Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Лабораторна робота №2

Обробка надвеликих масивів даних

Тема: Робота з графовими структурами

Виконав
студент групи ІП-51мн:
Панченко С. В.

Перевірив:
Олійник Ю. О.

Київ 2025

ЗМІСТ

1 МЕТА

Відпрацювати роботу з графовими структурами.

2 ВИКОНАННЯ

2.1 Підготовка даних та середовища розробки

Підключимося до віддаленої машини.

```
gcloud auth login
gcloud compute ssh cluster-11a7-m
```

Для початку встановимо Scala.

```
curl -fL
https://github.com/coursier/coursier/releases/latest/download/cs-
x86_64-pc-linux.gz | gzip -d > cs && chmod +x cs && ./cs setup
```

Вийдемо та зайдемо з системи, щоб оновити PATH та отримати консольні команди, як-от: cs, sbt тощо.

```
exit
gcloud compute ssh cluster-11a7-m
```

Потім встановимо lsp-сервер для Scala та збілдимо його, встановивши 17-ту Java.

```
sudo apt install openjdk-17-jdk
```

Завантажуємо metals та білдимо його.

```
wget
https://github.com/scalameta/metals/archive/refs/tags/v1.6.2.zip
unzip v1.6.2.zip
cd metals-1.6.2/
sbt publishLocal --java-home /usr/lib/jvm/java-17-openjdk-amd64/
```

```
[warn] 141 deprecations
[warn] 1 deprecation (since 0.9.0)
[warn] 1 deprecation (since 1.2)
[warn] 1 deprecation (since 2.11.0)
[warn] 24 deprecations (since 2.13.0)
[warn] 24 deprecations (since 2.13.3)
[warn] 9 deprecations (since 2.13.7)
[warn] 1 deprecation (since >4.4.35)
[warn] 202 deprecations in total; re-run with -deprecation for details
[warn] 12 warnings found
[info] :: delivering :: org.scalameta#metals_2.13;1.6.2-SNAPSHOT :: 1.6.2-SNAPSHOT :: integration :: Sun Sep 28 10:28:22 UTC 2025
[info]  delivering ivy file to /home/user/metals-1.6.2/metals/target/scala-2.13/ivy-1.6.2-SNAPSHOT.xml
[info]  published metals_2.13 to /home/user/.ivy2/local/org.scalameta/metals_2.13/1.6.2-SNAPSHOT/poms/metals_2.13.pom
[info]  published metals_2.13 to /home/user/.ivy2/local/org.scalameta/metals_2.13/1.6.2-SNAPSHOT/jars/metals_2.13.jar
[info]  published ivy to /home/user/.ivy2/local/org.scalameta/metals_2.13/1.6.2-SNAPSHOT/ivys/ivy.xml
[success] Total time: 155 s (0:02:35.0), completed Sep 28, 2025, 10:28:22 AM
```

Рисунок 2.1: Встановлення metals

Далі додамо lsp до neovim у ~/.config/nvim/init.lua

```
vim.g.mapleader = " "
vim.g.maplocalleader = "\\"
vim.opt.nu = true
vim.opt.tabstop = 4
vim.opt.shiftwidth = 4
vim.opt.expandtab = true
local lazypath = vim.fn.stdpath("data") .. "/lazy/lazy.nvim"
if not (vim.uv or vim.loop).fs_stat(lazypath) then
    local lazyrepo = "https://github.com/folke/lazy.nvim.git"
    local out = vim.fn.system({
        "git", "clone", "--filter=blob:none", "--branch=stable",
lazyrepo, lazypath
    })
end
vim.opt.rtp:prepend(lazypath)
require('lazy').setup({
    {
        "catppuccin/nvim",
        name = "catppuccin",
        priority = 1000
    },
    {
        "nvim-tree/nvim-tree.lua",
        version = "*",
        lazy = false,
        dependencies = { "nvim-tree/nvim-web-devicons" },
        config = function()
            require("nvim-tree").setup {}
        end,
    },
    {
        'akinsho/bufferline.nvim',
        version = "*",
        dependencies = 'nvim-tree/nvim-web-devicons'
    },
```

```
    {
      'saghen/blink.cmp',
      -- optional: provides snippets for the snippet source
      dependencies = { 'rafamadriz/friendly-snippets' },

      -- use a release tag to download pre-built binaries
      version = '1.*',
      opts = {
        -- All presets have the following mappings:
        -- C-space: Open menu or open docs if already open
        -- C-n/C-p or Up/Down: Select next/previous item
        -- C-e: Hide menu
        -- C-k: Toggle signature help (if signature.enabled =
true)
        --
        -- See :h blink-cmp-config-keymap for defining your own
keymap
        keymap = { preset = 'default' },

        appearance = {
          nerd_font_variant = 'mono'
        },

        -- (Default) Only show the documentation popup when
manually triggered
        completion = {
            documentation = {
                auto_show = true,
                auto_show_delay_ms = 0
            }
        },
        signature = { enabled = true },
        -- Default list of enabled providers defined so that you
can extend it
        -- elsewhere in your config, without redefining it, due to
`opts_extend`
        sources = {
```

```
          default = { 'lsp', 'path', 'snippets', 'buffer' },
        },

        fuzzy = { implementation = "prefer_rust_with_warning" }
      },
      opts_extend = { "sources.default" }
    },
    {
        'windwp/nvim-autopairs',
        event = "InsertEnter", -- Only load the plugin when
entering insert mode
        config = function()
            require('nvim-autopairs').setup({
                -- You can customize options here
                -- For example, to disable it in certain
filetypes:
                -- disable_filetype = { "TelescopePrompt",
"vim" },
            })
        end,
    }
})
require("nvim-tree").setup {
  git = {
    enable = true,
    ignore = false,
  },
  filters = {
    dotfiles = false,
  },
  sync_root_with_cwd = true,
}
vim.cmd('colorscheme catppuccin-macchiato')
vim.keymap.set({"n", "v"}, "<leader>y", [["+y]])
vim.keymap.set("n", "<leader>w", ":NvimTreeFocus<cr>")
vim.keymap.set("n", "<leader>e", ":NvimTreeFindFileToggle<cr>")
vim.opt.termguicolors = true
```

```
require("bufferline").setup()

vim.opt.completeopt = { "menuone", "noselect", "popup" }
vim.lsp.config['metals'] = {
    cmd = {'/home/user/metals'}, -- Only the executable here
    filetypes = {'scala'},
    root_markers = { 'build.sbt' },
    cmd_env = {
        JAVA_HOME = '/usr/lib/jvm/java-17-openjdk-amd64',
    },
    settings = {
        javaHome = '/usr/lib/jvm/temurin-11-jdk-amd64'
    },
}
vim.lsp.enable('metals')
```

Створимо чистий проект з допомогою stb.

```
stb new
```

Додамо apache spark graphx в build.stb.

```
ThisBuild / scalaVersion := "2.12.18"
lazy val root = (project in file("."))
.settings(
name := "lab_2",
version := "0.1.0",
libraryDependencies ++= Seq(
"org.apache.spark" %% "spark-core" % "3.5.3" % "provided",
"org.apache.spark" %% "spark-sql" % "3.5.3" % "provided",
"org.apache.spark" %% "spark-graphx" % "3.5.3" % "provided"
)
)
```

bloop — це build server для Scala. Коли ми вказали apache spark як залежність, треба оновити конфігурацію bloop, щоб не підсвічувався import червоним.

```
sbt reload update bloopInstall
```

Запишемо hello-world в src/main/scala/example/Hello.scala.

```
package example

import org.apache.spark.graphx._

object Hello extends Greeting with App {
  println(greeting)
}

trait Greeting {
  lazy val greeting: String = "hello"
}
```

Скомпілюємо проект.

```
sbt compile
```

```
user@cluster-11a7-m:~/masters_first_semester_bigdata/lab_2$ sbt compile
[info] welcome to sbt 1.11.6 (Eclipse Adoptium Java 11.0.20.1)
[info] loading settings for project lab_2-build-build from metals.sbt...
[info] loading project definition from /home/user/masters_first_semester_bigdata/lab_2/project/project
[info] loading settings for project lab_2-build from metals.sbt...
[info] loading project definition from /home/user/masters_first_semester_bigdata/lab_2/project
[success] Generated .bloop/lab_2-build.json
[success] Total time: 4 s, completed Sep 28, 2025, 5:34:04 PM
[info] loading settings for project root from build.sbt...
[info] set current project to lab_2 (in build file:/home/user/masters_first_semester_bigdata/lab_2/)
[info] Executing in batch mode. For better performance use sbt's shell
[success] Total time: 2 s, completed Sep 28, 2025, 5:34:07 PM
user@cluster-11a7-m:~/masters_first_semester_bigdata/lab_2$
```

Рисунок 2.2 — Компіляція прикладу

Запустимо приклад.

```
sbt run
```

```
user@cluster-11a7-m:~/masters_first_semester_bigdata/lab_2$ sbt run
[info] welcome to sbt 1.11.6 (Eclipse Adoptium Java 11.0.20.1)
[info] loading settings for project lab_2-build-build from metals.sbt...
[info] loading project definition from /home/user/masters_first_semester_bigdata/lab_2/project/project
[info] loading settings for project lab_2-build from metals.sbt...
[info] loading project definition from /home/user/masters_first_semester_bigdata/lab_2/project
[success] Generated .bloop/lab_2-build.json
[success] Total time: 4 s, completed Sep 28, 2025, 5:35:44 PM
[info] loading settings for project root from build.sbt...
[info] set current project to lab_2 (in build file:/home/user/masters_first_semester_bigdata/lab_2/)
[info] running example.Hello
hello
[success] Total time: 2 s, completed Sep 28, 2025, 5:35:47 PM
user@cluster-11a7-m:~/masters_first_semester_bigdata/lab_2$
```

Рисунок 2.3 — Запуск прикладу

Додамо репозиторій sampledata як підмодуль.

```
git submodule add git@github.com:fcerbell/sampledata.git
sampledata
```

```
user@cluster-11a7-m:~/masters_first_semester_bigdata/lab_2$ git submodule add https://github.com/fcerbell/sampledata.git sampledata
Cloning into '/home/user/masters_first_semester_bigdata/lab_2/sampledata'...
remote: Enumerating objects: 31, done.
remote: Total 31 (delta 0), reused 0 (delta 0), pack-reused 31 (from 1)
Receiving objects: 100% (31/31), 1.16 MiB | 7.10 MiB/s, done.
Resolving deltas: 100% (8/8), done.
```

Рисунок 2.4 — Довання підмодуля

Тепер можна перейти до виконання завдання.

