && root->rchild == NULL)

return 1;

}

int main() {

BinNode\* root = init();

root->data = 8;

root->lchild = init(); root->lchild->lchild = NULL; root->lchild->rchild = NULL;

root->lchild->data = 6;

root->rchild = init(); root->rchild->lchild = NULL; root->rchild->rchild = NULL;

root->rchild->data = 9;

BinNode\* newRoot = init(); newRoot->rchild = NULL;

newRoot->data = 1;

newRoot->lchild = root;

printf("%d\n", isBinSearchTree(newRoot));

system("pause");

}

【测试运行界面】

