Федеральное государственное автономное образовательное
учреждение высшего образования
Университет ИТМО

Факультет инфокоммуникационных технологий

Алагритмы и структуры данных:

Отчёт по лабораторной работе №3: Подстроки

Выполнил:
Бочкарь Артём Артёмович

Группа: **K33392**
Вариант: **13**

Преподаватели:
Артамонова В. Е.

Санкт-Петербург 2024 г.

Введение:
Программы реализовывал на языке Swift. Ввод и вывод реализовал через консоль по причине того, что использовал web-версию среды разработки Swift: среда.
Тесты реализовал с помощью следующей функции:

```
import Foundation

// Функция для измерения времени выполнения кода
func measureTime(block: () -> Void) -> TimeInterval {
    let startTime = Date()
    block()
    let endTime = Date()
    return endTime.timeIntervalSince(startTime)
}

// Пример использования
let timeTaken = measureTime {
    // Код, время выполнения которого нужно измерить
}

print("Время выполнения: \(timeTaken) секунд")
```

Задача №1: Наивный поиск подстроки в строке (1 балл)

Даны строки p и t. Требуется найти все вхождения строки p в строку t в качестве подстроки.

Формат ввода / входного файла (input.txt).
Первая строка входного файла содержит p, вторая – t. Строки состоят из букв латинского алфавита.

Формат вывода / выходного файла (output.txt).
В первой строке выведите число вхождений строки p в строку t. Во второй строке выведите в возрастающем порядке номера символов строки t, с которых начинаются вхождения p. Символы нумеруются с единицы.

Код приложения:

```
import Foundation

// Ввод данных через консоль
print("Введите строку p:")
if let p = readLine() {
    print("Введите строку t:")
    if let t = readLine() {
        let startTime = Date()
        var occurrences: [Int] = []

        // Поиск всех вхождений строки p в строку t
        let tLength = t.count
        let pLength = p.count
        for i in 0...(tLength - pLength) {
            let range = t.index(t.startIndex, offsetBy: i)..<t.index(t.startIndex, offsetBy: i + pLength)
```

```
            if t[range] == p {
                occurrences.append(i + 1)
            }
        }

        let endTime = Date()
        let timeElapsed = endTime.timeIntervalSince(startTime)

        // Вывод результатов
        print("")
        print(occurrences.count)
        print(occurrences.map { String($0) }.joined(separator: " "))
        print("Время выполнения: \(timeElapsed) секунд")
    }
}
```

Результат работы программы:

```
swift /tmp/qi61qiDVHz.swift
Введите строку p:
aba
Введите строку t:
abaCaba

2
1 5
Время выполнения: 9.894371032714844e-05 секунд
```

Задача №3: Паттерн в тексте (1 балл)

В этой задаче ваша цель – реализовать алгоритм Рабина-Карпа для поиска заданного шаблона (паттерна) в заданном тексте.

Формат ввода / входного файла (input.txt).
На входе две строки: паттерн P и текст T. Требуется найти все вхождения строки P в строку T в качестве подстроки.

Формат вывода / выходного файла (output.txt).
В первой строке выведите число вхождений строки P в строку T. Во второй строке выведите в возрастающем порядке номера символов строки T, с которых начинаются вхождения P. Символы нумеруются с единицы.

В первом примере паттерн aba можно найти в позициях 1 (abacaba) и 5 (abacaba) текста abacaba.
Паттерн и текст в этой задаче чувствительны к регистру. Поэтому во втором примере паттерн Test встречается только в 45 позиции в тексте testTesttesT.

Обратите внимание, что вхождения шаблона в тексте могут перекрываться, и это нормально, вам все равно нужно вывести их все.

