Федеральное агентство связи

Ордена трудового красного знамени

Федеральное государственное образовательное

Бюджетное

Учреждение высшего профессионального образования

Московский Технический Университет связи и информатики

Кафедра информатики

Лабораторная работа №4

по дисциплине «СиАОД»

«Реализация стека/дека»

Выполнил: студ. гр. БСТ1902

Потрываев А.Г

Вариант №15

Проверил: Кутейников И.А.

Москва 2021

Содержание

2 Ход выполнения работы 5

Часть 1 5

Часть 2 7

Реализовать следующие структуры данных:

● Стек (stack): операции для стека: инициализация, проверка на пустоту, добавление нового элемента в начало, извлечение элемента из начала;

● Дек (двусторонняя очередь, deque): операции для дека: инициализация, проверка на пустоту, добавление нового элемента в начало, добавление нового элемента в конец, извлечение элемента из начала, извлечение элемента из конца.

Разработать программу обработки данных, содержащихся в заранее подготовленном txt-файле, в соответствии с заданиями, применив указанную в задании структуру данных. Результат работы программы вывести на экран и сохранить в отдельном txt-файле.

Оформить отчет о лабораторной работе в ipynb или pdf-файле.

Задания:

1. Отсортировать строки файла, содержащие названия книг, в алфавитном порядке с использованием двух деков.

2. Дек содержит последовательность символов для шифровки сообщений. Дан текстовый файл, содержащий зашифрованное сообщение. Пользуясь деком, расшифровать текст. Известно, что при шифровке каждый символ сообщения заменялся следующим за ним в деке по часовой стрелке через один.

3. Даны три стержня и n дисков различного размера. Диски можно надевать на стержни, образуя из них башни. Перенести n дисков со стержня A на стержень C, сохранив их первоначальный порядок. При переносе дисков необходимо соблюдать следующие правила: - на каждом шаге со стержня на стержень переносить только один диск; - диск нельзя помещать на диск меньшего размера;

4. Дан текстовый файл с программой на алгоритмическом языке. За один просмотр файла проверить баланс круглых скобок в тексте, используя стек.

5. Дан текстовый файл с программой на алгоритмическом языке. За один просмотр файла проверить баланс квадратных скобок в тексте, используя дек.

6. Дан файл из символов. Используя стек, за один просмотр файла напечатать сначала все цифры, затем все буквы, и, наконец, все остальные символы, сохраняя исходный порядок в каждой группе символов.

7. Дан файл из целых чисел. Используя дек, за один просмотр файла напечатать сначала все отрицательные числа, затем все положительные числа, сохраняя исходный порядок в каждой группе.

8. Дан текстовый файл. Используя стек, сформировать новый текстовый файл, содержащий строки исходного файла, записанные в обратном порядке: первая строка становится последней, вторая – предпоследней и т.д.

9. Дан текстовый файл. Используя стек, вычислить значение логического выражения, записанного в текстовом файле в следующей форме: < ЛВ > ::= T | F | (N) | (A) | (X) | (O), где буквами обозначены логические константы и операции: T – True, F – False, N – Not, A – And, X – Xor, O – Or.

10. Дан текстовый файл. В текстовом файле записана формула следующего вида: ::= | M(,) | N(Формула>,) < Цифра > ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 где буквами обозначены функции: M – определение максимума, N – определение минимума. Используя стек, вычислить значение заданного выражения. 11. Дан текстовый файл. Используя стек, проверить, является ли содержимое текстового файла правильной записью формулы вида: < Формула > ::= < Терм > | < Терм > + < Формула > | < Терм > - < Формула > < Терм > ::= < Имя > | (< Формула >) < Имя > ::= x | y |x

2 Ход выполнения работы

Часть 1

Реализация стека\дека

```
open class MyStack<E>() {

    protected var list = LinkedList<E>()



    fun push(e:E){
        if(list.isEmpty())
            list.add(e)
        else {
            val newList = LinkedList<E>(listOf(e))
            newList.addAll(list)
            list = newList
        }
    }

    fun size() = list.size;

    fun isEmpty() = list.isEmpty()

    fun pop():E?{
        if(list.isEmpty()) return null;
```

```
        val temp = list[0]
        list.removeAt(0)
        return temp;
    }

    fun peek():E? {
        if (list.isEmpty()) return null
        return list[0]
    }

}

class MyDeque<E> : MyStack<E>() {

    fun popLast():E?{
        if(list.isEmpty()) return null
        val temp = list[list.size-1]
        list.removeAt(list.size-1)
        return temp;
    }

    fun peekLast():E?{
        if(list.isEmpty()) return null
        return list[list.size-1]
    }

    fun pushLast(e:E) = list.add(e)
```