Рисунок 2.5 — Готове для роботи середовище розробки з LSP

Увімкнений LSP з JVM сильно навантажають машину, тому збільшимо оперативну пам’ять.

Рисунок 2.6 — Велике споживання оперативної пам'яті

Виділимо у загальному 16 ГБ опертивної пам’яті.

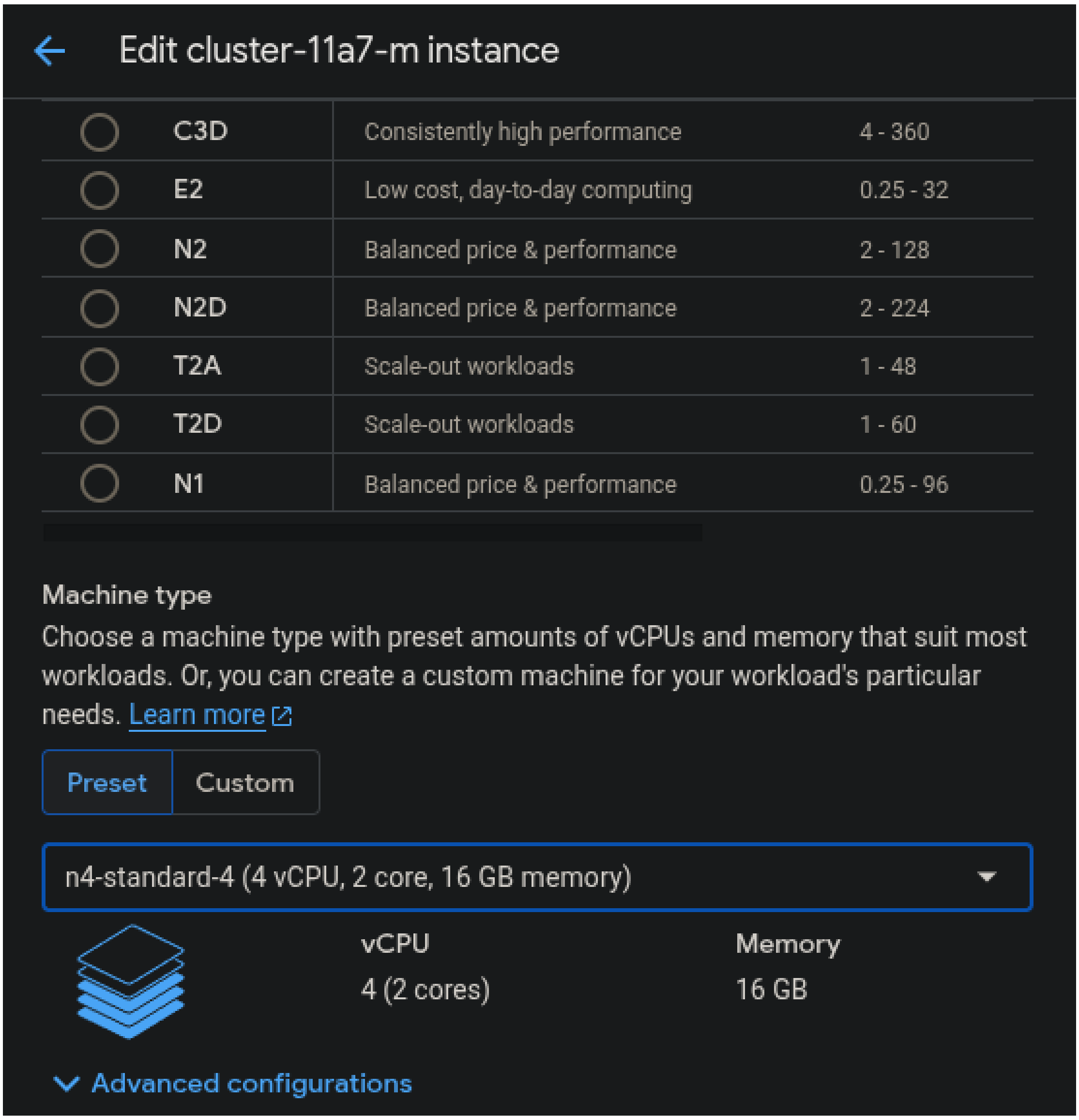

Рисунок 2.7 — Збільшення оперативної пам'яті

Знову викличемо htop.

Рисунок 2.8 — Майже 16 ГБ оперативної пам'яті

2.2 Завдання 1

Для початку завантажимо та відфільтруємо дані.

```
val sampledataOpenFlightsOrg = "sampledata/OpenFlights.org/";
```

```
val csvOpts = Map(
  "header" -> "false",
  "inferSchema" -> "false",
  "quote" -> "\"",
  "escape" -> "\"",
  "mode" -> "PERMISSIVE",
  "multiLine" -> "false",
  "ignoreLeadingWhiteSpace" -> "true",
  "ignoreTrailingWhiteSpace" -> "true"
);
sealed trait SqlField {
  def sqlType: spark.sql.types.DataType
  def name: String
  def index: Int
}
sealed trait AirportField extends SqlField
object AirportField {
  case object Id extends AirportField {
    val sqlType = spark.sql.types.LongType
    val name = "id"
    val index = 0
  }
  case object Name extends AirportField {
    val sqlType = spark.sql.types.StringType
    val name = "name"
    val index = 1
  }
  case object City extends AirportField {
    val sqlType = spark.sql.types.StringType
    val name = "city"
    val index = 2
  }
  case object Country extends AirportField {
    val sqlType = spark.sql.types.StringType
    val name = "country"
    val index = 3
  }
```

```
    case object IATA extends AirportField {
      val sqlType = spark.sql.types.StringType
      val name = "iata"
      val index = 4
    }
    case object ICAO extends AirportField {
      val sqlType = spark.sql.types.StringType
      val name = "icao"
      val index = 5
    }
    case object Latitude extends AirportField {
      val sqlType = spark.sql.types.DoubleType
      val name = "latitude"
      val index = 6
    }
    case object Longitude extends AirportField {
      val sqlType = spark.sql.types.DoubleType
      val name = "longitude"
      val index = 7
    }
    case object Altitude extends AirportField {
      val sqlType = spark.sql.types.DoubleType
      val name = "altitude"
      val index = 8
    }
    case object TimezoneOffset extends AirportField {
      val sqlType = spark.sql.types.IntegerType
      val name = "timezone_offset"
      val index = 9
    }
    case object DaylightSavingTime extends AirportField {
      val sqlType = spark.sql.types.StringType
      val name = "daylight_saving_time"
      val index = 10
    }
  }
  sealed trait AirlineField extends SqlField
```

```
object AirlineField {
  case object Id extends AirlineField {
    val sqlType = spark.sql.types.LongType
    val name = "id"
    val index = 0
  }
  case object Name extends AirlineField {
    val sqlType = spark.sql.types.StringType
    val name = "name"
    val index = 1
  }
  case object Alias extends AirlineField {
    val sqlType = spark.sql.types.StringType
    val name = "alias"
    val index = 2
  }
  case object IATA extends AirlineField {
    val sqlType = spark.sql.types.StringType
    val name = "iata"
    val index = 3
  }
  case object ICAO extends AirlineField {
    val sqlType = spark.sql.types.StringType
    val name = "icao"
    val index = 4
  }
  case object Callsign extends AirlineField {
    val sqlType = spark.sql.types.StringType
    val name = "callsign"
    val index = 5
  }
  case object Country extends AirlineField {
    val sqlType = spark.sql.types.StringType
    val name = "country"
    val index = 6
  }
  case object Active extends AirlineField {
```

```
      val sqlType = spark.sql.types.StringType
      val name = "active"
      val index = 7
    }
  }
  sealed trait RouteField extends SqlField
  object RouteField {
    case object Airline extends RouteField {
      val sqlType = spark.sql.types.StringType
      val name = "airline"
      val index = 0
    }
    case object AirlineId extends RouteField {
      val sqlType = spark.sql.types.LongType
      val name = "airline_id"
      val index = 1
    }
    case object SourceAirport extends RouteField {
      val sqlType = spark.sql.types.StringType
      val name = "src"
      val index = 2
    }
    case object SourceAirportId extends RouteField {
      val sqlType = spark.sql.types.LongType
      val name = "src_id"
      val index = 3
    }
    case object DestAirport extends RouteField {
      val sqlType = spark.sql.types.StringType
      val name = "dest"
      val index = 4
    }
    case object DestAirportId extends RouteField {
      val sqlType = spark.sql.types.LongType
      val name = "dest_id"
      val index = 5
    }
```

```
    case object Codeshare extends RouteField {
      val sqlType = spark.sql.types.StringType
      val name = "codeshare"
      val index = 6
    }
    case object Stops extends RouteField {
      val sqlType = spark.sql.types.IntegerType
      val name = "stops"
      val index = 7
    }
    case object Equipment extends RouteField {
      val sqlType = spark.sql.types.StringType
      val name = "equipment"
      val index = 8
    }
  }
  def readData(sparkSession: spark.sql.SparkSession, name: String,
fields: Vector[SqlField]) = {
    val columns = fields.map { field =>
      spark.sql.functions.col(s"_c${field.index}").as(field.name)
    }
    val baseDf =
sparkSession.read.options(csvOpts).csv(sampledataOpenFlightsOrg +
name)
      .select(columns: _*)
      .filter(columns.map { col => col.isNotNull }.reduce(_ && _))
      .filter(columns.map { col => !col.equalTo("\\N") }.reduce(_
&& _))

    fields.foldLeft(baseDf) { (df, field) =>
      df.withColumn(field.name,
spark.sql.functions.col(field.name).cast(field.sqlType))
    }
    .select(fields.map(f => spark.sql.functions.col(f.name)): _*)
  }
  def readAirports(sparkSession: spark.sql.SparkSession, fields:
Vector[AirportField]) = {
```

```
    readData(sparkSession, "airports-extended.dat", fields)
  }
  def readAirlines(sparkSession: spark.sql.SparkSession, fields:
Vector[AirlineField]) = {
    readData(sparkSession, "airlines.dat", fields)
  }
  def readRoutes(sparkSession: spark.sql.SparkSession, fields:
Vector[RouteField]) = {
    readData(sparkSession, "routes.dat", fields)
  }
```

