Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Ярославский государственный технический университет»

Кафедра «Информационные системы и технологии»

Отчет защищен

с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_

Преподаватель

\_\_\_\_\_\_\_\_\_Д.В.Дидковская

«\_\_\_»\_\_\_\_\_\_\_\_\_\_ 2022

**Линейные списки. очередь. стек. бинарные деревья**

Отчёт о лабораторной работе №6 по курсу “Информационные технологии”

ЯГТУ 09.03.02-024 ЛР

|  |  |
| --- | --- |
|  | Отчет выполнил  студент группы ЭИС-26  \_\_\_\_\_\_\_\_\_\_\_ А.А.Хрящев  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_ 2022 |

2022

Цель работы:

Ознакомился с линейными списками (однонаправленными и двунаправленными), очередью, стеком, бинарными деревьями.

Задание:

1. Однонаправленные линейные списки.

Создать метод, который удаляет из непустого списка L последний элемент

2. Двунаправленные линейные списки.

Создать метод, который меняет местами первый и последний элементы списка L.

3. Работа с очередью.

Создать программу, в которых в качестве поля d используются данные типа double. Организовать поиск заданного элемента.

4. Работа со стеком.

Создать программу, в которых в качестве поля d используются данные типа chat. Организовать поиск заданного элемента.

5. Двоичные деревья.

1. Создать двоичное дерево, элементы для которого взять из заданного целочисленного одномерного массива.

2. В созданном двоичном дереве организовать поиск заданного элемента.

3. В созданном двоичном дереве организовать просмотр его элементов любым способом (см. файл «Справка»).

Код программы:

1)

import io.reactivex.rxjava3.core.Observable;

@Value

@Builder(toBuilder = true)

public class Work6 {

@NonNull

OneWayLinearList oneWayLinearList;

public Observable<String> partOne() {

return Observable.create(observer -> {

//заполняем список

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

L.addBack(i+1);

}

observer.onNext("Список до удаления последнего элемента: ");

L.printList();

observer.onNext("\n\r");

L.delEnd();

observer.onNext("Список после удаления последнего элемента: ");

L.printList();

observer.onComplete();

});

}

}  
  
class ListElement {  
 ListElement next; // указатель на следующий элемент  
 int data; // поле данных типа int  
}  
  
class OneWayLinearList {  
  
 private ListElement head; // указатель на первый элемент  
 private ListElement tail; // указатель на последний элемент  
  
 void addFront(int data) //добавить спереди  
 {  
 ListElement a = new ListElement(); //создаём новый элемент  
 a.data = data; //инициализируем данные.  
// указатель на следующий элемент автоматически инициализируется как null  
 if(head == null) //если список пуст  
 { //то указываем ссылки начала и конца на новый элемент  
 head = a; //т.е. список теперь состоит из одного элемента  
 tail = a;  
 }  
 else {  
 a.next = head; //иначе новый элемент теперь ссылается на "бывший" первый  
 head = a; //а указатель на первый элемент теперь ссылается на новый элемент  
 }  
 }  
  
 void addBack(int data) { //добавление в конец списка  
 ListElement a = new ListElement(); //создаём новый элемент  
 a.data = data;  
 if (tail == null) //если список пуст  
 { //то указываем ссылки начала и конца на новый элемент  
 head = a; //т.е. список теперь состоит из одного элемента  
 tail = a;  
 }  
 else {  
 tail.next = a; //иначе "старый" последний элемент теперь ссылается на новый  
 tail = a; //а в указатель на последний элемент записываем адрес нового элемента  
 }  
 }  
  
 void printList() //печать списка  
 {  
 ListElement t = head; //получаем ссылку на первый элемент  
 while (t != null) //пока элемент существует  
 {  
 System.*out*.print(t.data + " "); //печатаем его данные  
 t = t.next; //и переключаемся на следующий  
 }  
 }  
  
