НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ФАКУЛЬТЕТ ІНФОРМАТИКИ І ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Лабораторна робота №1

з дисципліни **«**Системне програмування 2**»**

Виконав:

студент групи ІО-42, ФІОТ

Коваленко В’ячеслав Сергійович

ЗК : 4209

м. Київ 2016 р.

Варіант

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 9 | unsigned short | char\*\_ unsigned short | union | Черговий |

Співпадіння максимуму початкових літер, включаючи літери латинського i слов'янського алфавіту, що співпадають за написанням інваріантно до регістру літер.

Лістинг програми

public class UnicodeCoincidence {  
 //A, B, C, E, H, K, M, O, P, T, X, Y  
 public static int[][] *unicodeCoincidence* = {{65, 97, 1040, 1072},  
 {66, 98, 1042, 1074},  
 {67, 99, 1057, 1089},  
 {69, 101, 1045, 1077},  
 {72, 104, 1053, 1085},  
 {75, 107, 1050, 1083},  
 {77, 109, 1052, 1084},  
 {79, 111, 1054, 1086},  
 {80, 112, 1056, 1088},  
 {84, 116, 1058, 1090},  
 {88, 120, 1061, 1093},  
 {89, 121, 1059, 1091}};  
}

public class TableRow {  
  
 private short directAddressKey;  
 private String foreignAddressKey;  
 private Union functionalField;  
  
 public TableRow(short directAddressKey, String foreignAddressKey, Union functionalField) {  
 this.directAddressKey = directAddressKey;  
 this.foreignAddressKey = foreignAddressKey;  
 this.functionalField = functionalField;  
 }  
  
 public short getDirectAddressKey() {  
 return directAddressKey;}  
  
 public void setDirectAddressKey(short directAddressKey) {  
 this.directAddressKey = directAddressKey;}  
  
 public String getForeignAddressKey() {  
 return foreignAddressKey;}  
  
 public void setForeignAddressKey(String foreignAddressKey) {  
 this.foreignAddressKey = foreignAddressKey; }  
  
 public Union getFunctionalField() {  
 return functionalField; }  
  
 public void setFunctionalField(Union functionalField) {  
 this.functionalField = functionalField; }  
  
 public void update(TableRow updatedTableRow) {  
 this.directAddressKey = updatedTableRow.getDirectAddressKey();  
 this.foreignAddressKey = updatedTableRow.getForeignAddressKey();  
 this.functionalField = updatedTableRow.getFunctionalField();}  
  
 @Override  
 public String toString() {  
 return "TableRow{" +  
 "directAddressKey=" + directAddressKey +  
 ", foreignAddressKey='" + foreignAddressKey + '\'' +  
 ", functionalField=" + functionalField.toString() +  
 '}'; } }

public class Table {  
  
 private ArrayList<TableRow> mainTable;  
  
 public Table(ArrayList<TableRow> rowList) {  
 this.mainTable = rowList;}  
  
 private void showRow(int currentIndex) {  
 System.*out*.println(mainTable.get(currentIndex).toString());}  
  
 private boolean isCharInUnicodeCoincidence(char currentChar) {  
 for (int i = 0; i < UnicodeCoincidence.*unicodeCoincidence*.length; i++) {  
 for (int j = 0; j < 4; j++) {  
 if ((int) currentChar == UnicodeCoincidence.*unicodeCoincidence*[i][j]) return true;  
 }  
 }  
 return false;  
 }  
 private int getCharRowInUnicodeCoincidence(char currentChar) {  
 for (int i = 0; i < UnicodeCoincidence.*unicodeCoincidence*.length; i++) {  
 for (int j = 0; j < 4; j++) {  
 if ((int) currentChar == UnicodeCoincidence.*unicodeCoincidence*[i][j]) return i;  
 }  
 }  
 return -1;  
 }  
 private int getRowIndexByDirectAdrKey(short directKey) {  
 for (int i = 0; i < mainTable.size(); i++) {  
 if (mainTable.get(i).getDirectAddressKey() == directKey) return i;  
 }  
 return -1;  
 }