Після цього перейдемо до побудови графа. Для розрахунку відстаней напишемо функцію haversineKm.

```
def haversineKm(lat1: Double, lon1: Double, lat2: Double, lon2:
Double): Double = {
    val R = 6371.0088
    val dLat = Math.toRadians(lat2 - lat1)
    val dLon = Math.toRadians(lon2 - lon1)
    val a = Math.pow(Math.sin(dLat / 2), 2) +
      Math.cos(Math.toRadians(lat1)) *
Math.cos(Math.toRadians(lat2)) *
        Math.pow(Math.sin(dLon / 2), 2)
    val c = 2 * Math.asin(Math.min(1.0, Math.sqrt(a)))
    R * c
  }

  def taskOne(): Unit = {
    val sparkSession = spark.sql.SparkSession.builder()
      .appName("local").getOrCreate()

    val airportsDf = readAirports(
      sparkSession,
      Vector(
        AirportField.Id,
        AirportField.Name,
        AirportField.Latitude,
        AirportField.Longitude
```

```
      )
    );
    val vertices = airportsDf.rdd.map { r =>
      (
        r.getAs[Long](AirportField.Id.name),
        (
          r.getAs[String](AirportField.Name.name),
          r.getAs[Double](AirportField.Latitude.name),
          r.getAs[Double](AirportField.Longitude.name)
        )
      )
    }

    val airportCoordMap = airportsDf.rdd.map{ r =>
      (
        r.getAs[Long](AirportField.Id.name),
        (
          r.getAs[Double](AirportField.Latitude.name),
          r.getAs[Double](AirportField.Longitude.name)
        )
      )
    }
    .collectAsMap();

    val edges = readRoutes(
      sparkSession,
      Vector(
        RouteField.AirlineId,
        RouteField.SourceAirportId,
        RouteField.DestAirportId)
      )
      .rdd.flatMap { r =>
        val airlineId = r.getLong(0)
        val srcId = r.getLong(1)
        val dstId = r.getLong(2)
        (airportCoordMap.get(srcId), airportCoordMap.get(dstId))
match {
```

```
          case (Some((slat, slon)), Some((dlat, dlon))) =>
            val dist = haversineKm(slat, slon, dlat, dlon)
            if (dist.isNaN || dist <= 0.0) None
            else Some(spark.graphx.Edge(srcId, dstId,
EdgeAttr(airlineId, dist)))
          case _ => None
        }
      }

    val graph = spark.graphx.Graph(vertices, edges)
    val totalsByAirline = graph.edges.map(e => (e.attr.airlineId,
e.attr.distanceKm))
        .reduceByKey(_ + _).cache()
    val topMax = totalsByAirline.takeOrdered(1)(Ordering.by[(Long,
Double), Double](-_._2)).headOption
    val topMin = totalsByAirline.takeOrdered(1)(Ordering.by[(Long,
Double), Double](_._2)).headOption

    val airlines = readAirlines(sparkSession,
Vector(AirlineField.Id, AirlineField.Name)).rdd.map { r =>
      (r.getLong(0), r.getString(1))
    }.collect().toMap

    def nameOf(id: Long): String = airlines.getOrElse(id,
s"Airline#$id")

    topMax.foreach { case (id, sumKm) =>
      println(f"MAX total distance: ${nameOf(id)} (airline_id=$id)
-> ${sumKm}%.2f km")
    }
    topMin.foreach { case (id, sumKm) =>
      println(f"MIN total distance: ${nameOf(id)} (airline_id=$id)
-> ${sumKm}%.2f km")
    }

    sparkSession.stop()
  }
```

Запустимо цей метод і побачимо результат.

Рисунок 2.9 — Результат першого завдання

2.3 Завдання 2

2.3.1 SQL

Для початку вирішимо завдання з допомогою spark.sql.

```
def taskTwoSql() = {
    val sparkSession =
spark.sql.SparkSession.builder().appName("local").getOrCreate()
    timeIt(sparkSession, "taskTwoSql") {
      val airports = readAirports(
        sparkSession,
        Vector(AirportField.Id, AirportField.Country)
      )
      .distinct()

      val routes = readRoutes(
        sparkSession,
        Vector(
          RouteField.SourceAirportId,
          RouteField.DestAirportId
        )
      )
      .distinct()

      import sparkSession.implicits._

      def onewayAnalyze(srcCountry: String, destCountry: String) =
{
        val airportsSrc = airports.filter($"country" ===
srcCountry)
          .as("asrc")
          .cache()
```

```
        val airportsDest = airports.filter($"country" ===
destCountry)
          .as("adest")
          .cache()

        val r0 = routes.as("r0")
        val q0 = r0
          .join(airportsSrc, $"asrc.id" === $"r0.src_id")
          .join(airportsDest, $"adest.id" === $"r0.dest_id")
          .select(spark.sql.functions.array($"asrc.id",
$"adest.id").as("path"))

        val r1 = routes.as("r1")
        val q1 = r0
          .join(r1, $"r0.dest_id" === $"r1.src_id")
          .join(airportsSrc, $"asrc.id" === $"r0.src_id")
          .join(airportsDest, $"adest.id" === $"r1.dest_id")
          .where(
            $"r0.dest_id" =!= $"asrc.id"
            && $"r0.dest_id" =!= $"adest.id"
            && $"r1.src_id" =!= $"asrc.id"
            && $"r1.src_id" =!= $"adest.id"
          )
          .select(spark.sql.functions.array($"r0.src_id",
$"r0.dest_id", $"r1.dest_id").as("path"))

        val r2 = routes.as("r2")
        val q2 = r0
          .join(r1, $"r0.dest_id" === $"r1.src_id")
          .join(r2, $"r1.dest_id" === $"r2.src_id")
          .join(airportsSrc, $"asrc.id" === $"r0.src_id")
          .join(airportsDest, $"adest.id" === $"r2.dest_id")
          .where(
            $"r0.dest_id" =!= $"asrc.id"
            && $"r0.dest_id" =!= $"adest.id"
            && $"r1.src_id" =!= $"asrc.id"
```

```
            && $"r1.src_id" =!= $"adest.id"
            && $"r1.dest_id" =!= $"asrc.id"
            && $"r1.dest_id" =!= $"adest.id"
            && $"r2.src_id" =!= $"asrc.id"
            && $"r2.src_id" =!= $"adest.id"
          )
          .select(spark.sql.functions.array($"r0.src_id",
$"r0.dest_id", $"r1.dest_id", $"r2.dest_id").as("path"))

          val q0l = q0.withColumn("stops",
spark.sql.functions.lit(0))
          val q1l = q1.withColumn("stops",
spark.sql.functions.lit(1))
          val q2l = q2.withColumn("stops",
spark.sql.functions.lit(2))
          val all = q0l.unionByName(q1l).unionByName(q2l)
          all
      }
      val srcCountry = "Poland"
      val destCountry = "France"
      val all = onewayAnalyze(srcCountry, destCountry)
        .unionByName(onewayAnalyze(destCountry, srcCountry))
        .distinct()
```

```
      all
        .limit(1000)
        .collect()
        .foreach { r =>
          val path = r.getAs[Seq[Long]]("path"); val s =
r.getAs[Int]("stops")
          println(s"[sql][$s stops] " + path.mkString(" -> "))
        }
      all
    }
    sparkSession.stop()
  }
```

```
[info] [taskTwoSql] took 24693.0 ms
```

виконання

```
[info] [sql][2 stops] 8832 -> 1555 -> 469 -> 1354
[info] [sql][2 stops] 679 -> 1555 -> 345 -> 1354
[info] [sql][2 stops] 669 -> 1613 -> 3953 -> 1354
[info] [sql][2 stops] 680 -> 1678 -> 1218 -> 1367
[info] [sql][2 stops] 668 -> 1555 -> 1273 -> 1386
[info] [sql][2 stops] 675 -> 1555 -> 1273 -> 1354
[info] [sql][2 stops] 669 -> 478 -> 1587 -> 1435
[info] [sql][2 stops] 669 -> 478 -> 1587 -> 1418
[info] [sql][2 stops] 680 -> 492 -> 1524 -> 1353
[info] [sql][2 stops] 679 -> 3953 -> 644 -> 1386
[info] [sql][2 stops] 674 -> 599 -> 302 -> 1354
[info] [sql][2 stops] 669 -> 679 -> 491 -> 1265
[info] [sql][2 stops] 679 -> 302 -> 1064 -> 1399
[info] [sql][2 stops] 669 -> 1519 -> 1562 -> 1386
[info] [sql][2 stops] 669 -> 580 -> 1590 -> 1423
[info] [sql][2 stops] 669 -> 502 -> 1324 -> 1399
[info] [sql][2 stops] 668 -> 636 -> 1200 -> 1382
[info] [sql][2 stops] 679 -> 1525 -> 490 -> 1382
[info] [sql][2 stops] 676 -> 599 -> 1289 -> 1386
[info] [sql][2 stops] 679 -> 1524 -> 1056 -> 1382
[info] [sql][2 stops] 679 -> 1638 -> 1633 -> 1386
```