Код приложения:

```
import Foundation

func rabinKarp(pattern: String, text: String) {
    let prime: UInt64 = 101 // Простое число для вычислений
    let patternLength = pattern.count
    let textLength = text.count
    var results: [Int] = []

    // Функция хеширования строки
    func hash(_ str: String) -> UInt64 {
        var hashValue: UInt64 = 0
        for char in str.unicodeScalars {
            hashValue = (hashValue * prime + UInt64(char.value))
        }
        return hashValue
    }

    let patternHash = hash(pattern)
    var textHash = hash(String(text.prefix(patternLength)))

    // Пересчитываем хеш и сравниваем подстроки
    for i in 0...(textLength - patternLength) {
        if patternHash == textHash, pattern == String(text[text.index(text.startIndex, offsetBy:
i)..<text.index(text.startIndex, offsetBy: i + patternLength)]) {
            results.append(i + 1)
        }
        if i < textLength - patternLength {
            textHash = textHash -
UInt64(text.unicodeScalars[text.unicodeScalars.index(text.unicodeScalars.startIndex, offsetBy: i)].value)
            textHash = textHash/prime
            textHash += UInt64(text.unicodeScalars[text.unicodeScalars.index(text.unicodeScalars.startIndex,
offsetBy: i + patternLength)].value) * UInt64(pow(Double(prime), Double(patternLength - 1)))
        }
    }

    print(results.count)
    for result in results {
        print(result, terminator: " ")
    }
    print("")
}

// Введите паттерн и текст через консоль
if let pattern = readLine(), let text = readLine() {
    let startTime = Date() // Засекаем начало выполнения программы
    print("")
    rabinKarp(pattern: pattern, text: text)
    let endTime = Date() // Засекаем конец выполнения программы
    let executionTime = endTime.timeIntervalSince(startTime) // Вычисляем время выполнения
    print("Время работы программы: \(executionTime) секунд")
}
```

Результат работы программы 1:

```
swift /tmp/fvVM5RoEmg.swift
aba
abacaba

2
1 5
Время работы программы: 0.00047099590301513 67 секунд
```

Результат работы программы 2:

```
swift /tmp/4SwiWa9iZt.swift
Test
testTesttesT

1
5
Время работы программы: 0.000007364926 секунд
```

Результат работы программы 3:

```
swift /tmp/vMTLo2rg0C.swift
aaaaa
baaaaaaa

3
2 3 4
Время работы программы: 0.000001467289 секунд
```

Задача №5: Префикс-функция (1.5 балла)

Постройте префикс-функцию для всех непустых префиксов заданной строки s.

Формат ввода / входного файла (input.txt).
Одна строка входного файла содержит s. Строка состоит из букв латинского алфавита.

Формат вывода / выходного файла (output.txt).
Выведите значения префикс-функции для всех префиксов строки s длиной 1, 2, ..., |s|, в указанном порядке.

Код программы:

```
import Foundation

func prefixFunction(_ input: String) -> [Int] {
    var result = [Int](repeating: 0, count: input.count)
    let s = Array(input)

    var i = 1
    var j = 0

    while i < s.count {
        if s[i] == s[j] {
            j += 1
            result[i] = j
            i += 1
        } else {
            if j != 0 {
                j = result[j - 1]
            } else {
                result[i] = 0
                i += 1
            }
        }
    }

    return result
}

func main() {
    print("Введите строку:")
    if let input = readLine() {
        // Засекаем время начала
        let startTime = Date()

        let pf = prefixFunction(input)
        print("\nВывод:")
        for value in pf {
            print("\(value)", terminator: " ")
        }

        // Рассчитываем прошедшее время и выводим его в секундах
        let endTime = Date()
        let elapsedTime = endTime.timeIntervalSince(startTime)
        print("\nВремя выполнения: \(elapsedTime) секунд")
    }
}

// Вызов основной функции
main()
```

Результат работы программы 1:

```
swift /tmp/pKyj8XxJOA.swift
Введите строку:
aaaAAA

Вывод:
0 1 2 0 0 0
Время выполнения: 0.00024199485778808594 секунд
```

Результат работы программы 2:

```
swift /tmp/4hLfu1vdD2.swift
Введите строку:
abacaba

Вывод:
0 0 1 0 1 2 3
Время выполнения: 0.0002269744873046875 секунд
```

Задача №6: Z-функция (1.5 балла)

Постройте Z-функцию для заданной строки s.

Формат ввода / входного файла (input.txt).
Одна строка входного файла содержит s. Строка состоит из букв латинского алфавита.

Формат вывода / выходного файла (output.txt).
Выведите значения Z-функции для всех индексов 1, 2, ..., |s| строки s, в указанном порядке.