}

Часть 2

Листинг выполнения задач

```
un main() {
    println("Task 1:\n")
    println(bookSort())
    println("Task 2:\n")
    println(coder())
    println("Task 3:\n")
    println(tower())
    println("Task 4:\n")
    println(check_brackets())
    println("Task 5:\n")
    println(check_square_brackets())
    println("Task 6:\n")
    println(write_by_order())
    println("Task 7:\n")
    println(numbers_sort())
    println("Task 8:\n")
    println(reversText())
    println("Task 9:\n")
    println(logicFun())
    println("Task 10:\n")
    println(maxFunction())
    println("Task 11:\n")
    println(funCulculate())
    write_to_res_file()
```

```
}

fun write_to_res_file() {
    File("resoult.txt").printWriter().use { out ->
        out.println("Task 1:\n")
        out.println(bookSort())
        out.println("Task 2:\n")
        out.println(coder())
        out.println("Task 3:\n")
        out.println(tower())
        out.println("Task 4:\n")
        out.println(check_brackets())
        out.println("Task 5:\n")
        out.println(check_square_brackets())
        out.println("Task 6:\n")
        out.println(write_by_order())
        out.println("Task 7:\n")
        out.println(numbers_sort())
        out.println("Task 8:\n")
        out.println(reversText())
        out.println("Task 9:\n")
        out.println(logicFun())
        out.println("Task 10:\n")
        out.println(maxFunction())
        out.println("Task 11:\n")
        out.println(funCulculate())
    }
}

fun createFile(s: String): File {
```

```
    val file = File("$s.txt")
    if (!file.exists()) {
        file.createNewFile()
    }
    return file
}

fun bookSort(): String {
    val result = MyDeque<String>();
    val temp = MyDeque<String>();
    val file = createFile("books")
    try {
        for (line in file.readLines()) {
            if (result.isEmpty())
                result.push(line)
            else {
                while (true) {
                    if (result.size() == 0) {
                        result.push(line)
                        for (i in 1..temp.size())
                            result.pushLast(temp.popLast()!!)
                        break
                    }
                    if (isBiggerString(line, result.peekLast()!!)) {
                        result.pushLast(line)
                        if (!temp.isEmpty())
                            for (i in 1..temp.size())
                                result.pushLast(temp.popLast()!!)
                        break
                    } else
```

```
                    temp.pushLast(result.popLast()!!)
                }
            }
        }
    } catch (ex: IOException) {
        println(ex.message)
    }
    var res = ""
    for (i in 1..result.size())
        res += result.pop() + "\n"
    return res

}

fun isBiggerString(a: String, b: String): Boolean {
    val arr = listOf<String>(a, b)
    Collections.sort(arr);
    return listOf(b, a) == arr
}

// TASK 2

fun coder(): String {
    //val string = "иъзйгдршыэцтаежючхявёлоксмуьщбнпф"
    val deque = MyDeque<Char>()
    val file = createFile("cipher")
    var string = file.bufferedReader().readLine()
    for (ch in string)
        deque.push(ch)
```

```
    val file2 = createFile("CipherText")
    if (!file.exists()) {
        file.createNewFile()
    }
    val text = file2.bufferedReader().readLine()
    var encoded = ""
    for (char in text) {
        if (char != encode(char, deque))
            encoded += encode(char, deque)
        else
            encoded += char

    }
    var res = ""
    res += encoded + "\n"

    var decoded = ""
    for (char in encoded) {
        if (char != decode(char, deque))
            decoded += decode(char, deque)
        else
            decoded += char

    }
    res += decoded
    return res
}

fun encode(c: Char, deque: MyDeque<Char>): Char? {
```

```
    for (i in 1..deque.size()) {
        val x = deque.popLast()
        if (x == c) {
            deque.push(x)
            val t = deque.popLast()
            deque.push(t!!)
            return t
        }
        deque.push(x!!)
    }
    return c
}

fun decode(c: Char, deque: MyDeque<Char>): Char? {
    for (i in 1..deque.size()) {
        val x = deque.popLast()
        if (x == c) {