Рисунок 2.10 — Шляхи

2.3.2 GraphFrames

```
  def taskTwoGraphFrames() = {
```

```
    val sparkSession =
spark.sql.SparkSession.builder().appName("local").getOrCreate()
    timeIt(sparkSession, "taskTwoGraphFrames") {

      import sparkSession.implicits._

      val airportsV = readAirports(
        sparkSession,
        Vector(AirportField.Id, AirportField.Country)
      )
      .distinct()

      val routesE = readRoutes(
        sparkSession,
        Vector(RouteField.SourceAirportId,
RouteField.DestAirportId)
      )
        .select(
          $"src_id".as("src"),
          $"dest_id".as("dst")
        )
        .distinct()

      import org.graphframes.GraphFrame

      val g = GraphFrame(airportsV, routesE)

      def onewayAnalyze(srcCountry: String, destCountry: String) =
{
        val p0 = g.find(" (a)-[e1]->(b) ")
          .filter(
            s"""
              a.country = '$srcCountry' AND b.country =
'$destCountry'
              AND a.id <> b.id
            """
          )
```

```
          .select(spark.sql.functions.array($"a.id",
$"b.id").as("path"))

        val p1 = g.find(" (a)-[e1]->(m1); (m1)-[e2]->(b) ")
          .filter(
            s"""
              a.country = '$srcCountry' AND b.country =
'$destCountry'
              AND a.id <> m1.id AND m1.id <> b.id AND a.id <> b.id
            """
          )
          .select(spark.sql.functions.array($"a.id", $"m1.id",
$"b.id").as("path"))

        val p2 = g.find(" (a)-[e1]->(m1); (m1)-[e2]->(m2); (m2)-
[e3]->(b) ")
          .filter(
            s"""
              a.country = '$srcCountry' AND b.country =
'$destCountry'
              AND a.id <> m1.id AND a.id <> m2.id AND a.id <> b.id
              AND m1.id <> m2.id AND m1.id <> b.id
              AND m2.id <> b.id
            """)
          .select(spark.sql.functions.array($"a.id", $"m1.id",
$"m2.id", $"b.id").as("path"))

        p0.withColumn("stops", spark.sql.functions.lit(0))
          .unionByName(p1.withColumn("stops",
spark.sql.functions.lit(1)))
          .unionByName(p2.withColumn("stops",
spark.sql.functions.lit(2)))
      }
      val srcCountry = "Poland"
      val destCountry = "France"

      val all = onewayAnalyze(srcCountry, destCountry)
```

```
        .unionByName(onewayAnalyze(destCountry, srcCountry))
        .distinct()

      all.limit(1000).collect().foreach { r =>
          val path = r.getAs[Seq[Long]]("path")
          val stops = r.getAs[Int]("stops")
          println(s"[motif][$stops stops] " + path.mkString(" ->
"))
      }
      all
    }
    sparkSession.stop()
  }
```

Бачимо, що шляхи побудовані.

```
[info] [motif][2 stops] 679 -> 580 -> 491 -> 1354
[info] [motif][2 stops] 8414 -> 1562 -> 492 -> 1264
[info] [motif][2 stops] 679 -> 1382 -> 1555 -> 1423
[info] [motif][2 stops] 679 -> 1382 -> 1555 -> 1264
[info] [motif][2 stops] 679 -> 1382 -> 1264 -> 1353
[info] [motif][2 stops] 679 -> 1382 -> 1520 -> 1423
[info] [motif][2 stops] 679 -> 1382 -> 293 -> 1386
[info] [motif][2 stops] 679 -> 1382 -> 1678 -> 1264
[info] [motif][2 stops] 669 -> 636 -> 502 -> 1359
[info] [motif][2 stops] 669 -> 636 -> 535 -> 1354
[info] [motif][2 stops] 675 -> 548 -> 1264 -> 1423
[info] [motif][2 stops] 675 -> 548 -> 1489 -> 1382
[info] [motif][2 stops] 675 -> 548 -> 609 -> 1280
[info] [motif][2 stops] 669 -> 1423 -> 1418 -> 1321
[info] [motif][2 stops] 669 -> 679 -> 491 -> 1265
[info] [motif][2 stops] 669 -> 679 -> 1587 -> 1354
[info] [motif][2 stops] 669 -> 679 -> 351 -> 1418
[info] [motif][2 stops] 668 -> 636 -> 1200 -> 1382
[info] [motif][2 stops] 668 -> 636 -> 1386 -> 1416
```

Рисунок 2.12 — Шляхи

Бачимо, що час виконання менший ніж в SQL-рішення.

```
[info] [taskTwoGraphFrames] took 11093.6 ms
```

Рисунок 2.13 — Час виконання

2.3.3 Завдання 3

Потрібно знайти найкоротший і найдовший шлях між еаропортами при кількості пересадок не більше 3.

```
def taskThree(): Unit = {
    val sparkSession = spark.sql.SparkSession.builder()
      .appName("local").getOrCreate()
    import sparkSession.implicits._

    val airportsDf = readAirports(
      sparkSession,
      Vector(
        AirportField.Id,
        AirportField.Name,
        AirportField.Latitude,
        AirportField.Longitude
      )
    ).distinct().cache()

    val vertices =
      airportsDf.rdd.map { r =>
        val id  = r.getAs[Long](AirportField.Id.name)
        val lat = r.getAs[Double](AirportField.Latitude.name)
        val lon = r.getAs[Double](AirportField.Longitude.name)
        (id, (lat, lon))
      }

    val id2name = airportsDf
      .select($"id", $"name")
      .as[(Long, String)]
      .collect()
      .toMap

    val bId2name = sparkSession.sparkContext.broadcast(id2name)

    val coordMap = vertices.collectAsMap()
    val bCoord   = sparkSession.sparkContext.broadcast(coordMap)
```

```
    val routesDf = readRoutes(
      sparkSession,
      Vector(RouteField.SourceAirportId, RouteField.DestAirportId)
    ).distinct().cache()

    val edges = routesDf.rdd.flatMap { r =>
      val srcId = r.getAs[Long]("src_id")
      val dstId = r.getAs[Long]("dest_id")
      (bCoord.value.get(srcId), bCoord.value.get(dstId)) match {
        case (Some((slat, slon)), Some((dlat, dlon))) =>
          val dist = haversineKm(slat, slon, dlat, dlon)
          if (!dist.isNaN && dist > 0.0)
Some(spark.graphx.Edge(srcId, dstId, dist)) else None
        case _ => None
      }
    }

    val graph = spark.graphx.Graph(vertices, edges).cache()

    import org.apache.spark.rdd.RDD
    def shortestAndLongestPaths(
        graph: spark.graphx.Graph[(Double, Double), Double],
        srcId: Long,
        dstId: Long,
        maxHops: Int = 3
    ): (Option[(Seq[Long], Double)], Option[(Seq[Long], Double)])
= {

      val sc = graph.vertices.sparkContext

      val adj = graph.edges
        .map(e => (e.srcId, (e.dstId, e.attr)))
        .groupByKey()
        .mapValues(_.toArray)
        .collectAsMap()
      val bAdj = sc.broadcast(adj)
```

```
      var frontier = sc.parallelize(Seq((Vector(srcId), 0.0)))
      var results = sc.emptyRDD[(Vector[Long], Double)]

      for (_ <- 1 to maxHops) {
        val expanded = frontier.flatMap { case (path, d) =>
          val last = path.last
          bAdj.value
            .getOrElse(last, Array.empty[(Long, Double)])
            .iterator
            .filter { case (nxt, _) => !path.contains(nxt) }
            .map   { case (nxt, w)  => (path :+ nxt, d + w) }
        }.persist()

        val hits = expanded.filter { case (p, _) => p.last ==
dstId }
        results = results.union(hits)

        frontier = expanded.filter { case (p, _) => p.last !=
dstId }
      }

      val shortest = results.takeOrdered(1)
(Ordering.by(_._2)).headOption
      val longest  = results.takeOrdered(1)(Ordering.by(-
_._2)).headOption

      (
        shortest.map{ case (p, d) => (p.toSeq, d) },
        longest .map{ case (p, d) => (p.toSeq, d) }
      )
    }

    val srcId = 679
    val dstId = 1382

    val (shortestOpt, longestOpt) = shortestAndLongestPaths(graph,
```

```
srcId, dstId, maxHops = 3)

    def fmtPath(p: Seq[Long]): String =
      p.map(id => s"$id:${bId2name.value.getOrElse(id,
"???")}").mkString(" -> ")

    shortestOpt match {
      case Some((path, km)) =>
        println(f"[shortest <=3 hops]\n  ${fmtPath(path)}\n  total
= $km%.2f km")
      case None =>
        println("No path (shortest).")
    }

    longestOpt match {
      case Some((path, km)) =>
        println(f"[longest <=3 hops]\n  ${fmtPath(path)}\n  total
= $km%.2f km")
      case None =>
        println("No path (longest).")
    }

    sparkSession.stop()
  }
```

```
[info] [shortest <=3 hops]
[info]   679:Warsaw Chopin Airport -> 1382:Charles de Gaulle International Airport
[info]   total = 1342.49 km
[info] [longest <=3 hops]
[info]   679:Warsaw Chopin Airport -> 3830:Chicago O'Hare International Airport -> 3077:Chek Lap Kok International Airport -> 1382:Charles de Gaulle International Airport
[info]   total = 29631.53 km
```

