**Прізвище:** Дуда

**Ім'я:** Костянтин

**Група:** КН-407

**Варіант:** 11

**Кафедра:** САПР

**Дисципліна:** Дискретні моделі в системному проектуванні

**Перевірив:** Кривий Р. З.

**Дата:**

**ЗВІТ**

до лабораторної роботи №4

на тему "Потокові алгоритми"

**Мета роботи:** Метою даної лабораторної роботи є вивчення потокових алгоритмів.

**Короткі теоретичні відомості:**

Останнім часом значно зросла зацікавленість учених та практиківпотоковими моделями. Це пов’язано із впровадженням та активним розвиткомрізноманітних територіально розподілених систем: трубопровідних,транспортних, телекомунікаційних та ін. Основою таких систем є певна мережа(мережа трубопроводів, доріг, каналів зв’язку тощо), в якій циркулюють певніпотоки (потоки речовин, транспорту, даних тощо), тому задачі, які доводитьсярозв’язувати при проектуванні та експлуатації систем з мережною структурою,часто зводяться до розробки математичних моделей розподілу потоків тапостановки і розв’язання відповідних оптимізаційних задач.Відомі моделі розподілу потоків у мережах базуються на поняттяхтеорії графів. Це пов’язано з тим, що граф дає можливість наочно відобразитиструктуру мережі, а параметри його вузлів і дуг – представити основнимичисловими характеристиками її елементів. Набір характеристик залежить відприроди модельованої системи, а також характеру розв’язуваних задач, однак употокових моделях їх, як правило, представляють такими параметрами, якзовнішній потік у вузлі, потік по дузі, пропускна здатність дуги, вартістьпередавання одиниці потоку по дузі тощо. Потокові задачі, як правило,зводяться до пошуку такого розподілу потоків у мережі, при якому бзабезпечувався екстремум деякого критерію. При цьому мають враховуватисяобмеження, що накладаються умовами збереження потоків у вузлах інеперевищення потоками пропускної здатності дуг. Типовими потоковимизадачами є задача про потік мінімальної вартості, про максимальний потік,транспортна задача, задача про призначення та інші. Для їх розв’язаннярозроблено чимало ефективних алгоритмів, сформувався навіть відповіднийнапрям обчислювальних методів під назвою потокового.

**Алгоритм пошуку максимального потоку.**

Основна задача алгоритму пошуку максимального потоку полягає в пошуку способів пересилання максимальної кількості одиниць потоку з витоку в стік при умові відсутності перевищення пропускних здатностей дуг вихідного графа. В основі алгоритму пошуку максимального потоку лежить наступна ідея: вибираємо початковий потік з витоку “s” в стік “t”, потім використовуємо алгоритм пошуку збільшуючого ланцюга. Цей алгоритм дозволяє знайти єдиний збільшуючий ланцюг з “s” в “t”, якщо той існує.

**Код програми:**

<https://github.com/Kruvyi/Lab4_DS>

package com.kostya.streams\_lab3

import scala.collection.mutable.ListBuffer

import scala.io.Source

object StreamWork {

val T = 7

val N = 6

val Sr: Array[KeyPoint] = new Array[KeyPoint](N)

var maxTer: Int = 0

private val fullPath = "C:\\Users\\kostya\_duda\\Desktop\\graphs\\src\\main\\scala\\"

def fillThread(): Seq[Stream] = {

readFromFile("streams.txt").toSeq

}

def readFromFile(filePath: String): ListBuffer[Stream] = {

val streams = new ListBuffer[Stream]

val bufferedSource = Source.fromFile(fullPath + filePath)

for (line <- bufferedSource.getLines) {

val edgeParams = line.split(" ")

streams += Stream(edgeParams(0).toInt, edgeParams(1).toInt, edgeParams(2).toInt)

}

bufferedSource.close

println(streams)

streams

}

def main(args: Array[String]): Unit = {

val thread = fillThread()

for (i <- 0 until N) {

Sr(i) = KeyPoint(i, 0, 0)

}

for (i <- 0 until N)

for (j <- 0 until T) {

if (Sr(i).name == thread(j).a)

Sr(i).output += thread(j).c

if (Sr(i).name == thread(j).b)

Sr(i).input += thread(j).c

}

Sr(N - 2).input = Sr(N - 2).output

Sr(N - 1).output = Sr(N - 1).input

for (i <- 0 until N)

Sr(i).capacity = Math.max(Sr(i).input, Sr(i).output)

maxTer = Math.max(Sr(N - 1).capacity, Sr(N - 2).capacity)

for (j <- 0 until N)

if (thread(j).b == N - 1)

Sr(N - 1).capacity -= Math.min(thread(j).c, Sr(thread(j).a).capacity)

for (i <- N - 3 until(0, -1))

for (j <- 0 until T)

if (thread(j).b == i)

Sr(i).capacity -= Math.min(thread(j).c, Sr(thread(j).a).capacity)

println("Thread max capacity: " + maxTer)

}

case class Stream(a: Int, b: Int, c: Int)

case class KeyPoint(name: Int, var input: Int, var output: Int) {

var capacity: Int = 0

}

case class Transformer() {

def numToUpperLetter(num: Int) : String = {

(num + 65).toString

}

def upperLetterToNum(s: Char): Int = s.toInt - 65

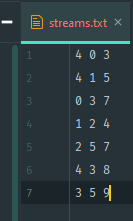
}

}

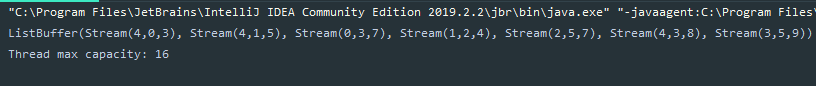
**Інструкція користувача:**

Для роботи програмі потрібно текстовий файл з списком ребер графа. Кожне ребро записується з нового рядка, через пробіл записується перша вершина ребра, друга вершина і пропускна здатність дуги. Коли вхідний файл буде готовий потрібно запустити програму, після того як програма завершить свою роботу на екран буде виведено заданий список ребер, список потоків, а також загальна величина потоку графа.

**Вхідний файл:**



**Результати:**



**Висновок:** у цій лабораторній роботі, я вивчив потокові алгоритми. На практиці закріпив отримані знання. Використовуючи об'єктно-орієнтовану мову програмування Scala, реалізував алгоритм пошуку максимального потоку.