Код программы:

```
import Foundation

func calculateZArray(_ string: String) -> [Int] {
    let n = string.count
    var z = [Int](repeating: 0, count: n)
    var l = 0, r = 0

    for i in 1..<n {
        if i <= r {
            z[i] = max(0, min(r - i + 1, z[i - l]))
```

```
        }
        while i + z[i] < n, string[string.index(string.startIndex, offsetBy: z[i])] ==
string[string.index(string.startIndex, offsetBy: i + z[i])] {
            z[i] += 1
        }
        if i + z[i] - 1 > r {
            l = i
            r = i + z[i] - 1
        }
    }
    return z
}

func main() {
    if let input = readLine() {
        let zValues = calculateZArray(input)
        print("")
        print(zValues.map{ String($0) }.joined(separator: " "))
    }
}

let startDate = Date()
main()
let endDate = Date()
let executionTime = endDate.timeIntervalSince(startDate)
print("Время работы алгоритма \(executionTime) секунд")
```

Результат работы программы 1:

```
swift /tmp/UVoOwr6CbU.swift
aaaAAA

0 2 1 0 0 0
Время работы алгоритма 8.604825973510742 секунд
```

Результат работы программы 2:

```
swift /tmp/aiI6qBuhRY.swift
abacaba

0 0 1 0 3 0 1
Время работы алгоритма 3.7407350540161133 секунд
```

Задача №7: Наибольшая общая подстрока (2 балла)

В задаче на наибольшую общую подстроку даются две строки s и t, и цель состоит в

том, чтобы найти строку w максимальной длины, которая является подстрокой как s, так и t. Это естественная мера сходства между двумя строками. Задача имеет применения для сравнения и сжатия текстов, а также в биоинформатике. Эту проблему можно рассматривать как частный случай проблемы расстояния редактирования (Левенштейна), где разрешены только вставки и удаления. Следовательно, ее можно решить за время O(|s||t|) с помощью динамического программирования.
Есть также весьма нетривиальные структуры данных для решения этой задачи за линейное время O(|s| + |t|). В этой
задаче ваша цель – использовать хеширование для решения почти за линейное время.

Формат ввода / входного файла (input.txt).
Каждая строка входных данных содержит две строки s и t, состоящие из строчных латинских букв.

Формат вывода / выходного файла (output.txt).
Для каждой пары строк si и ti найдите ее самую длинную общую подстроку и уточните ее параметры, выведя три целых числа: ее начальную позицию в s, ее начальную позицию в t (обе считаются с 0) и ее длину. Формально выведите целые числа $0 \le i < |s|$, $0 \le j < |t|$ и $l \ge 0$ такие, что и l максимально. (Как обычно, если таких троек с максимальным l много, выведите любую из них.)

Код программы:

```
import Foundation

// Function to the longest common substring using binary search
func longestCommonSubstring(_ s: String, _ t: String) -> (Int, Int, Int) {
    var maxLength = 0
    var startS = 0
    var startT = 0

    // Function to calculate the hash of a string substring
    func hash(str: String, start: Int, length: Int) -> Int {
        var hashValue = 0
        for i in start..<(start + length) {
            let char = str[str.index(str.startIndex, offsetBy: i)]
            hashValue = hashValue * 31 + Int(char.asciiValue ?? 0)
        }
        return hashValue
    }

    // Binary search to find the longest common substring
    func binarySearch(s: String, t: String, len: Int) -> (Int, Int) {
        var hashTable = Set<Int>()
        let base = 31
        var power = 1
        for _ in 0..<len - 1 {
            power *= base
        }

        for i in 0...(s.count - len) {
            let h = hash(str: s, start: i, length: len)
            hashTable.insert(h)
        }
```

```
        for j in 0...(t.count - len) {
            let h = hash(str: t, start: j, length: len)
            if hashTable.contains(h) {
                return (j, h)
            }
        }
        return (-1, -1)
    }

    var left = 1
    var right = min(s.count, t.count) + 1

    while left < right {
        let mid = left + (right - left) / 2
        let res = binarySearch(s: s, t: t, len: mid)
        if res.0 != -1 {
            maxLength = mid
            startS = res.1
            startT = res.0
            left = mid + 1
        } else {
            right = mid
        }
    }

    return (startS, startT, maxLength)
}

// Reading input from console
if let input = readLine() {
    let inputs = input.split(separator: " ")
    let s = String(inputs[0])
    let t = String(inputs[1])

    // Measure the execution time
    let start = DispatchTime.now()

    // Finding the longest common substring
    let result = longestCommonSubstring(s, t)

    // Calculating the elapsed time
    let end = DispatchTime.now()
    let nanoTime = end.uptimeNanoseconds - start.uptimeNanoseconds
    let timeInterval = Double(nanoTime) / 1_000_000_000

    // Output the result along with the elapsed time
    print("")
    print("\(result.0) \(result.1) \(result.2) - Время работы алгоритма: \(timeInterval) секунд")
}
```