Рисунок 2.14 — Результат виконання

2.4 Завдання 5

Виділимо великі кластери аеропортів з допомогою connectedComponents.

```
  def taskFive() = {
    val sparkSession =
spark.sql.SparkSession.builder().appName("local").getOrCreate()
    val airportsDf = readAirports(
      sparkSession,
      Vector(
```

```
      AirportField.Id,
      AirportField.Name,
      AirportField.Country
    )
  )
  val vertices = airportsDf.rdd.map { r =>
    (r.getLong(0), (r.getString(1), r.getString(2)))
  }

  val edges = readRoutes(
    sparkSession,
      Vector(
        RouteField.SourceAirportId,
        RouteField.DestAirportId
      )
  ).rdd.map { r =>
    spark.graphx.Edge(r.getLong(0), r.getLong(1), ())
  }
  val graph = spark.graphx.Graph(vertices, edges)
  val cc = graph.connectedComponents().vertices

  val labeled = cc.join(vertices).map {
    case (id, (compId, (name, country))) =>
      (compId, (id, name, country))
  }

  val clusters = labeled.groupByKey()
    .mapValues(_.toSeq)
    .filter { case (_, members) => members.size >= 5 }

  val sortedClusters = clusters.sortByKey()

  sortedClusters.take(10).foreach { case (compId, members) =>
    println(s"Cluster $compId (size=${members.size}):")
    members.take(10).foreach { case (id, name, country) =>
      println(s"  $id $name ($country)")
    }
```

```
    }

    sparkSession.stop()
  }
```

```
[info] Cluster 1 (size=3304):
[info]   3877 McCarran International Airport (United States)
[info]   1 Goroka Airport (Papua New Guinea)
[error] 25/09/30 08:34:39 INFO SparkContext: SparkContext is stopping with exitCode 0.
[info]   5928 Bam Airport (Iran)
[info]   2334 Osaka International Airport (Japan)
[info]   1813 Lázaro Cárdenas Airport (Mexico)
[info]   9025 Bijie Feixiong Airport (China)
[info]   9829 Mbeya Airport (Tanzania)
[info]   7456 Raivavae Airport (French Polynesia)
[info]   1596 Haifa International Airport (Israel)
[info]   1780 Sangster International Airport (Jamaica)
[info] Cluster 1998 (size=10):
[info]   5920 Île Art - Waala Airport (New Caledonia)
[info]   5921 Île des Pins Airport (New Caledonia)
[info]   2001 Nouméa Magenta Airport (New Caledonia)
[info]   2002 Maré Airport (New Caledonia)
[info]   5919 Tiga Airport (New Caledonia)
[info]   1998 Koné Airport (New Caledonia)
[info]   1999 Koumac Airport (New Caledonia)
[info]   2000 Lifou Airport (New Caledonia)
[info]   2004 Ouvéa Airport (New Caledonia)
[info]   2003 Touho Airport (New Caledonia)
```

Рисунок 2.15 — Результат виконання

Те що в логах пишеться [error] — це нормально, бо за замовчуванням spark кидає всі логи в stderr.

2.5 Завдання 6

Покажемо впливовість аеропортів з допомогою pageRank.

```
  def taskSix() = {
    val sparkSession =
spark.sql.SparkSession.builder().appName("local").getOrCreate()
    import sparkSession.implicits._

    val airportsDf = readAirports(
      sparkSession,
      Vector(
        AirportField.Id,
        AirportField.Name,
        AirportField.Country
      )
```

```
    )
    val vertices = airportsDf.rdd.map { r =>
      (r.getLong(0), (r.getString(1), r.getString(2)))
    }

    val edges = readRoutes(
      sparkSession,
        Vector(
          RouteField.SourceAirportId,
          RouteField.DestAirportId
        )
    ).rdd.map { r =>
      spark.graphx.Edge(r.getLong(0), r.getLong(1), ())
    }
    val graph = spark.graphx.Graph(vertices, edges)
    val tol = 0.00001
    val ranks = graph.pageRank(tol).vertices

    val labeled = ranks.join(vertices)
      .map { case (id, (rank, (name, country))) => (id, name,
country, rank) }

    val top10 = labeled.takeOrdered(10)
(Ordering.by[(Long,String,String,Double), Double](-_._4))

    println("\n=== PageRank: Top 10 airports ===")
    top10.zipWithIndex.foreach { case ((id, name, country, rank),
i) =>
      println(f"${i+1}%2d) id=$id%6d  PR=${rank}%.6f  $name
($country)")
    }

    sparkSession.stop()
  }
```

Рисунок 2.16 — Найбільш впливові аеропорти

2.6 Завдання 7

2.6.1 ConnectedComponents

```
def showClusters(
    airportIdWithComponentIdRdd: spark.graphx.VertexRDD[Long],
    vertices: spark.rdd.RDD[(Long, (String, String))],
    title: String,
    minSize: Int = 2,
    topN: Int = 10
  ): Unit = {

    val componentIdSizeRdd = airportIdWithComponentIdRdd.map
{ case (_, cid) => (cid, 1) }.reduceByKey(_ + _)
    val largestComponentId = componentIdSizeRdd.max()
(Ordering.by(_._2))._1

    val filteredComponentIds = componentIdSizeRdd
      .filter { case (cid, sz) => sz >= minSize && cid !=
largestComponentId }
      .map(_._1)

    airportIdWithComponentIdRdd.map { case (airportId,
componentId) => (componentId, airportId) }
      .join(filteredComponentIds.map((_, ())))
      .map { case (componentId, (airportId, _)) => (airportId,
componentId) }
      .join(vertices)
      .map { case (airportId, (componentId, (name, country))) =>
```

```
(componentId, (airportId, name, country)) }
      .groupByKey()
      .mapValues(_.toArray)
      .foreach { case (componentId, info) =>
        println(s"Component $componentId (size=${info.size})")
        info
          .take(topN)
          .foreach { case (airportId, name, country) =>
          println(s"  $airportId $name ($country)")
        }
      }
  }
def taskSevenConnectedComponents() = {
    val sparkSession =
spark.sql.SparkSession.builder().appName("local").getOrCreate()
    import sparkSession.implicits._
    val airportsDf = readAirports(
      sparkSession,
      Vector(
        AirportField.Id,
        AirportField.Name,
        AirportField.Country
      )
    )
    val vertices = airportsDf.rdd.map { r =>
      (r.getLong(0), (r.getString(1), r.getString(2)))
    }

    val edges = readRoutes(
      sparkSession,
        Vector(
          RouteField.SourceAirportId,
          RouteField.DestAirportId
        )
    ).rdd.map { r =>
      spark.graphx.Edge(r.getLong(0), r.getLong(1), ())
    }
```

```
    val graph = spark.graphx.Graph(vertices, edges)
    val airportIdWithComponentIdRdd =
graph.connectedComponents().vertices
    showClusters(airportIdWithComponentIdRdd, vertices,
"connectedComponents")
    sparkSession.stop()
  }
```

```
[info] Component 7111 (size=6)
[info]   7208 Koyukuk Airport (United States)
[info]   6717 Kaltag Airport (United States)
[info]   3764 Edward G. Pitka Sr Airport (United States)
[info]   7108 Huslia Airport (United States)
[info]   7111 Nulato Airport (United States)
[info]   7107 Hughes Airport (United States)
[info] Component 196 (size=3)
[info]   109 Fort Chipewyan Airport (Canada)
[info]   196 Yellowknife Airport (Canada)
[info]   4238 Tulita Airport (Canada)
[info] Component 5 (size=26)
[info]   1 Goroka Airport (Papua New Guinea)
[info]   5420 Chimbu Airport (Papua New Guinea)
[info]   4 Nadzab Airport (Papua New Guinea)
[info]   5428 Kavieng Airport (Papua New Guinea)
[info]   5429 Mendi Airport (Papua New Guinea)
[info]   5425 Kiunga Airport (Papua New Guinea)
[info]   5437 Wapenamanda Airport (Papua New Guinea)
[info]   3 Mount Hagen Kagamuga Airport (Papua New Guinea)
[info]   5431 Moro Airport (Papua New Guinea)
[info]   5423 Girua Airport (Papua New Guinea)
[info] Component 5906 (size=2)
[info]   5894 Sola Airport (Vanuatu)
[info]   5906 Gaua Island Airport (Vanuatu)
[info] Component 2688 (size=2)
[info]   10792 Aeropuerto Jumandy (Ecuador)
[info]   2680 Coronel E Carvajal Airport (Ecuador)
[info] Component 6265 (size=2)
[info]   6265 Elcho Island Airport (Australia)
[info]   6295 Milingimbi Airport (Australia)
[info] Component 2487 (size=2)
[info]   2491 Gobernador Castello Airport (Argentina)
[info]   2487 General E. Mosconi Airport (Argentina)
```

