**Прізвище:** Дуда

**Ім'я:** Констянтин

**Група:** КН-407

**Варіант:** 11

**Кафедра:** САПР

**Дисципліна:** Дискретні моделі в системному проектуванні

**Перевірив:** Кривий Р. З.

**Дата:**

**ЗВІТ**

до лабораторної роботи №1

на тему "Алгоритм побудови дерев"

**Мета роботи:** Метою даної лабораторної роботи є вивчення алгоритмів рішення задач побудови остових дерев.

**Короткі теоретичні відомості:**

Для того, щоб розглянути алгоритми побудови дерев нагадаємо деякі основні положення теорії графів. Графом G називають скінчену множину V з нерефлексивним симетричним відношенням R на V. Визначим E як множину симетричних пар в R. Кожний елемент V називають вершиною. Кожний елемент Е називають ребром, а E множиною ребер G.

Граф називається зв’язним, якщо в ньому для будь-якої пари вершин знайдеться ланцюг, який їх з’єднує, тобто, якщо по ребрах (дугах) можна попасти з будь-якої вершини в іншу.

Дерево - це зв’язний граф без циклів. Лісом називається будь-яка сукупність дуг (ребер) інцидентних до вершин, які не містять циклів. Таким чином, ліс складається з одного або більше дерев.

**Алгоритм Прима.**

Алгоритм Прима — жадібний алгоритм побудови мінімального кістякового дерева зваженого зв'язного неорієнтованого графа.

Побудова починається з дерева, що включає в себе одну (довільну) вершину. Протягом роботи алгоритму дерево розростається, поки не охопить всі вершини вихідного графа. На кожному кроці алгоритму до поточного дереву приєднується найлегше з ребер, що з'єднують вершину з побудованого дерева і вершину, що не належить дереву.

1. Спочатку ребра сортують за зростанням ваги.
2. Додають найменше ребро в дерево.
3. Зі списку ребер із найменшою вагою вибирають таке нове ребро, щоб одна з його вершин належала дереву, а інша — ні.
4. Це ребро додають у дерево і знову переходять до кроку 3.
5. Робота закінчується, коли всі вершини будуть у дереві.

https://github.com/Kruvyi/Lab1\_DS.git

**Код програми:**

package com.pavlo.prim\_lab1

import com.pavlo.{Edge, Graph}

import scala.collection.mutable.ListBuffer

class Prim(primGraph: Graph) {

private val vertices = primGraph.getVertices

def findPrim() : ListBuffer[Edge] = {

val tree = new Graph(new ListBuffer[Edge])

val sortedEdges = primGraph.EdgesSortedByWage()

val graph = new Graph(sortedEdges)

graph.print()

addToTree(tree, graph, graph.pop)

println()

graph.print()

while (vertices.nonEmpty) {

val nextEdge = findNextMin(graph.edges, tree.edges, tree.edges)

addToTree(tree, graph, nextEdge)

}

println("Tree")

tree.print()

tree.edges

}

def findNextMin(graph : ListBuffer[Edge], tree : ListBuffer[Edge], initTree: ListBuffer[Edge]) : Edge = {

if (tree.nonEmpty) {

if (graph.head.compareVertices(tree.head) && !graph.head.isPresent(vertices)) {

println(graph.head)

println(vertices)

graph.head

}

else findNextMin(graph, tree.tail, initTree)

} else if (graph.nonEmpty) {

findNextMin(graph.tail, initTree, initTree)

}

else Edge("", "", 0)

}

def addToTree(tree: Graph, graph: Graph, nextEdge: Edge): Unit = {

tree.edges += nextEdge

graph.edges -= nextEdge

vertices -= nextEdge.vertex1

vertices -= nextEdge.vertex2

}

}package com.kostya.prim\_lab1

import com.kostya.{Edge, Graph}

import scala.collection.mutable.ListBuffer

class Prim(primGraph: Graph) {

private val vertices = primGraph.getVertices

def findPrim() : ListBuffer[Edge] = {

val tree = new Graph(new ListBuffer[Edge])

val sortedEdges = primGraph.EdgesSortedByWage()

val graph = new Graph(sortedEdges)

graph.print()

addToTree(tree, graph, graph.pop)

println()

graph.print()

while (vertices.nonEmpty) {

val nextEdge = findNextMin(graph.edges, tree.edges, tree.edges)

addToTree(tree, graph, nextEdge)

}

println("Tree")

tree.print()

tree.edges

}

def findNextMin(graph : ListBuffer[Edge], tree : ListBuffer[Edge], initTree: ListBuffer[Edge]) : Edge = {

if (tree.nonEmpty) {

if (graph.head.compareVertices(tree.head) && !graph.head.isPresent(vertices)) {

println(graph.head)

println(vertices)

graph.head

}

else findNextMin(graph, tree.tail, initTree)

} else if (graph.nonEmpty) {

findNextMin(graph.tail, initTree, initTree)

}

else Edge("", "", 0)

}

def addToTree(tree: Graph, graph: Graph, nextEdge: Edge): Unit = {

tree.edges += nextEdge

graph.edges -= nextEdge

vertices -= nextEdge.vertex1

vertices -= nextEdge.vertex2

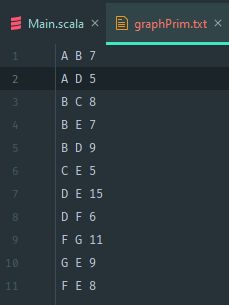
}

}

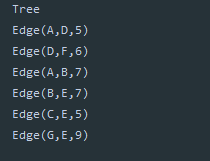
**Інструкція користувача:**

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 вага дуги. Коли вхідний файл буде готовий потрібно запустити програму, після того як програма завершить свою роботу на екран буде виведено заданий список ребер, список ребер остового дерева, а також вага остового дерева.

**Вхідний файл:**



**Результати:**



**Висновок:** у цій лабораторній роботі, я вивчив алгоритми рішення задач побудови остових дерев. На практиці закріпив отримані знання. Використовуючи об'єктно-орієнтовану мову програмування Scala, реалізував алгоритм Прима.