            deque.pushLast(x)
            val t = deque.pop()
            deque.pushLast(t!!)
            return t
        }
        deque.push(x!!)
    }
    return c
}
//TASK 3

fun tower(): String {
    val A = MyStack<Int>()
```

```
    val B = MyStack<Int>()
    val C = MyStack<Int>()

    val file = createFile("tower")
    val disks = Integer.parseInt(file.bufferedReader().readLine())


    for (i in disks downTo 0)
        A.push(i)


    if (disks % 2 == 0) {
        while (C.size() != disks) {
            move(A, B)
            move(A, C)
            move(B, C)
        }
    } else
        while (C.size() != disks) {
            move(A, C)
            move(A, B)
            move(B, C)
        }
    var res: String = ""
    while (!C.isEmpty())
        res += C.pop().toString() + "\n"
    return res
}
```

```
fun move(a: MyStack<Int>, b: MyStack<Int>) {
    if (a.size() == 0 && b.size() > 0)
        a.push(b.pop()!!)
    else if (a.size() > 0 && b.size() == 0)
        b.push(a.pop()!!)
    else if (a.peek()!! > b.peek()!!)
        a.push(b.pop()!!)
    else
        b.push(a.pop()!!)
}

//TASK 4

fun check_brackets(): Boolean {
    val file = createFile("brackets")
    val string = file.bufferedReader().readLine()
    val bracket_stack = MyStack<Char>()
    for (i in string) {
        if (i == '(')
            bracket_stack.push(i)
        else if (i == ')') {
            if (bracket_stack.isEmpty())
                return false
            bracket_stack.pop()
        }
    }
    return bracket_stack.isEmpty()
}

//TASK 5
```

```
fun check_square_brackets(): Boolean {
    val file = createFile("squareBrackets")
    val string = file.bufferedReader().readLine()
    val bracket_stack = MyStack<Char>()
    for (i in string) {
        if (i == '[')
            bracket_stack.push(i)
        else if (i == ']') {
            if (bracket_stack.isEmpty())
                return false
            bracket_stack.pop()
        }
    }
    return bracket_stack.isEmpty()
}

//TASK 6
fun write_by_order(): String {
    val letters = MyStack<Char>()
    val digits = MyStack<Char>()
    val others = MyStack<Char>()
    val file = createFile("write_by_order")
    val string = file.bufferedReader().readLine()

    for (i in string) {
        when (true) {
            (i.isDigit()) -> digits.push(i)
            (i.isLetter()) -> letters.push(i)
            else -> others.push(i)
        }
```

```
            }
            var res = ""
            while (!digits.isEmpty())
                res += digits.pop()
            while (!letters.isEmpty())
                res += letters.pop()
            while (!others.isEmpty())
                res += others.pop()
            return res + "\n"
        }


//TASK 7

fun numbers_sort(): String {
    val numbers = listOf(-1, -2, 3, 5, -5, 6, -13, 44, 13, -100, 100)
    val deque = MyDeque<Int>()
    for (n in numbers)
        if (n < 0)
            deque.push(n)
        else
            deque.pushLast(n)

    while (!deque.isEmpty()) {
        val x = deque.pop()
        if (x!! < 0)
            deque.pushLast(x)
        else
            deque.push(x)
```

```
            break
        }
        var res = ""
        while (!deque.isEmpty()) {
            val x = deque.popLast()
            if (x!! < 0) {
                res += x
                res += "\n"
            } else
                deque.pushLast(x)
            break
        }
        while (!deque.isEmpty()) {
            res += deque.pop()
            res += "\n"
        }
        return res
    }

    //TASK 8
    fun reversText(): String {
        val file = File("Text.txt")
        val stack = MyStack<String>()
        if (!file.exists()) {
            file.createNewFile()
        }
        try {
            for (line in file.readLines()) {
```

```
            stack.push(line)
        }
    } catch (ex: IOException) {
        println(ex.message)
    }
    var res = ""
    while (!stack.isEmpty()) {
        res += stack.pop()
        res += "\n"
    }
    return res
}


//TASK 9

fun logicFun(): String {
    val file = createFile("logicFun")
    val text = file.bufferedReader().readLine()
    val opstack = MyStack<Char>()
    val vstack = MyStack<Char>()

    var cur = 0
    while (true) {
        var read = false