 void delEl(int data) //удаление элемента  
 {  
 if(head == null) //если список пуст -  
 return; //ничего не делаем  
 if (head == tail) { //если список состоит из одного элемента  
 head = null; //очищаем указатели начала и конца  
 tail = null;  
 return; //и выходим  
 }  
 if (head.data == data) { //если первый элемент - тот, что нам нужен  
 head = head.next; //переключаем указатель начала на второй элемент  
 return; //и выходим  
 }  
 ListElement t = head; //иначе начинаем искать  
 while (t.next != null) { //пока следующий элемент существует  
 if (t.next.data == data) { //проверяем следующий элемент  
 if(tail == t.next) //если он последний  
 {  
 tail = t; //то переключаем указатель на последний элемент на текущий  
 }  
 t.next = t.next.next; //найденный элемент выкидываем  
 return; //и выходим  
 }  
 t = t.next; //иначе ищем дальше  
 }  
 }  
  
 void delEnd()  
 {  
 if (head == tail) { //если список состоит из одного элемента  
 head = null; //очищаем указатели начала и конца  
 tail = null;  
 return; //и выходим  
 }  
  
 ListElement t = head;  
 while (t!=null) {  
 if(tail == t.next) //если следующий элемент последний  
 {  
 tail = t; //то переключаем указатель с последнего элемент на текущий  
 t.next = t.next.next; //выкидываем последний элемент  
 return;  
 }  
 t = t.next;  
 }  
 }  
}  
  
public class Main {

public static void main(String[] args) {  
 Work6 work = Work6.builder().oneWayLinearList(new OneWayLinearList()).build();

work.partOne().subscribe(System.out::print);  
 }

}

2)

import Util.Work6.BidirectionalLinearList;

import io.reactivex.rxjava3.core.Observable;

import lombok.Builder;

import lombok.NonNull;

import lombok.Value;

@Value

@Builder(toBuilder = true)

public class Work6 {

@NonNull BidirectionalLinearList bidirectionalLinearList;

public Observable<String> partTwo() {

return Observable.create(observer -> {

//заполняем список

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

bidirectionalLinearList.addBack(i+1);

}

System.out.print("Список до перестановки: ");

bidirectionalLinearList.printList();

System.out.println();

bidirectionalLinearList.p\_end\_start();

System.out.print("Список после перестановки: ");

bidirectionalLinearList.printList();

observer.onComplete();

});

}

}  
  
class Item {  
 Item next; // указатель на следующий  
 Item prev; // указатель на предыдущий элемент  
 int data; // поле данных типа int  
}  
  
class BidirectionalLinearList {  
  
 private Item head; // указатель на первый элемент  
 private Item tail; // указатель на последний элемент  
  
 void addBack(int data) { //добавление в конец списка  
 Item a = new Item(); //создаём новый элемент  
 a.data = data;  
 if (tail == null) //если список пуст  
 { //то указываем ссылки начала и конца на новый элемент  
 head = a; //т.е. список теперь состоит из одного элемента  
 tail = a;  
 }  
 else {  
 tail.next = a; //иначе "старый" последний элемент теперь ссылается на новый  
 a.prev = tail; //а в указатель на последний элемент записываем адрес нового элемента  
 tail = a;  
 }  
 }  
  
 void printList() //печать списка  
 {  
 Item t = head; //получаем ссылку на первый элемент  
 while (t != null) //пока элемент существует  
 {  
 System.*out*.print(t.data + " "); //печатаем его данные  
 t = t.next; //и переключаемся на следующий  
 }  
 }  
  
 void delEl(int data) //удаление элемента  
 {  
 if(head == null) //если список пуст -  
 return; //ничего не делаем  
 if (head == tail) { //если список состоит из одного элемента  
 head = null; //очищаем указатели начала и конца  
 tail = null;  
 return; //и выходим  
 }  
 if (head.data == data) { //если первый элемент - тот, что нам нужен  
 head = head.next; //переключаем указатель начала на второй элемент  
 return; //и выходим  
 }  
 Item t = head; //иначе начинаем искать  
 while (t.next != null) { //пока следующий элемент существует  
 if (t.next.data == data) { //проверяем следующий элемент  
 if(tail == t.next) //если он последний  
 {  
 tail = t; //то переключаем указатель на последний элемент на текущий  
 }  
 t.next = t.next.next; //найденный элемент выкидываем  
 return; //и выходим  
 }  
 t = t.next; //иначе ищем дальше  
 }  
 }  
  
 void delEnd()  
 {  
 if (head == tail) { //если список состоит из одного элемента  
 head = null; //очищаем указатели начала и конца  
 tail = null;  
 return; //и выходим  
 }  
  
