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 += ","

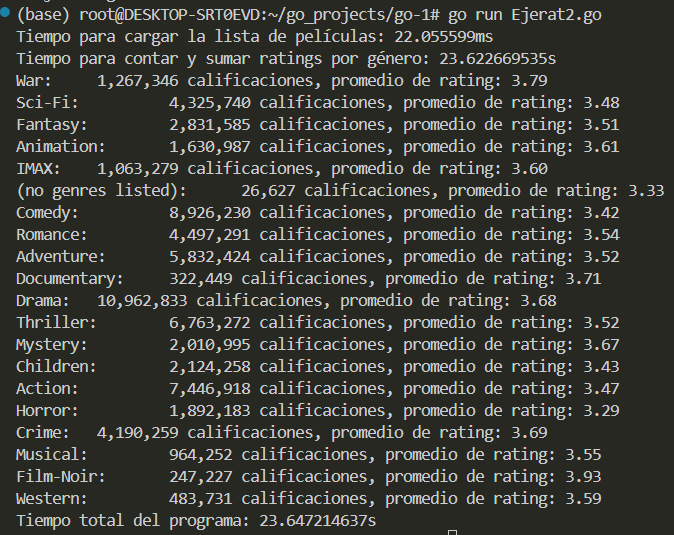
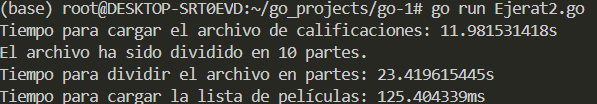
}

result += *string*(digit)

}

return result

}



Julia

Celda 1: Dividir archivo

using Pkg

Pkg.add("CSV")

Pkg.add("DataFrames")

using CSV

using DataFrames

# Cargar el archivo CSV

df = CSV.File("ratings.csv") |> DataFrame

# Definir el número de partes en las que quieres dividir el archivo

N = 10

# Calcular el tamaño de cada parte

num\_rows = nrow(df)

rows\_per\_part = div(num\_rows, N)

# Crear un loop para dividir el DataFrame y guardar cada parte

for i in 0:(N-1)

start\_row = i \* rows\_per\_part + 1

end\_row = (i == N - 1) ? num\_rows : (start\_row + rows\_per\_part - 1)

# Crear un DataFrame para la parte actual

part\_df = df[start\_row:end\_row, :]

# Guardar la parte como un nuevo archivo CSV

CSV.write("ratings\_part\_$(i+1).csv", part\_df)

end

println("El archivo ha sido dividido en $N partes.")

Celda 2

using CSV

using DataFrames

using Base.Threads

using Printf

# Asegúrate de que el número de hilos esté configurado

println("Número de hilos disponibles: ", nthreads())

# Función para cargar los archivos CSV y contar calificaciones por género

function cargar\_y\_contar(i, movies)

ratings = CSV.read("ratings\_part\_" \* string(i) \* ".csv", DataFrame)

return count\_ratings\_by\_genre(ratings, movies)

end

# Función para contar las calificaciones por género

function count\_ratings\_by\_genre(ratings, movies)

# Unir las tablas por 'movieId'

data = innerjoin(ratings, movies, on=:movieId)

# Inicializar un diccionario para contar las calificaciones por género

genre\_count = Dict{String, Int}()

for row in eachrow(data)

genres = split(row.genres, "|") # Dividir los géneros por '|'

for genre in genres

genre\_count[genre] = get(genre\_count, genre, 0) + 1

end

end

return genre\_count

end

# Cargar la lista de películas

movies = CSV.read("movies.csv", DataFrame)

# Arreglo para almacenar los resultados de cada hilo

results = Vector{Dict{String, Int}}(undef, 10)

# Usar Threads.@threads para ejecutar en paralelo

Threads.@threads for i in 1:10

results[i] = cargar\_y\_contar(i, movies)

end

# Combinar los resultados de todos los hilos

final\_result = Dict{String, Int}()

for result in results

for (genre, count) in result

final\_result[genre] = get(final\_result, genre, 0) + count

end

end

function format\_with\_commas(n::Int)

s = string(n)

len = length(s)

# Si la longitud es menor o igual a 3, solo retorna el número

if len <= 3

return s

end

# Crear una cadena con comas

result = ""

for (i, digit) in enumerate(reverse(s))

if i > 1 && (i - 1) % 3 == 0

result \*= ","

end

result \*= digit

end

return reverse(result)

end

# Iterar sobre el diccionario e imprimir el resultado formateado

for (genre, count) in final\_result

formated\_count = format\_with\_commas(count)

println("$genre: \t $formated\_count calificaciones")

end

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media