实验报告二 线性表

班级： 姓名： 学号： 专业：

1. 实验目的：
2. 理解线性表的逻辑结构、两种存储结构和数据操作；
3. 应用顺序表的基本算法实现集合A=AUB算法，应用顺序表的基本算法实现两有序顺序表的归并算法；
4. 掌握单链表的遍历、插入和删除等操作算法，实现多项式相加。
5. 实验内容：

1、设有线性表 LA＝（3,5,8,11）和 LB=（2,6,8,9,11,15,20），用顺序表实现线性表；

① 若LA和LB分别表示两个集合A和B，求新集合

A＝A U B（‘并’操作，相同元素不保留）；

预测输出：LA=（3,5,8,11,2,6,9,15,20）

实现代码：

package 数据结构JAVA.实验二.实验内容1;

import java.util.\*;

class SeqList<T> extends Object{

private Object[] element;

private int len;

public SeqList(int lenth) {

this.element=new Object[lenth];

this.len=0;

}

public SeqList() {

this(2048);

}

public SeqList(T[] element) {

this(element.length);

this.len=element.length;

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

this.element[i]=element[i];

}

}

public boolean isEmpty() {

return this.len==0;

}

public int size() {

return len;

}

public T get(int i) {

if(i<0||i>=this.len)

return null;

return (T)element[i];

}

public void set(int i,T x) {

if(x==null)

throw new NullPointerException("x=null");

if(i>=0&&i<this.len)

this.element[i]=x;

else

throw new java.lang.IndexOutOfBoundsException(i+"");

}

public SeqList<T> Mix(SeqList<T> a,SeqList<T> b) {

SeqList<T> result;

int len=a.len+b.len;

int index=0;

result=new SeqList<T>(len);

int i=0,j=0;

while(i<a.len&&j<b.len) {

if(Integer.*valueOf*(a.element[i].toString())<Integer.*valueOf*(b.element[j].toString())) {

result.element[index++]=a.element[i++];

}

else if(Integer.*valueOf*(a.element[i].toString())>Integer.*valueOf*(b.element[j].toString()))

result.element[index++]=b.element[j++];

else {

result.element[index++]=a.element[i++];

j++;

}

}

while(i<a.len)result.element[index++]=a.element[i++];

while(j<b.len)result.element[index++]=b.element[j++];

result.len=index;

return result;

}

public void toPrint() {

for(int i=0;i<this.len;i++)

System.*out*.print(this.element[i]+" ");

}

}

public class SequenceList{

public static void main(String[] args) {

int len=0;

int Len=0;

Scanner reader=new Scanner(System.*in*);

len=reader.nextInt();

Integer[] number=new Integer[len];

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

number[i]=reader.nextInt();

}

Len=reader.nextInt();

Integer[] Number=new Integer[Len];

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

Number[i]=reader.nextInt();

}

SeqList<Integer> a=new SeqList<Integer>(number);

SeqList<Integer> b=new SeqList<Integer>(Number);

SeqList<Integer> result=new SeqList<Integer>();

result=result.Mix(a, b);

result.toPrint();

}

}

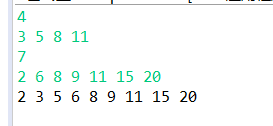
粘贴运行结果：

4

3 5 8 11

7

2 6 8 9 11 15 20



② 将LA与LB表归并，要求仍有序（相同元素要保留）

预测输出：LC=（2,3,5,6,8,8,9,11,11,15,20）

package 数据结构JAVA.实验二.实验内容1;

import java.util.\*;

class SeqList<T> extends Object{

private Object[] element;

private int len;

public SeqList(int lenth) {

this.element=new Object[lenth];

this.len=0;

}

public SeqList() {

this(2048);

}

public SeqList(T[] element) {

this(element.length);

this.len=element.length;

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

this.element[i]=element[i];

}

}

public boolean isEmpty() {

return this.len==0;

}

public int size() {

return len;

}

public T get(int i) {

if(i<0||i>=this.len)

return null;

return (T)element[i];

}

public void set(int i,T x) {

if(x==null)

throw new NullPointerException("x=null");

if(i>=0&&i<this.len)

this.element[i]=x;

else

throw new java.lang.IndexOutOfBoundsException(i+"");

}

public SeqList<T> Mix(SeqList<T> a,SeqList<T> b) {

SeqList<T> result;

int len=a.len+b.len;

int index=0;

result=new SeqList<T>(len);

int i=0,j=0;

while(i<a.len&&j<b.len) {

if(Integer.*valueOf*(a.element[i].toString())<=Integer.*valueOf*(b.element[j].toString())) {

result.element[index++]=a.element[i++];