Результат работы программы 1:

```
swift /tmp/bnDGrmBjn4.swift
cool toolbox

1 1 3 - Время работы алгоритма: 0.00018787 секунд
```

Результат работы программы 2:

```
swift /tmp/Appev2rXM4.swift
aaa bb

0 1 0 - Время работы алгоритма: 0.000193211 секунд
```

Результат работы программы 3:

```
swift /tmp/ClJ9lPA79i.swift
aabaa babbaab

0 4 3 - Время работы алгоритма: 0.00025056 секунд
```

Задача №9: Декомпозиция строки (2 балла)

Строка |ABCABCDEDEDEF| содержит подстроку |ABC| , повторяющуюся два раза подряд, и подстроку |DE| , повторяющуюся три раза подряд. Таким образом, ее можно записать как |ABC*2+DE*3+F|, что занимает меньше места, чем исходная запись той же строки.
Ваша задача – построить наиболее экономное представление данной строки s в виде, продемонстрированном выше, а именно, подобрать такие s1, a1, ..., sk, ak, где si - строки, а ai - числа, чтобы s = s1 · a1 + ... + sk · ak. Под операцией умножения строки на целое положительное число подразумевается конкатенация одной или нескольких копий строки, число которых равно числовому множителю, то есть, |ABC*2=ABCABC|. При этом требуется минимизировать общую длину итогового описания, в котором компоненты разделяются знаком | + | , а умножение строки на число записывается как умножаемая строка и множитель, разделенные знаком | * | . Если же множитель равен единице, его, вместе со знаком | * | , допускается не указывать.

Формат ввода / входного файла (input.txt).
Одна строка входного файла содержит s. Строка состоит из букв латинского алфавита.

Формат вывода / выходного файла (output.txt).
Выведите оптимальное представление строки, данной во входном файле. Если оптимальных представлений несколько, выведите любое.

Код программы:

```swift
import Foundation

func findOptimalRepresentation(inputString: String) -> String {
    var compressedString = ""
    var currentPosition = inputString.startIndex

    while currentPosition < inputString.endIndex {
        let currentChar = inputString[currentPosition]
        var substring = String(currentChar)
 var count = 1
        var tempPosition = inputString.index(after: currentPosition)

        while tempPosition < inputString.endIndex && inputString[tempPosition] == currentChar {
            substring += String(currentChar)
            count += 1
            tempPosition = inputString.index(after: tempPosition)
        }

        currentPosition = tempPosition

        if count > 1 {
            compressedString += "\(substring)*\(count)+"
        } else {
            compressedString += "\(currentChar)+"
        }
    }

    // Remove the extra '+' at the end of the string
    compressedString.removeLast()

    return compressedString
}

// Timer to measure program execution time
let startTime = Date()

// User input via console
print("")
if let inputString = readLine() {
    let optimalRepresentation = findOptimalRepresentation(inputString: inputString)
    print("")
    print("Оптимальное представление: \(optimalRepresentation)")

    // Calculate and print program execution time
    let endTime = Date()
    let timeInterval = endTime.timeIntervalSince(startTime)
    print("Время работы программы: \(timeInterval) секунд")
}
```

Результат работы программы 1:

```
swift /tmp/ruRpTPY6Mn.swift
Введите строку:
ABCABCDEDEDEF

Оптимальное представление: ABC*2+DE*3+FF
Время работы программы: 7.336874008178711 секунд
```

Результат работы программы 2:

```
swift /tmp/ruRpTPY6Mn.swift
Введите строку:
Hello

Оптимальное представление: Hello
Время работы программы: 3.7785250971537247 секунд
```

Вывод:

Решил задачи: 1(1б), 3(1б), 5(1.5б), 6(1.5б), 7(2б), 9(2б), в сумме 9 баллов.