package main

import (

"encoding/csv"

"fmt"

"log"

"os"

"path/filepath"

"strconv"

"strings"

"sync"

"time"

)

// Función para cargar el archivo CSV

func loadCSV(*filepath* *string*) [][]*string* {

file, err := os.Open(filepath)

if err != nil {

log.Fatal(err)

}

defer file.Close()

reader := csv.NewReader(file)

records, err := reader.ReadAll()

if err != nil {

log.Fatal(err)

}

return records

}

// Función para escribir un archivo CSV

func writeCSV(*filepath* *string*, *data* [][]*string*) {

file, err := os.Create(filepath)

if err != nil {

log.Fatal(err)

}

defer file.Close()

writer := csv.NewWriter(file)

defer writer.Flush()

for \_, record := range data {

err := writer.Write(record)

if err != nil {

log.Fatal(err)

}

}

}

// Función para dividir el archivo en partes

func splitCSV(*records* [][]*string*, *parts* *int*, *outputPath* *string*) {

numRows := len(records) - 1

rowsPerPart := numRows / parts

for i := 0; i < parts; i++ {

startRow := i \* rowsPerPart

endRow := startRow + rowsPerPart

if i == parts-1 {

endRow = numRows

}

part := append([][]*string*{records[0]}, records[startRow+1:endRow+1]...)

writeCSV(filepath.Join(outputPath, fmt.Sprintf("ratings\_part\_%d.csv", i+1)), part)

}

fmt.Printf("El archivo ha sido dividido en %d partes.\n", parts)

}

// Función para contar las calificaciones y sumar ratings por género

func countRatingsByGenre(*ratings* [][]*string*, *movies* [][]*string*, *genreCount* map[*string*]*int*, *genreRatingsSum* map[*string*]*float64*, *wg* \*sync.WaitGroup, *mu* \*sync.Mutex) {

defer wg.Done()

movieMap := make(map[*string*]*string*)

for \_, movie := range movies[1:] {

movieMap[movie[0]] = movie[2] // movieId -> genres

}

for \_, row := range ratings[1:] {

movieId := row[1]

rating, err := strconv.ParseFloat(row[2], 64)

if err != nil {

log.Printf("Error al convertir rating: %v", err)

continue

}

if genres, ok := movieMap[movieId]; ok {

for \_, genre := range strings.Split(genres, "|") {

mu.Lock()

genreCount[genre]++

genreRatingsSum[genre] += rating

mu.Unlock()

}

}

}

}

func main() {

// Tiempo total del programa

startTotal := time.Now()

// Medir tiempo de carga del archivo de calificaciones

start := time.Now()

//ratings := loadCSV("Ratings/ratings.csv")

//fmt.Printf("Tiempo para cargar el archivo de calificaciones: %s\n", time.Since(start))

// Dividir el archivo en partes

//start = time.Now()

N := 10

//outputPath := "Ratings/"

//splitCSV(ratings, N, outputPath)

//fmt.Printf("Tiempo para dividir el archivo en partes: %s\n", time.Since(start))

// Medir tiempo de carga de la lista de películas

start = time.Now()

movies := loadCSV("Ratings/movies.csv")

fmt.Printf("Tiempo para cargar la lista de películas: %s\n", time.Since(start))

// Crear diccionarios para almacenar el conteo y la suma de calificaciones por género

genreCount := make(map[*string*]*int*)

genreRatingsSum := make(map[*string*]*float64*)

// Usar goroutines para procesar en paralelo

var wg sync.WaitGroup

var mu sync.Mutex

start = time.Now()

for i := 1; i <= N; i++ {

wg.Add(1)

go func(*partNum* *int*) {

partFile := fmt.Sprintf("Ratings/ratings\_part\_%d.csv", partNum)

ratingsPart := loadCSV(partFile)

countRatingsByGenre(ratingsPart, movies, genreCount, genreRatingsSum, &wg, &mu)

}(i)

}

wg.Wait()

fmt.Printf("Tiempo para contar y sumar ratings por género: %s\n", time.Since(start))

// Formatear los resultados e imprimir el promedio de calificaciones por género

for genre, count := range genreCount {

averageRating := genreRatingsSum[genre] / *float64*(count)

fmt.Printf("%s: \t %s calificaciones, promedio de rating: %.2f\n", genre, formatWithCommas(count), averageRating)

}

// Tiempo total del programa

fmt.Printf("Tiempo total del programa: %s\n", time.Since(startTotal))

}

// Función para formatear números con comas

func formatWithCommas(*n* *int*) *string* {

str := strconv.Itoa(n)

length := len(str)

if length <= 3 {

return str

}

result := ""

for i, digit := range str {

if i > 0 && (length-i)%3 == 0 {

result += ","

}

result += *string*(digit)

}

return result

}