}

else if(Integer.*valueOf*(a.element[i].toString())>Integer.*valueOf*(b.element[j].toString()))

result.element[index++]=b.element[j++];

else {

result.element[index++]=a.element[i++];

j++;

}

}

while(i<a.len)result.element[index++]=a.element[i++];

while(j<b.len)result.element[index++]=b.element[j++];

result.len=index;

return result;

}

public void toPrint() {

for(int i=0;i<this.len;i++)

System.*out*.print(this.element[i]+" ");

}

}

public class SequenceList{

public static void main(String[] args) {

int len=0;

int Len=0;

Scanner reader=new Scanner(System.*in*);

len=reader.nextInt();

Integer[] number=new Integer[len];

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

number[i]=reader.nextInt();

}

Len=reader.nextInt();

Integer[] Number=new Integer[Len];

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

Number[i]=reader.nextInt();

}

SeqList<Integer> a=new SeqList<Integer>(number);

SeqList<Integer> b=new SeqList<Integer>(Number);

SeqList<Integer> result=new SeqList<Integer>();

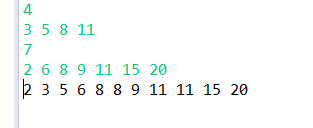
result=result.Mix(a, b);

result.toPrint();

}

}

粘贴运行结果：



2、在SinglyLinkedList类中增加下列成员方法。

public SinglyLinkedList(E[] element)//由指定数组中的多个对象构造单链表

public SinglyLinkedList(SinglyLinkedList list)//以单链表list构造新的单链表，复制单链表

public void concat(SinglyLinkedList list)//将指定单链表list链接在当前单链表之后

public Node<E> search(E element)//若查找到指定，则返回结点，否则返回null

public boolean contain (E element)//以查找结果判断单链表是否包含指定对象

public boolean remove (E element)//移去首次出现的指定对象

public boolean replace (Object obj, E element)//将单链表中的obj对象替换为对象element

public boolean equals(Object obj)//比较两条单链表是否相等

实现代码：

package 数据结构JAVA.实验二.实验内容2;

class Node<T>{

protected T data;

protected Node<T> next;

public Node(T data,Node<T> next) {

this.data=data;

this.next=next;

}

public Node() {

this(null,null);

}

public String toString() {

return this.data.toString();

}

}

public class SinglyLinkedList<T> extends Node<T>{

public Node<T> Head;

public SinglyLinkedList() {

this.Head=new Node<T>();

}

public SinglyLinkedList(T[] element) {

this();

Node<T> OperationNode=this.Head;

for(int i=0;i<element.length;i++) {

OperationNode.next=new Node<T>(element[i],null);

OperationNode=OperationNode.next;

}

}

public SinglyLinkedList(SinglyLinkedList<T> List){

this();

this.Head=List.Head;

}

public void concat(SinglyLinkedList<T> List) {

Node<T> OperationNode=this.Head;

Node<T> OperationNodeCat=List.Head.next;

while(OperationNode.next!=null)

OperationNode=OperationNode.next;

OperationNode.next=OperationNodeCat;

}

public Node<T> search(T element){

Node<T> OperationNode=this.Head;

while(OperationNode.next!=null) {

if(OperationNode.next.data.equals(element))

return OperationNode;

OperationNode=OperationNode.next;

}

return null;

}

public boolean contain(T element) {

Node<T> OperationNode=this.Head;

while(OperationNode.next!=null) {

if(OperationNode.next.data.equals(element))

return true;

OperationNode=OperationNode.next;

}

return false;

}

public boolean remove(T element) {

Node<T> OperationNode=this.search(element);

if(OperationNode.equals(null))

return false;

else {

OperationNode.next=OperationNode.next.next;

return true;

}

}

public boolean replace(Object obj,T element) {

Node<T> OperationNode=this.Head;

while(OperationNode.next!=null) {

if(OperationNode.next.data.equals(obj))

break;

OperationNode=OperationNode.next;

}

if(OperationNode.next.data.equals(null))

return false;

else {

OperationNode.next.data=element;

return true;

}

}

public boolean equals(SinglyLinkedList<T> obj) {

Node<T> OperationNode1=this.Head;

Node<T> OperationNode2=obj.Head;

while(!OperationNode1.next.equals(null)) {

if(!OperationNode1.next.equals(OperationNode2.next))

return false;

}

if(!OperationNode2.next.equals(null))

return false;

return true;