private boolean isAlreadyInTable(short directAdrKey, String foreignAdrKey){  
 for (TableRow row : mainTable) {  
 if((row.getDirectAddressKey() == directAdrKey) || (row.getForeignAddressKey().compareToIgnoreCase(foreignAdrKey) == 0)) return false; }  
 return true; }  
 private int getRowIndexByForeignAddressKey(String foreignKey) {  
 int resultIndex = -1;  
 int maxNumberOfIdenticalChars = -1;  
 int currentNumberOfIdenticalChars;  
  
 if ((foreignKey != null) || (foreignKey.length() > 0)) {  
  
 for (int i = 0; i < mainTable.size(); i++) {  
 String currentForeignKey = mainTable.get(i).getForeignAddressKey();  
 currentNumberOfIdenticalChars = 0;  
 int comparisonLength = (currentForeignKey.length() >= foreignKey.length()) ? foreignKey.length() : currentForeignKey.length();  
 for (int j = 0; j < comparisonLength; j++) {  
 if (isCharInUnicodeCoincidence(currentForeignKey.charAt(j)) && isCharInUnicodeCoincidence(foreignKey.charAt(j))) {  
 // check row numbers inside unicodeTable  
 if (getCharRowInUnicodeCoincidence(currentForeignKey.charAt(j)) == getCharRowInUnicodeCoincidence(foreignKey.charAt(j))) {  
 currentNumberOfIdenticalChars++;  
 } else break;  
 } else {  
 if ((currentForeignKey.substring(j,j + 1).compareToIgnoreCase(foreignKey.substring(j,j + 1))) == 0) {  
 currentNumberOfIdenticalChars++;  
 } else break;  
 }  
 }  
 if ((currentNumberOfIdenticalChars != 0) && (currentNumberOfIdenticalChars >= maxNumberOfIdenticalChars)) {  
 maxNumberOfIdenticalChars = currentNumberOfIdenticalChars;  
 resultIndex = i;  
 } } }  
 return resultIndex;  
 }  
 public void addRow(TableRow tableRow) {  
 mainTable.add(tableRow);  
 }  
 public void showTable(ArrayList<TableRow> mainTable) {  
 for (TableRow row : mainTable) {  
 System.*out*.println(row.toString());  
 } }  
 public void showTable() {  
 for (TableRow row : mainTable) {  
 System.*out*.println(row.toString());  
 }  
 }  
  
 // --- Linear search ---  
  
 // - direct address -  
 public void createRowDirectAdr(TableRow newTableRow) {

if(isAlreadyInTable(newTableRow.getDirectAddressKey(), newTableRow.getForeignAddressKey())) mainTable.add(newTableRow);  
else System.*out*.println("Already exist with identical key");  
}  
 public void readRowDirectAdr(short directKey) {  
 int currentIndex = getRowIndexByDirectAdrKey(directKey);  
 if (currentIndex != -1) showRow(currentIndex);  
 }  
 public void updateRowDirectAdr(short directKey, TableRow updatedTableRow) {  
 int currentIndex = getRowIndexByDirectAdrKey(directKey);  
 if (currentIndex != -1) mainTable.get(currentIndex).update(updatedTableRow);  
 else System.*out*.println("No such field");  
 }  
 public void deleteRowDirectAdr(short directKey) {  
 int currentIndex = getRowIndexByDirectAdrKey(directKey);  
 if (currentIndex != -1) mainTable.remove(currentIndex);  
 else System.*out*.println("No such field");  
 }  
 // - foreign address -  
 public void createRowForeignAdr(TableRow newTableRow) {  
if(isAlreadyInTable(newTableRow.getDirectAddressKey(), newTableRow.getForeignAddressKey())) mainTable.add(newTableRow);  
else System.*out*.println("Already exist with identical key");