 Item t = head;  
 while (t!=null) {  
 if(tail == t.next) //если следующий элемент последний  
 {  
 tail = t; //то переключаем указатель с последнего элемент на текущий  
 t.next = t.next.next; //выкидываем последний элемент  
 return;  
 }  
 t = t.next;  
 }  
 }  
  
 void p\_end\_start() {  
 int a = tail.data;  
 int b= head.data;  
 head.data = a;  
 tail.data = b;  
  
 }  
}  
  
public class Main {  
  
 public static void main(String[] args) {

Work6 work = Work6.builder()

.bidirectionalLinearList(new BidirectionalLinearList())

.build();

work.partTwo().subscribe(System.out::print);  
 }  
}

3)

@Value

@Builder(toBuilder = true)

public class Work6 {

@NonNull Queue queue;

public Observable<String> partThree() {

return Observable.create(observer -> {

//заполняем список

queue.first(1.9);

for (double i=2;i<2.6;i=i+0.1){

queue.add(i);

}

while (queue.pbeg!=null){

System.out.print(" "+ queue.pop());

}

System.out.println();

queue.first(1.9);

for (double i=2;i<2.6;i=i+0.1){

queue.add(i);

}

queue.poisk(2.1);

queue.poisk(1.8);

observer.onComplete();

});

}

}  
  
class Item {  
 double d; // ключ  
 Item next;  
  
 // конструктор:  
 Item(double value){  
 d=value;  
 }  
}  
  
class Queue {  
 Item pbeg=null; // указатель на начало очереди  
 Item pend=null; // указатель на конец очереди  
  
 void first(double value){  
 Item pv=new Item(value);  
 pv.next=null;  
 pbeg=pv;  
 pend=pbeg;  
 }  
  
 // Добавление в конец очереди  
 void add(double value){  
 Item pv=new Item(value);  
 pv.next=null;  
 pend.next=pv;  
 pend=pv;  
 }  
  
 // Выборка из начала очереди с удалением  
 double pop (){  
 double temp=pbeg.d;  
 pbeg=pbeg.next;  
 return temp;  
 }  
  
 void poisk(double a){  
 boolean t=false; //проверка найден ли элемент  
 while (pbeg!=null){ //пока очередь не пуста  
 if (pbeg.d == a) {  
 System.out.println("Элемент " + a + " найден!");  
 t = true;  
 break;  
 }  
 pbeg = pbeg.next;  
 }  
 if (!t) System.out.println("Элемент " + a + " не найден!");  
 }  
}  
  
public class Main {  
 public static void main(String[] args) {  
 Work6 work = Work6.builder()

.queue(new Queue())

.build();

work.partThree().subscribe(System.out::print);  
 }  
}

4)

@Value

@Builder(toBuilder = true)

public class Work6 {

@NonNull Stack stack;

public Observable<String> partFour() {

return Observable.create(observer -> {

stack.first('a');

char k;

for (char i='b';i<'e';i++){

stack.push(i);

}

while (stack.top!=null){

observer.onNext(" "+ stack.pop());

}

observer.onNext("\n\r");

stack.first('a');

for (char i='b';i<'e';i++){

stack.push(i);

}

stack.poisk('a');

});

}

}

class Item{  
 char d;//ключ  
 Item next;  
  
 //конструктор:  
 Item(char value){  
 d=value;  
 }  
}  
  
class Stack {  
 Item top=null;//указатель на вершину стека  
  
 void first(char value){  
 Item pv=new Item(value);  
 pv.next=null;  
 top=pv;  
 }  
  