Рисунок 2.17 — Частина результату

2.6.2 LabelPropagation

```
def taskSevenLabelPropagation() = {
    val sparkSession =
spark.sql.SparkSession.builder().appName("local").getOrCreate()
    import sparkSession.implicits._
    val airportsDf = readAirports(
      sparkSession,
      Vector(
        AirportField.Id,
        AirportField.Name,
        AirportField.Country
      )
    )
    val vertices = airportsDf.rdd.map { r =>
      (r.getLong(0), (r.getString(1), r.getString(2)))
    }
    val edges = readRoutes(
      sparkSession,
        Vector(
          RouteField.SourceAirportId,
          RouteField.DestAirportId
        )
    ).rdd.map { r =>
      spark.graphx.Edge(r.getLong(0), r.getLong(1), ())
    }
    val graph = spark.graphx.Graph(vertices, edges)
    val airportIdWithComponentIdRdd =
spark.graphx.lib.LabelPropagation.run(graph, maxSteps =
100).vertices

    showClusters(airportIdWithComponentIdRdd, vertices,
"LabelPropagation")
    sparkSession.stop()
  }
```

```
[info] Component 1998 (size=10)
[info]   5920 Île Art - Waala Airport (New Caledonia)
[info]   5921 Île des Pins Airport (New Caledonia)
[info]   2001 Nouméa Magenta Airport (New Caledonia)
[info]   2002 Maré Airport (New Caledonia)
[info]   5919 Tiga Airport (New Caledonia)
[info]   1998 Koné Airport (New Caledonia)
[info]   1999 Koumac Airport (New Caledonia)
[info]   2000 Lifou Airport (New Caledonia)
[info]   2004 Ouvéa Airport (New Caledonia)
[info]   2003 Touho Airport (New Caledonia)
[info] Component 5642 (size=4)
[info]   5642 Ondangwa Airport (Namibia)
[info]   6779 Katima Mulilo Airport (Namibia)
[info]   5645 Eros Airport (Namibia)
[info]   7634 Rundu Airport (Namibia)
[info] Component 7309 (size=2)
[info]   7309 Charlotte Amalie Harbor Seaplane Base (Virgin Islands)
[info]   7310 Christiansted Harbor Seaplane Base (Virgin Islands)
[info] Component 3860 (size=4)
[info]   7196 Nikolski Air Station (United States)
[info]   7195 Atka Airport (United States)
[info]   6134 Akutan Seaplane Base (United States)
[info]   3860 Unalaska Airport (United States)
[info] Component 3726 (size=4)
[info]   3726 Boeing Field King County International Airport (United States)
[info]   7082 Friday Harbor Airport (United States)
[info]   5731 William R Fairchild International Airport (United States)
[info]   7083 Orcas Island Airport (United States)
[info] Component 6448 (size=2)
[info]   6449 Boulder City Municipal Airport (United States)
[info]   6448 Grand Canyon West Airport (United States)
```

Рисунок 2.18 — Результат виконання

3 ВИСНОВОК

У результаті виконнаня відпрацювали роботу з графовими структурами.