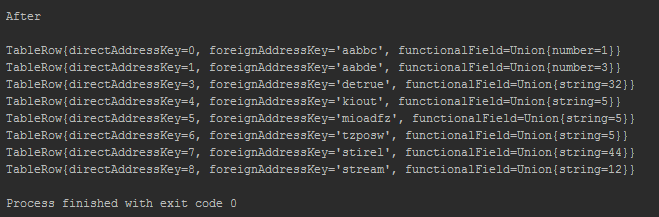
}  
 public void readRowForeignAdr(String foreignKey) {  
 int currentIndex = getRowIndexByForeignAddressKey(foreignKey);  
 if (currentIndex != -1) showRow(currentIndex);  
 else System.*out*.println("No such field");  
 }  
 public void updateRowForeignAdr(String foreignKey, TableRow updatedTableRow) {  
 int currentIndex = getRowIndexByForeignAddressKey(foreignKey);  
 if (currentIndex != -1) mainTable.get(currentIndex).update(updatedTableRow);  
 else System.*out*.println("No such field");  
 }  
 public void deleteRowForeignAdr(String foreignKey) {  
 int currentIndex = getRowIndexByForeignAddressKey(foreignKey);  
 if (currentIndex != -1) mainTable.remove(currentIndex);  
 else System.*out*.println("No such field");  
 }  
 // --- Binary search ---  
 private int getNumberOfIdenticalChars(String currentForeignKey, String foreignKey) {  
 int currentNumberOfIdenticalChars = 0;  
 int comparisonLength = (currentForeignKey.length() > foreignKey.length()) ? foreignKey.length() : currentForeignKey.length();  
  
 for (int j = 0; j < comparisonLength; j++) {  
 if (isCharInUnicodeCoincidence(currentForeignKey.charAt(j)) && isCharInUnicodeCoincidence(foreignKey.charAt(j)))   
 // check row numbers inside unicodeTable  
 if (getCharRowInUnicodeCoincidence(currentForeignKey.charAt(j)) == getCharRowInUnicodeCoincidence(foreignKey.charAt(j))) {  
 currentNumberOfIdenticalChars++;  
 } else break;  
 } else {  
 if ((currentForeignKey.substring(j,j + 1).compareToIgnoreCase(foreignKey.substring(j,j + 1))) == 0) {  
 currentNumberOfIdenticalChars++;  
 } else break;  
 }  
 }  
 return currentNumberOfIdenticalChars;  
 }  
 public void getRowWithBinarySearch(String foreignKey) {  
 int currentIndex = -1;  
 ArrayList<TableRow> tempTableList = (ArrayList<TableRow>) mainTable.clone();  
 ComparatorByKey comparator = new ComparatorByKey();  
 Collections.*sort*(tempTableList, comparator);  
  
 System.*out*.println("Sorted table : ");  
 showTable(tempTableList);  
 int start = 0;  
 int end = tempTableList.size()-1;  
 int middle;  
  
 while (start < end) {  
 middle = (start + end) / 2;  
  
 if ((end - start) == 1) {  
 int startValue = getNumberOfIdenticalChars(tempTableList.get(start).getForeignAddressKey(), foreignKey);  
 int endValue = getNumberOfIdenticalChars(tempTableList.get(end).getForeignAddressKey(), foreignKey);  
  
 if ((startValue == endValue) && (startValue == 0)) {  
 currentIndex = -1;  
 break;  
 } else if ((startValue == endValue) && (startValue != 0)) {  
 currentIndex = end;  
 break;  
 } else if (startValue > endValue) {  
 currentIndex = start;  
 break;  
 } else {  
 currentIndex = end;  
 break;  
 }  
 }  
 if ((middle == 0) || (middle == (tempTableList.size() - 1))) {  
 currentIndex = middle;  
 break;  
 }  
 int numberSameCharsCurrent = getNumberOfIdenticalChars(tempTableList.get(middle).getForeignAddressKey(), foreignKey);  
 int numberSameCharsPrevious = getNumberOfIdenticalChars(tempTableList.get(middle - 1).getForeignAddressKey(), foreignKey);  
 int numberSameCharsNext = getNumberOfIdenticalChars(tempTableList.get(middle + 1).getForeignAddressKey(), foreignKey);  
 if ((numberSameCharsCurrent > numberSameCharsPrevious) && (numberSameCharsCurrent > numberSameCharsNext)) {  
 currentIndex = middle;  
 break;  
 } else {  
 int min = numberSameCharsCurrent, max = numberSameCharsCurrent;  
  
