**西南石油大学实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **项目编号** | **001063000502** | **项目名称** | 特殊线性表的应用 | **成绩** |  |
| **专业年级** | **计算机科学与技术2022级** | | | **指导教师** | **胡卫东** |
| **姓名** | **王磊** | **学号** | **202231060435** | **实验日期** | **2023.5.25** |

1. **实验目的**
2. 掌握链表存储结构的定义及C/C++语言实现
3. 掌握双链表的特点和基本运算及C/C++语言实现
4. 建立并正反序显示双链表结点数据，并执行结点插入操作
5. **实验工具**

PC微机，Windows，DOS，Turbo C或Visual C++

1. **实验步骤**

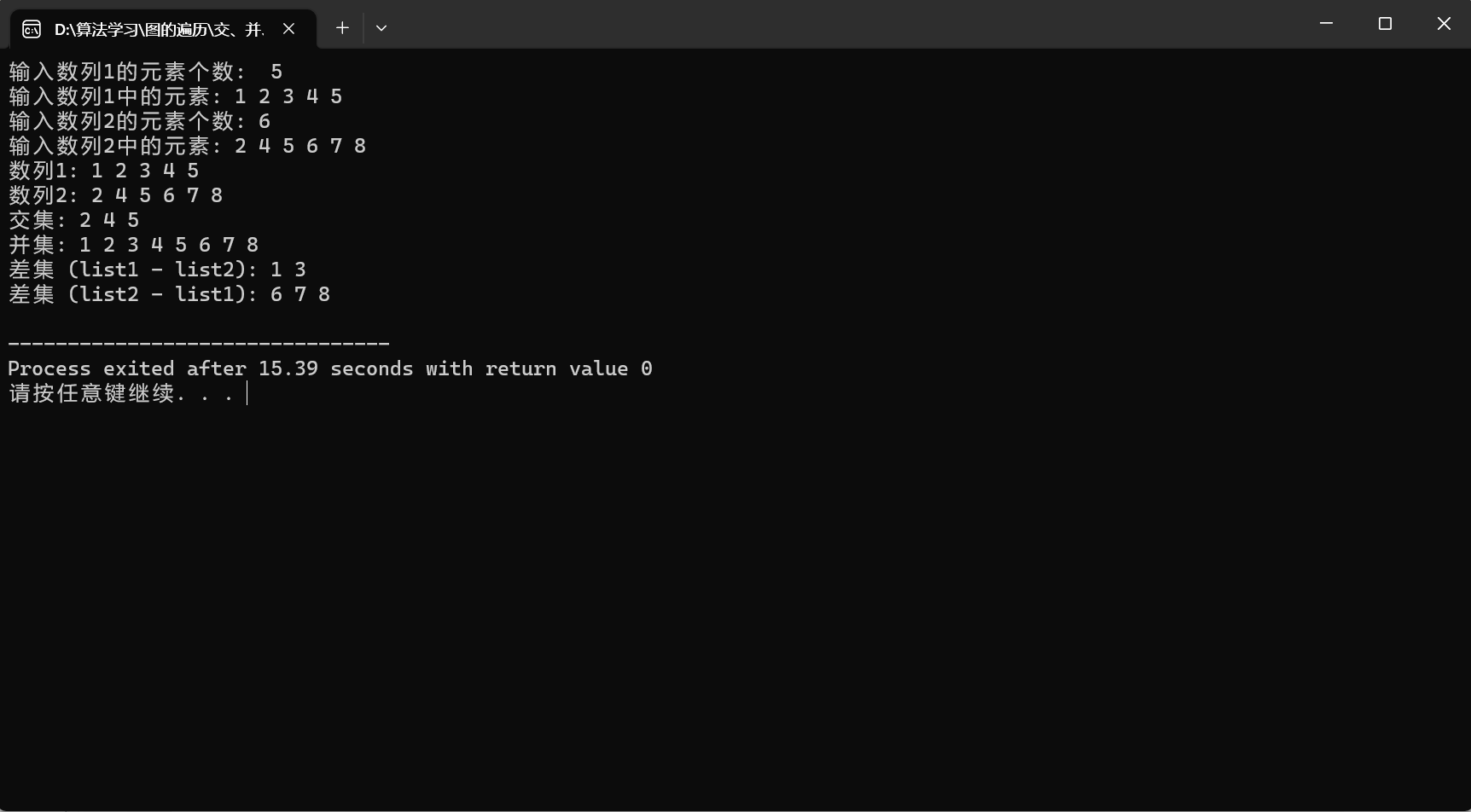
**定义链表节点结构体Node，包含一个整数数据data和指向下一个节点的指针next。 定义链表结构体LinkedList，包含一个指向链表头节点的指针head。**

**实现函数intersection，用于求两个集合的交集。函数接受两个链表指针list1和list2作为参数。创建一个新的链表result作为结果集合。遍历list1中的每个节点，对于每个节点，遍历list2中的每个节点，如果找到相同的值，将该值添加到result中。**

**实现函数unionSet，用于求两个集合的并集。函数接受两个链表指针list1和list2作为参数。创建一个新的链表result作为结果集合。遍历list1中的每个节点，将节点的值添加到result中。然后遍历list2中的每个节点，对于每个节点，检查是否已经在result中存在，如果不存在则将节点的值添加到result中。**

**实现函数difference，用于求两个集合的差集。函数接受两个链表指针list1和list2作为参数。创建一个新的链表result作为结果集合。遍历list1中的每个节点，对于每个节点，遍历list2中的每个节点，如果找不到相同的值，则将该值添加到result中。**