4 ЛІСТИНГ КОДУ

```
import org.apache.{spark => spark}

object Main {
  private def timeIt[A](sparkSession: spark.sql.SparkSession,
label: String)(f: => A): A = {
    val t0 = System.nanoTime()
    val res = f

sparkSession.sparkContext.runJob(sparkSession.sparkContext.range(0
, 1), (_: Iterator[Long]) => ())
    val t1 = System.nanoTime()
    println(f"[$label] took ${(t1 - t0)/1e3/1e3}%.1f ms")
    res
  }
  case class EdgeAttr(airlineId: Long, distanceKm: Double)
  val sampledataOpenFlightsOrg = "sampledata/OpenFlights.org/";
  val csvOpts = Map(
    "header" -> "false",
    "inferSchema" -> "false",
    "quote" -> "\"",
    "escape" -> "\"",
    "mode" -> "PERMISSIVE",
    "multiLine" -> "false",
    "ignoreLeadingWhiteSpace" -> "true",
    "ignoreTrailingWhiteSpace" -> "true"
  );
  sealed trait SqlField {
    def sqlType: spark.sql.types.DataType
    def name: String
    def index: Int
  }
  sealed trait AirportField extends SqlField
  object AirportField {
    case object Id extends AirportField {
```

```
    val sqlType = spark.sql.types.LongType
    val name = "id"
    val index = 0
  }
  case object Name extends AirportField {
    val sqlType = spark.sql.types.StringType
    val name = "name"
    val index = 1
  }
  case object City extends AirportField {
    val sqlType = spark.sql.types.StringType
    val name = "city"
    val index = 2
  }
  case object Country extends AirportField {
    val sqlType = spark.sql.types.StringType
    val name = "country"
    val index = 3
  }
  case object IATA extends AirportField {
    val sqlType = spark.sql.types.StringType
    val name = "iata"
    val index = 4
  }
  case object ICAO extends AirportField {
    val sqlType = spark.sql.types.StringType
    val name = "icao"
    val index = 5
  }
  case object Latitude extends AirportField {
    val sqlType = spark.sql.types.DoubleType
    val name = "latitude"
    val index = 6
  }
  case object Longitude extends AirportField {
    val sqlType = spark.sql.types.DoubleType
    val name = "longitude"
```

```
      val index = 7
    }
    case object Altitude extends AirportField {
      val sqlType = spark.sql.types.DoubleType
      val name = "altitude"
      val index = 8
    }
    case object TimezoneOffset extends AirportField {
      val sqlType = spark.sql.types.IntegerType
      val name = "timezone_offset"
      val index = 9
    }
    case object DaylightSavingTime extends AirportField {
      val sqlType = spark.sql.types.StringType
      val name = "daylight_saving_time"
      val index = 10
    }
  }
  sealed trait AirlineField extends SqlField
  object AirlineField {
    case object Id extends AirlineField {
      val sqlType = spark.sql.types.LongType
      val name = "id"
      val index = 0
    }
    case object Name extends AirlineField {
      val sqlType = spark.sql.types.StringType
      val name = "name"
      val index = 1
    }
    case object Alias extends AirlineField {
      val sqlType = spark.sql.types.StringType
      val name = "alias"
      val index = 2
    }
    case object IATA extends AirlineField {
      val sqlType = spark.sql.types.StringType
```

```
      val name = "iata"
      val index = 3
    }
    case object ICAO extends AirlineField {
      val sqlType = spark.sql.types.StringType
      val name = "icao"
      val index = 4
    }
    case object Callsign extends AirlineField {
      val sqlType = spark.sql.types.StringType
      val name = "callsign"
      val index = 5
    }
    case object Country extends AirlineField {
      val sqlType = spark.sql.types.StringType
      val name = "country"
      val index = 6
    }
    case object Active extends AirlineField {
      val sqlType = spark.sql.types.StringType
      val name = "active"
      val index = 7
    }
  }
  sealed trait RouteField extends SqlField
  object RouteField {
    case object Airline extends RouteField {
      val sqlType = spark.sql.types.StringType
      val name = "airline"
      val index = 0
    }
    case object AirlineId extends RouteField {
      val sqlType = spark.sql.types.LongType
      val name = "airline_id"
      val index = 1
    }
    case object SourceAirport extends RouteField {
```

```
      val sqlType = spark.sql.types.StringType
      val name = "src"
      val index = 2
    }
    case object SourceAirportId extends RouteField {
      val sqlType = spark.sql.types.LongType
      val name = "src_id"
      val index = 3
    }
    case object DestAirport extends RouteField {
      val sqlType = spark.sql.types.StringType
      val name = "dest"
      val index = 4
    }
    case object DestAirportId extends RouteField {
      val sqlType = spark.sql.types.LongType
      val name = "dest_id"
      val index = 5
    }
    case object Codeshare extends RouteField {
      val sqlType = spark.sql.types.StringType
      val name = "codeshare"
      val index = 6
    }
    case object Stops extends RouteField {
      val sqlType = spark.sql.types.IntegerType
      val name = "stops"
      val index = 7
    }
    case object Equipment extends RouteField {
      val sqlType = spark.sql.types.StringType
      val name = "equipment"
      val index = 8
    }
  }
  def readData(sparkSession: spark.sql.SparkSession, name: String,
fields: Vector[SqlField]) = {
```

```
    val columns = fields.map { field =>
      spark.sql.functions.col(s"_c${field.index}").as(field.name)
    }
    val baseDf =
sparkSession.read.options(csvOpts).csv(sampledataOpenFlightsOrg +
name)
      .select(columns: _*)
      .filter(columns.map { col => col.isNotNull }.reduce(_ && _))
      .filter(columns.map { col => !col.equalTo("\\N") }.reduce(_
&& _))

    fields.foldLeft(baseDf) { (df, field) =>
      df.withColumn(field.name,
spark.sql.functions.col(field.name).cast(field.sqlType))
    }
    .select(fields.map(f => spark.sql.functions.col(f.name)): _*)
  }
  def readAirports(sparkSession: spark.sql.SparkSession, fields:
Vector[AirportField]) = {
    readData(sparkSession, "airports-extended.dat", fields)
  }
  def readAirlines(sparkSession: spark.sql.SparkSession, fields:
Vector[AirlineField]) = {
    readData(sparkSession, "airlines.dat", fields)
  }
  def readRoutes(sparkSession: spark.sql.SparkSession, fields:
Vector[RouteField]) = {
    readData(sparkSession, "routes.dat", fields)
  }

  def haversineKm(lat1: Double, lon1: Double, lat2: Double, lon2:
Double): Double = {
    val R = 6371.0088
    val dLat = Math.toRadians(lat2 - lat1)
    val dLon = Math.toRadians(lon2 - lon1)
    val a = Math.pow(Math.sin(dLat / 2), 2) +
      Math.cos(Math.toRadians(lat1)) *
```

```
Math.cos(Math.toRadians(lat2)) *
        Math.pow(Math.sin(dLon / 2), 2)
    val c = 2 * Math.asin(Math.min(1.0, Math.sqrt(a)))
    R * c
  }

  def taskOne(): Unit = {
    val sparkSession = spark.sql.SparkSession.builder()
      .appName("local").getOrCreate()

    val airportsDf = readAirports(
      sparkSession,
      Vector(
        AirportField.Id,
        AirportField.Name,
        AirportField.Latitude,
        AirportField.Longitude
      )
    );
    val vertices = airportsDf.rdd.map { r =>
        (
          r.getAs[Long](AirportField.Id.name),
          (
            r.getAs[String](AirportField.Name.name),
            r.getAs[Double](AirportField.Latitude.name),
            r.getAs[Double](AirportField.Longitude.name)
          )
        )
      }

    val airportCoordMap = airportsDf.rdd.map{ r =>
      (
        r.getAs[Long](AirportField.Id.name),
        (
          r.getAs[Double](AirportField.Latitude.name),
          r.getAs[Double](AirportField.Longitude.name)
        )
```

```
      )
    }
    .collectAsMap();

    val edges = readRoutes(
      sparkSession,
      Vector(
        RouteField.AirlineId,
        RouteField.SourceAirportId,
        RouteField.DestAirportId)
      )
      .rdd.flatMap { r =>
        val airlineId = r.getLong(0)
        val srcId = r.getLong(1)
        val dstId = r.getLong(2)
        (airportCoordMap.get(srcId), airportCoordMap.get(dstId))
match {
          case (Some((slat, slon)), Some((dlat, dlon))) =>
            val dist = haversineKm(slat, slon, dlat, dlon)
            if (dist.isNaN || dist <= 0.0) None
            else Some(spark.graphx.Edge(srcId, dstId,
EdgeAttr(airlineId, dist)))
          case _ => None
        }
      }

    val graph = spark.graphx.Graph(vertices, edges)
    val totalsByAirline = graph.edges.map(e => (e.attr.airlineId,
e.attr.distanceKm))
        .reduceByKey(_ + _).cache()
    val topMax = totalsByAirline.takeOrdered(1)(Ordering.by[(Long,
Double), Double](-_._2)).headOption
    val topMin = totalsByAirline.takeOrdered(1)(Ordering.by[(Long,
Double), Double](_._2)).headOption

    val airlines = readAirlines(sparkSession,
Vector(AirlineField.Id, AirlineField.Name)).rdd.map { r =>
```

```
      (r.getLong(0), r.getString(1))
    }.collect().toMap

    def nameOf(id: Long): String = airlines.getOrElse(id,
s"Airline#$id")

    topMax.foreach { case (id, sumKm) =>
      println(f"MAX total distance: ${nameOf(id)} (airline_id=$id)
-> ${sumKm}%.2f km")
    }
    topMin.foreach { case (id, sumKm) =>
      println(f"MIN total distance: ${nameOf(id)} (airline_id=$id)
-> ${sumKm}%.2f km")
    }

    sparkSession.stop()
  }

  def taskTwoGraphFrames() = {
    val sparkSession =
spark.sql.SparkSession.builder().appName("local").getOrCreate()
    timeIt(sparkSession, "taskTwoGraphFrames") {

      import sparkSession.implicits._

      val airportsV = readAirports(
        sparkSession,
        Vector(AirportField.Id, AirportField.Country)
      )
      .distinct()

      val routesE = readRoutes(
        sparkSession,
        Vector(RouteField.SourceAirportId,
RouteField.DestAirportId)
      )
        .select(
```

```
          $"src_id".as("src"),
          $"dest_id".as("dst")
        )
        .distinct()

      import org.graphframes.GraphFrame

      val g = GraphFrame(airportsV, routesE)

      def onewayAnalyze(srcCountry: String, destCountry: String) =
{
        val p0 = g.find(" (a)-[e1]->(b) ")
          .filter(
            s"""
              a.country = '$srcCountry' AND b.country =
'$destCountry'
              AND a.id <> b.id
            """
          )
          .select(spark.sql.functions.array($"a.id",
$"b.id").as("path"))

        val p1 = g.find(" (a)-[e1]->(m1); (m1)-[e2]->(b) ")
          .filter(
            s"""
              a.country = '$srcCountry' AND b.country =
'$destCountry'
              AND a.id <> m1.id AND m1.id <> b.id AND a.id <> b.id
            """
          )
          .select(spark.sql.functions.array($"a.id", $"m1.id",
$"b.id").as("path"))

        val p2 = g.find(" (a)-[e1]->(m1); (m1)-[e2]->(m2); (m2)-
[e3]->(b) ")
          .filter(
            s"""
```

```
              a.country = '$srcCountry' AND b.country =
'$destCountry'
              AND a.id <> m1.id AND a.id <> m2.id AND a.id <> b.id
              AND m1.id <> m2.id AND m1.id <> b.id
              AND m2.id <> b.id
            """)
          .select(spark.sql.functions.array($"a.id", $"m1.id",
$"m2.id", $"b.id").as("path"))

        p0.withColumn("stops", spark.sql.functions.lit(0))
          .unionByName(p1.withColumn("stops",
spark.sql.functions.lit(1)))
          .unionByName(p2.withColumn("stops",
spark.sql.functions.lit(2)))
      }
      val srcCountry = "Poland"
      val destCountry = "France"

      val all = onewayAnalyze(srcCountry, destCountry)
        .unionByName(onewayAnalyze(destCountry, srcCountry))
        .distinct()

      all.limit(1000).collect().foreach { r =>
          val path = r.getAs[Seq[Long]]("path")
          val stops = r.getAs[Int]("stops")
          println(s"[motif][$stops stops] " + path.mkString(" ->
"))
      }
      all
    }
    sparkSession.stop()
  }

  def taskTwoSql() = {
    val sparkSession =
spark.sql.SparkSession.builder().appName("local").getOrCreate()
    timeIt(sparkSession, "taskTwoSql") {
```

```
      val airports = readAirports(
        sparkSession,
        Vector(AirportField.Id, AirportField.Country)
      )
      .distinct()

      val routes = readRoutes(
        sparkSession,
        Vector(
          RouteField.SourceAirportId,
          RouteField.DestAirportId
        )
      )
      .distinct()