 for (int i = (middle - 1); i >= start; i--) {  
 if (getNumberOfIdenticalChars(tempTableList.get(i).getForeignAddressKey(), foreignKey) != numberSameCharsCurrent) {  
 min = getNumberOfIdenticalChars(tempTableList.get(i).getForeignAddressKey(), foreignKey);  
 break;  
 } }  
 for (int i = (middle + 1); i <= end; i++) {  
 if (getNumberOfIdenticalChars(tempTableList.get(i).getForeignAddressKey(), foreignKey) != numberSameCharsCurrent) {  
 max = getNumberOfIdenticalChars(tempTableList.get(i).getForeignAddressKey(), foreignKey);  
 break;  
 } }  
 if ((min == max) && (min == numberSameCharsCurrent)) {  
 if (min <= 0) {  
 currentIndex = -1;  
 break;  
 } else start = middle + 1;  
 } else if (numberSameCharsCurrent < max) start = middle + 1;  
 else if (numberSameCharsCurrent < min) end = middle - 1;  
 else if (min <= numberSameCharsCurrent) start = middle;  
 } }  
 if(start == end) currentIndex = start;  
 if (currentIndex != -1) System.*out*.println(tempTableList.get(currentIndex).getForeignAddressKey());  
 else System.*out*.println("No such field");}  
 private class ComparatorByKey implements Comparator<TableRow> {  
 @Override  
 public int compare(TableRow o1, TableRow o2) {  
 if (o1 == null || o2 == null) return 0;  
 return o1.getForeignAddressKey().compareToIgnoreCase(o2.getForeignAddressKey());  
 }  
 } }

public class Main {  
 public static void main(String[] args) {  
 ArrayList<TableRow> rowList = new ArrayList<>(5);  
 rowList.add(new TableRow((short) 0, "aabbc", new Union(1)));  
 rowList.add(new TableRow((short) 1, "aabde", new Union(3)));  
 rowList.add(new TableRow((short) 2, "abjko", new Union(2)));  
 rowList.add(new TableRow((short) 3, "detrue", new Union("32")));  
 rowList.add(new TableRow((short) 4, "kiout", new Union("5")));  
 rowList.add(new TableRow((short) 5, "mioadfz", new Union("5")));  
 rowList.add(new TableRow((short) 6, "tzposw", new Union("5")));  
  
 Table table = new Table(rowList);  
 System.*out*.println("Before");  
 table.showTable();  
 System.*out*.println(" ");  
  
 //table.createRowDirectAdr(new TableRow((short) 9, "LG", new Union("54")));  
 //table.readRowDirectAdr((short) 2);  
 //table.updateRowDirectAdr((short) 3, new TableRow((short) 7, "LG", new Union("54")));  
 //table.deleteRowDirectAdr((short) 4);  
  
 //table.createRowForeignAdr(new TableRow((short) 7, "zoo", new Union("54")));  
 //table.readRowForeignAdr("АА");  
 //table.updateRowForeignAdr("AA", new TableRow((short) 7, "LG", new Union("54")));  
 //table.deleteRowForeignAdr("AВ");  
  
 System.*out*.println(" ");  
 System.*out*.println("After");  
 System.*out*.println(" ");  
  
 table.showTable();  
 } }

public class Union {  
 private int number;  
 private String string;  
 public Union(int number) {  
 this.number = number;  
 this.string = "-1";}  
 public Union(String string) {  
 this.string = string;  
 this.number = -1; }  
 public int getNumber() {  
 return number; }  
 public void setNumber(int number) {  
 this.number = number; }  
 public String getString() {  
 return string;}  
 public void setString(String string) {  
 this.string = string; }  
 @Override  
 public String toString() {  
 if (string.equals("-1"))  
 return "Union{" +  
 "number=" + number + '}';  
 else return "Union{" +  
 "string=" + string + '}';  
 } }

Результати програми

table.deleteRowForeignAdr("AВ");



Висновок

В ході даної лабораторної роботи була створена програма з наступним функціоналом – додати, видалити, змінити, вибрати рядок з таблиці використовуючи ключ для прямої адреси та ключ для інших видів пошуку. Також був розроблений алгоритм бінарного пошуку.