        if (!opstack.isEmpty()) {
            if (opstack.peek() == 'N') {
                if (vstack.isEmpty())
```

```
            read = true
        else {
            if (vstack.pop() == 'T')
                vstack.push('F')
            else
                vstack.push('T')
            opstack.pop()
        }
    } else if (opstack.peek() == 'A') {
        if (vstack.size() < 2)
            read = true
        else {
            val a = vstack.pop()
            val b = vstack.pop();
            if (a == b && b == 'T')
                vstack.push('T')
            else
                vstack.push('F')
            opstack.pop()
        }
    } else if (opstack.peek() == 'O')
        if (vstack.size() < 2)
            read = true
        else {
            val a = vstack.pop()
            val b = vstack.pop();
            if (a == 'T' || b == 'T')
                vstack.push('T')
```

```
            else
                vstack.pop()
            opstack.pop()
        }
    else if (opstack.peek() == 'X')
        if (vstack.size() < 2)
            read = true
        else {
            val a = vstack.pop()
            val b = vstack.pop()
            if (a != b)
                vstack.push('T')
            else
                vstack.push('F')
            opstack.pop()
        }
    else if (opstack.peek() == '(')
        read = true
    else if (opstack.peek() == ')') {
        opstack.pop()
        opstack.pop()
    }
} else
    read = true
if (read) {
    val i = text[cur]
    if (i in "FT")
        vstack.push(i)
```

```
            if (i in "AXON()")
                opstack.push(i)
            cur += 1
        }

        if (cur == text.length && opstack.size() == 0)
            break
    }
    var res = ""
    while (!vstack.isEmpty()) {
        res += vstack.pop()
        res += "\n"
    }
    return res
}

//TASK 10

fun maxFunction(): String {

    val file = createFile("maxFunction")
    val text = file.bufferedReader().readLine()

    val op = MyStack<Char>()
    val nums = MyStack<Int>()
    var num = "";
    var cur = 0;
    while (cur < text.length) {
```

```
    val i = text[cur]
    if (i.isDigit()) {
        num += i
    } else if (num != "") {
        nums.push(Integer.parseInt(num))
        num = ""
    }

    if (i == ')') {
        var a = nums.pop()
        var b = nums.pop()
        if (a!! < b!!) {
            val temp = a
            a = b
            b = temp
        }
        if (op.pop() == 'M')
            nums.push(a)
        else
            nums.push(b)
    }


    if (i in "MN")
        op.push(i)
    cur += 1

}
```

```
    while (!op.isEmpty()) {
        var a = nums.pop()
        var b = nums.pop()
        if (a!! < b!!) {
            val temp = a
            a = b
            b = temp
        }
        if (op.pop() == 'M')
            nums.push(a)
        else
            nums.push(b)
    }
    var res = ""
    while (!nums.isEmpty()) {
        res += nums.pop()
        res += "\n"
    }
    return res
}

fun funCulculate(): Boolean {
    val file = createFile("funCulculate")
    val text = file.bufferedReader().readLine()

    val stack = MyStack<Char>()
    var cur = 0
    while (true) {
```

```
var read = false
if (!stack.isEmpty()) {
    if (stack.peek() == '('
        read = true
    else if (stack.peek() == ')') {
        stack.pop()
        if (stack.size() < 2 || stack.pop() != 'F' || stack.pop() != '('
            return false
        stack.push('F')
    } else if (stack.peek() == 'F') {
        stack.pop()
        if (stack.size() > 1 && (stack.peek()!! in "+-"))
            if (stack.pop()!! in "+-" && stack.pop() == 'F')
                stack.push('F')
            else
                return false
        else {
            stack.push('F')
            read = true
        }

    } else
        read = true
} else
    read = true
if (read) {
    val i = text[cur]
    if (i in "xyz")
```

```
                stack.push('F')
            if (i in "()+-")
                stack.push(i)
            cur += 1
        }
        if (cur == text.length && stack.size() == 1)
            break



    }
    return true
}
```

Вывод: В результате лабораторной работы была создана реализация стека и дека и решены 11 задач с использованием этих структур данных.