**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 №3

на тему "Алгоритм рішення задачі комівояжера"

**Мета роботи:** Метою даної лабораторної роботи є вивчення і дослідження алгоритмів рішення задачі комівояжера.

**Короткі теоретичні відомості:**

Невідомо, коли проблему комівояжера було досліджено вперше. Однак, відома видана в 1832 році книжка з назвою «Комівояжер — як він має поводитись і що має робити для того, аби доставляти товар та мати успіх в своїх справах — поради старого Кур'єра», в якій описано проблему, але математичний апарат для її розв'язання не застосовується. Натомість, в ній запропоновано приклади маршрутів для деяких регіонів Німеччини та Швейцарії. Раннім варіантом задачі може розглядатись англ. Icosian Game Вільяма Гамільтона 19 століття, яка полягала в тому, щоб знайти маршрути на графі з 20 вузлами. Перші згадки в якості математичної задачі на оптимізацію нелажать Карлу Менґеру (нім. Karl Menger), який сформулював її в математичному колоквіумі в 1930 році наступним чином. Ми називаємо проблемою женця (оскільки це питання виникає в кожного листоноші, зокрема, її вирішують багато мандрівників) завдання віднайти найкоротший шлях між скінченною множиною місць, відстань між якими відома. Невдовзі з'явилась відома зараз назва задача мандруючого продавця (англ. Traveling Salesman Problem), яку запропонував Гаслер Вітні (англ. Hassler Whitney) з Принстонського Університету. Рішенням задачі комівояжера є оптимальний гамільтоновий контур. На жаль, не всі графи містять гамільтоновий контур. Отже перед тим, ніж перейти до пошуку оптимального гамільтонового контура потрібно довести факт його існування в даному графі. Введемо ряд визначень, які в подальшому нам знадобляться. Граф називається сильно зв`язаним, якщо в ньому для будь-яких двох вершин “х” і “у” існує шлях від “х” до “у”. Підмножина вершин Х деякого графа називається сильно зв`язаною, якщо для будь-яких пар вершин х і у існує шлях з “х” в “у” і Х не є підмножиною ніякої іншої множини вершин, які володіють тими ж властивостями.

**Код програми:**

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

package com.kostya.salesman\_lab4

import com.kostya.{Edge, Graph}

import scala.collection.mutable.ListBuffer

import scala.util.Random

class Salesman(graph: Graph) {

var cycle = new ListBuffer[Edge]

private val edges = graph.edges.clone()

val vertices = graph.getVertices

def gamiltonCycle(): Unit = {

val nextEdge = graph.pop

println(vertices)

cycle += nextEdge

vertices -= nextEdge.vertex1

vertices -= nextEdge.vertex2

while (vertices.nonEmpty) {

val edgesCopy = edges.clone()

addNextEdge(edgesCopy)

}

addLastEdge()

println(cycle + " length: " + getCycleLength(cycle))

var swappedCycle = swap()

for (i <- 0 to 100) {

println(swappedCycle + " length: " + getCycleLength(swappedCycle))

if (getCycleLength(swappedCycle) < getCycleLength(cycle)) cycle = swappedCycle

swappedCycle = swap()

}

println("Final cycle: " + cycle + " length: " + getCycleLength(cycle))

}

def addNextEdge(edgesCopy: ListBuffer[Edge]): Unit = {

if (edgesCopy.nonEmpty) {

if (!edgesCopy.head.isPresent(vertices) && areConnected(cycle.last, edgesCopy.head)) {

addToCycle(edgesCopy, edgesCopy.head)

if (edgesCopy.nonEmpty)

addNextEdge(edgesCopy.tail)

} else addNextEdge(edgesCopy.tail)

}

}

def swap(): ListBuffer[Edge] = {

val swappedCycle = new ListBuffer[Edge]

graph.getVertices

var randomVertice = ""

var swapped = " "

do {

randomVertice = graph.getVertices.toVector(new Random().nextInt(graph.getVertices.size))

swapped = graph.getVertices.toVector(new Random().nextInt(graph.getVertices.size))

} while (randomVertice == swapped)

println("vertices to swap: " + randomVertice + " " + swapped)

var edgeWage = 0

for (edge <- cycle) {

if (sameVertices(edge, randomVertice, swapped)) {

swappedCycle += Edge(edge.vertex2, edge.vertex1, edge.wage)

} else if (edge.vertex1 == randomVertice) {

edgeWage = getEdgeWage(swapped, edge.vertex2)

swappedCycle += Edge(swapped, edge.vertex2, edgeWage)

}

else if (edge.vertex2 == randomVertice) {

edgeWage = getEdgeWage(edge.vertex1, swapped)

swappedCycle += Edge(edge.vertex1, swapped, edgeWage)

} else if (edge.vertex1 == swapped) {

edgeWage = getEdgeWage(randomVertice, edge.vertex2)

swappedCycle += Edge(randomVertice, edge.vertex2, edgeWage)

} else if (edge.vertex2 == swapped) {

edgeWage = getEdgeWage(edge.vertex1, randomVertice)

swappedCycle += Edge(edge.vertex1, randomVertice, edgeWage)

} else swappedCycle += edge

}

swappedCycle

}

def getEdgeWage(vertex1: String, vertex2: String): Int = {

edges.find(e => (e.vertex1 == vertex1 && e.vertex2 == vertex2)

|| (e.vertex1 == vertex2 && e.vertex2 == vertex1)).get.wage

}

def sameVertices(edge: Edge, vertex1: String, vertex2: String): Boolean = {

(edge.vertex1 == vertex1 && edge.vertex2 == vertex2) ||

(edge.vertex1 == vertex2 && edge.vertex2 == vertex1)

}

def addLastEdge(): Unit = {

for (edge <- edges) {

if (cycle.last.vertex2 == edge.vertex1) {

cycle += edge

return

}

}

}

def getCycleLength(cycler: ListBuffer[Edge]): Int = {

var num = 0

for (edge <- cycler) {

num += edge.wage

}

num

}

def addToCycle(edgesCopy: ListBuffer[Edge], nextEdge: Edge): Unit = {

cycle += nextEdge

edgesCopy -= nextEdge

vertices -= nextEdge.vertex1

vertices -= nextEdge.vertex2

}

def areConnected(edge1: Edge, edge2: Edge): Boolean = {

if (edge1.vertex2 == edge2.vertex1) true

else false

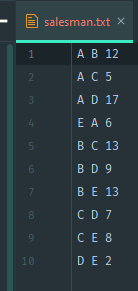
}

}

**Інструкція користувача:**

Для роботи програмі потрібно текстовий файл з кількістю вершин в графі і 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 матриця ваг для заданого графа і рішення задачі комівояжера.

**Вхідний файл:**



**Результати:**



**Висновок:** у цій лабораторній роботі, я вивчив і дослідив алгоритми рішення задачі комівояжера. На практиці закріпив отримані знання. Використовуючи об'єктно-орієнтовану мову програмування Scala, реалізував рішення задачі комівояжера.