1. **实验结果**



1. **实验总结**

这个算法使用链表实现了求集合的并、交和差集的功能。主要步骤包括链表的初始化、节点的添加、链表元素的显示以及交、并、差集的计算。算法的总体思路是通过遍历链表，比较节点的值来求解集合的交、并、差集。

但是该算法的时间复杂度较高，特别是在求交、并、差集的过程中，存在两层循环嵌套。对于较大的集合，算法的效率可能较低。

**2、求集合的并、交和差集**

1. **问题描述**

求出任意两个正整数集合的的交、并和差集

1. **基本要求**

程序运行后显现提示信息，由用户输入两组整数分别作为两个集合的元素。程序将自动滤去由程序计算它们的交、并和差集，并将运算结果输出。

1. **算法描述**

**定义链表节点结构体Node，包含一个整数数据data和指向下一个节点的指针next。 定义链表结构体LinkedList，包含一个指向链表头节点的指针head。**

**实现函数intersection，用于求两个集合的交集。函数接受两个链表指针list1和list2作为参数。创建一个新的链表result作为结果集合。遍历list1中的每个节点，对于每个节点，遍历list2中的每个节点，如果找到相同的值，将该值添加到result中。**

**实现函数unionSet，用于求两个集合的并集。函数接受两个链表指针list1和list2作为参数。创建一个新的链表result作为结果集合。遍历list1中的每个节点，将节点的值添加到result中。然后遍历list2中的每个节点，对于每个节点，检查是否已经在result中存在，如果不存在则将节点的值添加到result中。**

**实现函数difference，用于求两个集合的差集。函数接受两个链表指针list1和list2作为参数。创建一个新的链表result作为结果集合。遍历list1中的每个节点，对于每个节点，遍历list2中的每个节点，如果找不到相同的值，则将该值添加到result中。**

**（4）算法实现——示例程序**

#include <stdio.h>

#include <stdlib.h>

// 定义链表节点结构体

typedef struct Node {

int data;

struct Node \*next;

} Node;

// 定义链表结构体

typedef struct LinkedList {

Node \*head;

} LinkedList;

// 初始化链表

void initLinkedList(LinkedList \*list) {

list->head = NULL;

}

// 在链表尾部添加节点

void addNode(LinkedList \*list, int data) {

Node \*new\_node = (Node \*)malloc(sizeof(Node));

new\_node->data = data;

new\_node->next = NULL;

if (list->head == NULL) {

list->head = new\_node;

} else {

Node \*current = list->head;

while (current->next != NULL) {

current = current->next;

}

current->next = new\_node;

}

}

// 输出链表中的元素

void displayList(LinkedList \*list) {

Node \*current = list->head;

while (current != NULL) {

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

current = current->next;

}

printf("\n");

}

// 求集合的交集

LinkedList \*intersection(LinkedList \*list1, LinkedList \*list2) {

LinkedList \*result = (LinkedList \*)malloc(sizeof(LinkedList));

initLinkedList(result);

Node \*current1 = list1->head;

while (current1 != NULL) {

Node \*current2 = list2->head;

while (current2 != NULL) {

if (current1->data == current2->data) {

addNode(result, current1->data);

break;

}

current2 = current2->next;

}

current1 = current1->next;

}

return result;

}

// 求集合的并集

LinkedList \*unionSet(LinkedList \*list1, LinkedList \*list2) {

LinkedList \*result = (LinkedList \*)malloc(sizeof(LinkedList));

initLinkedList(result);

Node \*current1 = list1->head;

while (current1 != NULL) {

addNode(result, current1->data);

current1 = current1->next;

}

Node \*current2 = list2->head;

while (current2 != NULL) {

Node \*current\_result = result->head;

int found = 0;

while (current\_result != NULL) {

if (current2->data == current\_result->data) {

found = 1;

break;

}

current\_result = current\_result->next;

}

if (!found) {

addNode(result, current2->data);

}

current2 = current2->next;

}

return result;

}

// 求集合的差集

LinkedList \*difference(LinkedList \*list1, LinkedList \*list2) {

LinkedList \*result = (LinkedList \*)malloc(sizeof(LinkedList));

initLinkedList(result);

Node \*current1 = list1->head;

while (current1 != NULL) {

Node \*current2 = list2->head;

int found = 0;

while (current2 != NULL) {

if (current1->data == current2->data) {

found = 1;

break;

}

current2 = current2->next;

}

if (!found) {

addNode(result, current1->data);

}

current1 = current1->next;

}

return result;

}

int main() {

LinkedList list1, list2;

initLinkedList(&list1);

initLinkedList(&list2);

int n1, n2, data;

printf("输入数列1的元素个数: ");

scanf("%d", &n1);

printf("输入数列1中的元素: ");

for (int i = 0; i < n1; i++) {

scanf("%d", &data);

addNode(&list1, data);

}

printf("输入数列2的元素个数: ");

scanf("%d", &n2);

printf("输入数列2中的元素: ");

for (int i = 0; i < n2; i++) {

scanf("%d", &data);

addNode(&list2, data);

}

printf("数列1: ");

displayList(&list1);

printf("数列2: ");

displayList(&list2);

LinkedList \*intersect = intersection(&list1, &list2);

printf("交集: ");

displayList(intersect);

LinkedList \*unionset = unionSet(&list1, &list2);

printf("并集: ");

displayList(unionset);

LinkedList \*diff = difference(&list1, &list2);

printf("差集 (list1 - list2): ");

displayList(diff);

LinkedList \*diff1 = difference(&list2, &list1);

printf("差集 (list2 - list1): ");

displayList(diff1);

free(intersect);

free(unionset);

free(diff);

return 0;

}