      import sparkSession.implicits._

      def onewayAnalyze(srcCountry: String, destCountry: String) =
{
        val airportsSrc = airports.filter($"country" ===
srcCountry)
          .as("asrc")
          .cache()

        val airportsDest = airports.filter($"country" ===
destCountry)
          .as("adest")
          .cache()

        val r0 = routes.as("r0")
        val q0 = r0
          .join(airportsSrc, $"asrc.id" === $"r0.src_id")
          .join(airportsDest, $"adest.id" === $"r0.dest_id")
          .select(spark.sql.functions.array($"asrc.id",
$"adest.id").as("path"))

        val r1 = routes.as("r1")
```

```
        val q1 = r0
          .join(r1, $"r0.dest_id" === $"r1.src_id")
          .join(airportsSrc, $"asrc.id" === $"r0.src_id")
          .join(airportsDest, $"adest.id" === $"r1.dest_id")
          .where(
            $"r0.dest_id" =!= $"asrc.id"
            && $"r0.dest_id" =!= $"adest.id"
            && $"r1.src_id" =!= $"asrc.id"
            && $"r1.src_id" =!= $"adest.id"
          )
          .select(spark.sql.functions.array($"r0.src_id",
$"r0.dest_id", $"r1.dest_id").as("path"))

        val r2 = routes.as("r2")
        val q2 = r0
          .join(r1, $"r0.dest_id" === $"r1.src_id")
          .join(r2, $"r1.dest_id" === $"r2.src_id")
          .join(airportsSrc, $"asrc.id" === $"r0.src_id")
          .join(airportsDest, $"adest.id" === $"r2.dest_id")
          .where(
            $"r0.dest_id" =!= $"asrc.id"
            && $"r0.dest_id" =!= $"adest.id"
            && $"r1.src_id" =!= $"asrc.id"
            && $"r1.src_id" =!= $"adest.id"
            && $"r1.dest_id" =!= $"asrc.id"
            && $"r1.dest_id" =!= $"adest.id"
            && $"r2.src_id" =!= $"asrc.id"
            && $"r2.src_id" =!= $"adest.id"
          )
          .select(spark.sql.functions.array($"r0.src_id",
$"r0.dest_id", $"r1.dest_id", $"r2.dest_id").as("path"))

          val q0l = q0.withColumn("stops",
spark.sql.functions.lit(0))
          val q1l = q1.withColumn("stops",
spark.sql.functions.lit(1))
          val q2l = q2.withColumn("stops",
```

```
spark.sql.functions.lit(2))
          val all = q0l.unionByName(q1l).unionByName(q2l)
          all
      }
      val srcCountry = "Poland"
      val destCountry = "France"
      val all = onewayAnalyze(srcCountry, destCountry)
        .unionByName(onewayAnalyze(destCountry, srcCountry))
        .distinct()
      all
        .limit(1000)
        .collect()
        .foreach { r =>
          val path = r.getAs[Seq[Long]]("path"); val s =
r.getAs[Int]("stops")
          println(s"[sql][$s stops] " + path.mkString(" -> "))
        }
      all
    }
    sparkSession.stop()
  }
  def taskThree(): Unit = {
    val sparkSession = spark.sql.SparkSession.builder()
      .appName("local").getOrCreate()
    import sparkSession.implicits._

    val airportsDf = readAirports(
      sparkSession,
      Vector(
        AirportField.Id,
        AirportField.Name,
        AirportField.Latitude,
        AirportField.Longitude
      )
    ).distinct().cache()

    val vertices =
```

```
      airportsDf.rdd.map { r =>
        val id  = r.getAs[Long](AirportField.Id.name)
        val lat = r.getAs[Double](AirportField.Latitude.name)
        val lon = r.getAs[Double](AirportField.Longitude.name)
        (id, (lat, lon))
      }

    val id2name = airportsDf
      .select($"id", $"name")
      .as[(Long, String)]
      .collect()
      .toMap

    val bId2name = sparkSession.sparkContext.broadcast(id2name)

    val coordMap = vertices.collectAsMap()
    val bCoord   = sparkSession.sparkContext.broadcast(coordMap)

    val routesDf = readRoutes(
      sparkSession,
      Vector(RouteField.SourceAirportId, RouteField.DestAirportId)
    ).distinct().cache()

    val edges = routesDf.rdd.flatMap { r =>
      val srcId = r.getAs[Long]("src_id")
      val dstId = r.getAs[Long]("dest_id")
      (bCoord.value.get(srcId), bCoord.value.get(dstId)) match {
        case (Some((slat, slon)), Some((dlat, dlon))) =>
          val dist = haversineKm(slat, slon, dlat, dlon)
          if (!dist.isNaN && dist > 0.0)
Some(spark.graphx.Edge(srcId, dstId, dist)) else None
        case _ => None
      }
    }

    val graph = spark.graphx.Graph(vertices, edges).cache()
```

```
    import org.apache.spark.rdd.RDD
    def shortestAndLongestPaths(
        graph: spark.graphx.Graph[(Double, Double), Double],
        srcId: Long,
        dstId: Long,
        maxHops: Int = 3
    ): (Option[(Seq[Long], Double)], Option[(Seq[Long], Double)])
= {

      val sc = graph.vertices.sparkContext

      val adj = graph.edges
        .map(e => (e.srcId, (e.dstId, e.attr)))
        .groupByKey()
        .mapValues(_.toArray)
        .collectAsMap()
      val bAdj = sc.broadcast(adj)

      var frontier = sc.parallelize(Seq((Vector(srcId), 0.0)))
      var results = sc.emptyRDD[(Vector[Long], Double)]

      for (_ <- 1 to maxHops) {
        val expanded = frontier.flatMap { case (path, d) =>
          val last = path.last
          bAdj.value
            .getOrElse(last, Array.empty[(Long, Double)])
            .iterator
            .filter { case (nxt, _) => !path.contains(nxt) }
            .map   { case (nxt, w)  => (path :+ nxt, d + w) }
        }.persist()

        val hits = expanded.filter { case (p, _) => p.last ==
dstId }
        results = results.union(hits)

        frontier = expanded.filter { case (p, _) => p.last !=
dstId }
```

```
      }

      val shortest = results.takeOrdered(1)
(Ordering.by(_._2)).headOption
      val longest  = results.takeOrdered(1)(Ordering.by(-
_._2)).headOption

      (
        shortest.map{ case (p, d) => (p.toSeq, d) },
        longest .map{ case (p, d) => (p.toSeq, d) }
      )
    }

    val srcId = 679
    val dstId = 1382

    val (shortestOpt, longestOpt) = shortestAndLongestPaths(graph,
srcId, dstId, maxHops = 3)

    def fmtPath(p: Seq[Long]): String =
      p.map(id => s"$id:${bId2name.value.getOrElse(id,
"???")}").mkString(" -> ")

    shortestOpt match {
      case Some((path, km)) =>
        println(f"[shortest <=3 hops]\n  ${fmtPath(path)}\n  total
= $km%.2f km")
      case None =>
        println("No path (shortest).")
    }

    longestOpt match {
      case Some((path, km)) =>
        println(f"[longest <=3 hops]\n  ${fmtPath(path)}\n  total
= $km%.2f km")
      case None =>
        println("No path (longest).")
```

```
    }

    sparkSession.stop()
  }

  def taskFive() = {
    val sparkSession =
spark.sql.SparkSession.builder().appName("local").getOrCreate()
    val airportsDf = readAirports(
      sparkSession,
      Vector(
        AirportField.Id,
        AirportField.Name,
        AirportField.Country
      )
    )
    val vertices = airportsDf.rdd.map { r =>
      (r.getLong(0), (r.getString(1), r.getString(2)))
    }

    val edges = readRoutes(
      sparkSession,
        Vector(
          RouteField.SourceAirportId,
          RouteField.DestAirportId
        )
    ).rdd.map { r =>
      spark.graphx.Edge(r.getLong(0), r.getLong(1), ())
    }
    val graph = spark.graphx.Graph(vertices, edges)
    val cc = graph.connectedComponents().vertices

    val labeled = cc.join(vertices).map {
      case (id, (compId, (name, country))) =>
        (compId, (id, name, country))
    }
```

```
    val clusters = labeled.groupByKey()
      .mapValues(_.toSeq)
      .filter { case (_, members) => members.size >= 5 }

    val sortedClusters = clusters.sortByKey()

    sortedClusters.take(10).foreach { case (compId, members) =>
      println(s"Cluster $compId (size=${members.size}):")
      members.take(10).foreach { case (id, name, country) =>
        println(s"  $id $name ($country)")
      }
    }

    sparkSession.stop()
  }
  def taskSix() = {
    val sparkSession =
spark.sql.SparkSession.builder().appName("local").getOrCreate()
    import sparkSession.implicits._

    val airportsDf = readAirports(
      sparkSession,
      Vector(
        AirportField.Id,
        AirportField.Name,
        AirportField.Country
      )
    )
    val vertices = airportsDf.rdd.map { r =>
      (r.getLong(0), (r.getString(1), r.getString(2)))
    }

    val edges = readRoutes(
      sparkSession,
        Vector(
          RouteField.SourceAirportId,
          RouteField.DestAirportId
```

```
        )
    ).rdd.map { r =>
      spark.graphx.Edge(r.getLong(0), r.getLong(1), ())
    }
    val graph = spark.graphx.Graph(vertices, edges)
    val tol = 0.00001
    val ranks = graph.pageRank(tol).vertices

    val labeled = ranks.join(vertices)
      .map { case (id, (rank, (name, country))) => (id, name,
country, rank) }

    val top10 = labeled.takeOrdered(10)
(Ordering.by[(Long,String,String,Double), Double](-_._4))

    println("\n=== PageRank: Top 10 airports ===")
    top10.zipWithIndex.foreach { case ((id, name, country, rank),
i) =>
      println(f"${i+1}%2d) id=$id%6d  PR=${rank}%.6f  $name
($country)")
    }

    sparkSession.stop()
  }

  def showClusters(
    airportIdWithComponentIdRdd: spark.graphx.VertexRDD[Long],
    vertices: spark.rdd.RDD[(Long, (String, String))],
    title: String,
    minSize: Int = 2,
    topN: Int = 10
  ): Unit = {

    val componentIdSizeRdd = airportIdWithComponentIdRdd.map
{ case (_, cid) => (cid, 1) }.reduceByKey(_ + _)
    val largestComponentId = componentIdSizeRdd.max()
(Ordering.by(_._2))._1
```

```
    val filteredComponentIds = componentIdSizeRdd
      .filter { case (cid, sz) => sz >= minSize && cid !=
largestComponentId }
      .map(_._1)

    airportIdWithComponentIdRdd.map { case (airportId,
componentId) => (componentId, airportId) }
      .join(filteredComponentIds.map((_, ())))
      .map { case (componentId, (airportId, _)) => (airportId,
componentId) }
      .join(vertices)
      .map { case (airportId, (componentId, (name, country))) =>
(componentId, (airportId, name, country)) }
      .groupByKey()
      .mapValues(_.toArray)
      .foreach { case (componentId, info) =>
        println(s"Component $componentId (size=${info.size})")
        info
          .take(topN)
          .foreach { case (airportId, name, country) =>
          println(s"  $airportId $name ($country)")
        }
      }
  }
  def taskSevenConnectedComponents() = {
    val sparkSession =
spark.sql.SparkSession.builder().appName("local").getOrCreate()
    import sparkSession.implicits._
    val airportsDf = readAirports(
      sparkSession,
      Vector(
        AirportField.Id,
        AirportField.Name,
        AirportField.Country
      )
    )
```

```
    val vertices = airportsDf.rdd.map { r =>
      (r.getLong(0), (r.getString(1), r.getString(2)))
    }

    val edges = readRoutes(
      sparkSession,
        Vector(
          RouteField.SourceAirportId,
          RouteField.DestAirportId
        )
    ).rdd.map { r =>
      spark.graphx.Edge(r.getLong(0), r.getLong(1), ())
    }
    val graph = spark.graphx.Graph(vertices, edges)
    val airportIdWithComponentIdRdd =
graph.connectedComponents().vertices
    showClusters(airportIdWithComponentIdRdd, vertices,
"connectedComponents")
    sparkSession.stop()
  }
  def taskSevenLabelPropagation() = {
    val sparkSession =
spark.sql.SparkSession.builder().appName("local").getOrCreate()
    import sparkSession.implicits._
    val airportsDf = readAirports(
      sparkSession,
      Vector(
        AirportField.Id,
        AirportField.Name,
        AirportField.Country
      )
    )
    val vertices = airportsDf.rdd.map { r =>
      (r.getLong(0), (r.getString(1), r.getString(2)))
    }
    val edges = readRoutes(
      sparkSession,
```

```
        Vector(
          RouteField.SourceAirportId,
          RouteField.DestAirportId
        )
    ).rdd.map { r =>
      spark.graphx.Edge(r.getLong(0), r.getLong(1), ())
    }
    val graph = spark.graphx.Graph(vertices, edges)
    val airportIdWithComponentIdRdd =
spark.graphx.lib.LabelPropagation.run(graph, maxSteps =
100).vertices

    showClusters(airportIdWithComponentIdRdd, vertices,
"LabelPropagation")
    sparkSession.stop()
  }
  def main(args: Array[String]): Unit = {
    // taskOne()
    taskTwoSql()
    // taskTwoGraphFrames()
    // taskThree()
    // taskFive()
    // taskSix()
    // taskSevenLabelPropagation()
    // taskSevenConnectedComponents()
  }
}
```