//Занесение в стек  
 void push(char value){  
 Item pv=new Item(value);  
 pv.next=top;  
 top=pv;  
 }  
  
//Выборка из стека  
 char pop (){  
 char temp=top.d;  
 top=top.next;  
 return temp;  
 }  
  
 void poisk(char a){  
 boolean t=false; //проверка найден ли элемент  
 while (top!=null){  
 if (top.d == a) {  
 System.out.println("Элемент " + a + " найден!");  
 t = true;  
 break;  
 }  
 top = top.next;  
 }  
 if (!t) System.out.println("Элемент " + a + " не найден!");  
 }  
}

5)

@Value

@Builder(toBuilder = true)

public class Work6 {

@NonNull Tree tree;

public Observable<String> partFive() {

return Observable.create(observer -> {

int b[] = {10, 25, 20, 6, 21, 8, 1, 30};

Tree m = new Tree();

m.first(b[0]);

for (int i = 1; i < 8; i++)

m.search\_insert(b[i]);

m.print\_tree(m.root);

m.poisk(-4);

m.poisk(20);

});

}

}

class Item{  
 int d;//ключ  
 Item left;  
 Item right;  
  
 //конструктор:  
 Item(int value){  
 d=value;  
 }  
}  
  
class Tree{  
 Item root;// корень дерева  
  
 //формирование первого элемента дерева  
 void first(int value){  
 Item pv=new Item(value);  
 pv.left=null;  
 pv.right=null;  
 root=pv;  
 }  
  
 //поиск с включением  
 void search\_insert(int value){  
 Item pv=root;  
 Item prev=null;  
 while(pv!=null){  
 prev=pv;  
 if (value==-pv.d) return;  
 else if (value<pv.d) pv=pv.left;  
 else pv=pv.right;  
 }  
  
 //Создание нового узла  
 Item pnew=new Item(value);  
 pnew.left=null;  
 pnew.right=null;  
 if (value<prev.d)  
 prev.left=pnew;//присоединение к левому поддереву предка  
 else  
 prev.right=pnew;//присоединение к правому поддереву предка  
 }  
  
 //Обход дерева  
 void print\_tree (Item p){  
 if (p!=null){  
 System.*out*.println(" "+p.d);  
 print\_tree (p.left);//вывод левого поддерева  
 print\_tree (p.right);//вывод правого поддерева  
 }  
 }  
  
 void poisk(int value){  
 boolean t = false;//проверка найден ли элемент  
 Item pv=root;  
 Item prev=null;  
 while(pv!=null){  
 prev=pv;  
 if (value==pv.d){  
 System.*out*.println("Элемент "+ value + " найден!");  
 return;}  
 else if (value<pv.d) pv=pv.left;  
 else pv=pv.right;  
 }  
 if (!t) System.*out*.println("Элемент "+ value+" не найден!");  
 }  
}

Скриншоты выполнения:



Рисунок 1 – Результат выполнения задания 1



Рисунок 2 – Результат выполнения задания 2

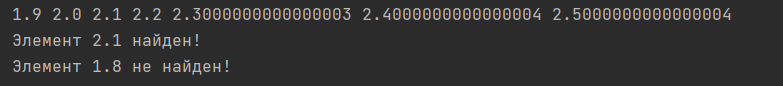


Рисунок 3 – Результат выполнения задания 3



Рисунок 4 – Результат выполнения задания 4

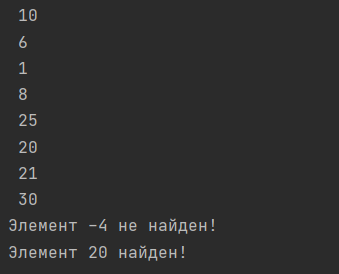


Рисунок 5 – Результат выполнения задания 5

Вывод:

Я ознакомился с линейными списками (однонаправленными и двунаправленными), очередью, стеком, бинарными деревьями, и их реализацией на языке Java. В результате выполнил 6 лабораторную работу.