}

}

3、算法实现：多项式相加

一条单链表可以表示一个一元多项式，每个结点包含三个域：指数域、系数域和后继结点链。表示多项式的单链表如图1所示。给定两个多项式，实现两个多项式相加算法。

系数 指数 链

head

5 1

-10 0 ∧

-6 2

3 4

实现代码：

package 数据结构JAVA.实验二.实验内容2;

import java.io.Reader;

import java.math.\*;

import java.util.\*;

class Node<T>{

protected T data1,data2;

protected Node<T> next;

public Node(T data1,T data2,Node<T> next) {

this.data1=data1;

this.data2=data2;

this.next=next;

}

public Node() {

this(null,null,null);

}

public String toString() {

return this.data1.toString()+" "+this.data2.toString();

}

}

public class SinglyLinkedList<T> extends Node<T>{

public Node<T> Head;

public SinglyLinkedList() {

this.Head=new Node<T>();

}

public SinglyLinkedList(T data1,T data2) {

this();

this.Head.next.data1=data1;

this.Head.next.data2=data2;

}

public SinglyLinkedList(SinglyLinkedList<T> List){

this();

this.Head=List.Head;

}

public Node<T> findLast() {

Node<T> OperationNode=this.Head;

while(OperationNode.next!=null)

OperationNode=OperationNode.next;

return OperationNode;

}

public void add(T data1,T data2) {

Node<T> OperationNode=findLast();

OperationNode.next=new Node(data1,data2,null);

}

public static void main(String[] args) {

SinglyLinkedList<Integer> a;

a=new SinglyLinkedList<Integer>();

SinglyLinkedList<Integer> b;

b=new SinglyLinkedList<Integer>();

double result=0;

Scanner reader;

reader=new Scanner(System.*in*);

while(true) {

int data1,data2;

data1=reader.nextInt();

data2=reader.nextInt();

a.add(data1, data2);

if(data2==0)

break;

}

while(true) {

int data1,data2;

data1=reader.nextInt();

data2=reader.nextInt();

b.add(data1, data2);

if(data2==0)

break;

}

Node<Integer> OperationNode1=a.Head;

Node<Integer> OperationNode2=b.Head;

while(OperationNode1.next!=null) {

if(OperationNode1.next.data2.equals(OperationNode2.next.data2)) {

OperationNode1.next.data1+=OperationNode2.next.data1;

OperationNode1=OperationNode1.next;

OperationNode2=OperationNode2.next;

}else if(Integer.*valueOf*(OperationNode1.next.data2.toString())>Integer.*valueOf*(OperationNode2.next.data2.toString())) {

OperationNode1=OperationNode1.next;

}else if(Integer.*valueOf*(OperationNode1.next.data2.toString())<Integer.*valueOf*(OperationNode2.next.data2.toString())) {

Node<Integer> cope=OperationNode2.next;

cope.next=OperationNode1.next;

OperationNode1.next=cope;

OperationNode2=OperationNode2.next;

OperationNode1=OperationNode1.next;

//System.out.println(OperationNode1.next.data1+" \*\*\* "+OperationNode1.next.next.data1);

}

}

OperationNode1=a.Head;

while(OperationNode1.next!=null) {

System.*out*.println(OperationNode1.next.data1.toString()+" "+OperationNode1.next.data2.toString());

OperationNode1=OperationNode1.next;

}

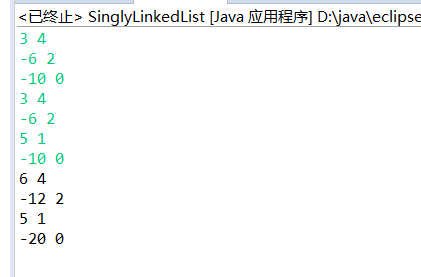
}}

粘贴运行结果：

3 4

-6 2

-10 0



3 4

-6 2

5 1

-10 0

1. 心得体会：（含上机中所遇问题的解决办法，所使用到的编程技巧、创新点及编程的心得）
2. 实验内容一，采用two points 算法，非常便捷的实现了及去重以及排序两个要求点。Two points排序法一般是归在归并排序中的，不过此次单独拿出来用效果也很不错。
3. 对于比较表中的两个对象时，他们是不能直接大小比较的，因为我们用的是泛型。我们可以采用Integrate.ValueOf(element.toString())来将对象转化成int型再进行比较。
4. Integrate类实质上就是int类型，在使用泛型时要使用前者。
5. 由于我采用了two points法排序，实验二只需添加一个等号就能实现，非常好用。
6. 操作链表时，我们最好先声明一个操作用的指针再进行一系列的操作，这样能让程序更整洁。