| EI1024/MT1024 “Programaci´on Concurrente y Paralela” 2022–23  Nombre y apellidos (1): Belén Ariño Bolinches  Nombre y apellidos (2): [Aleix Grau Bas](mailto:al394752@uji.es)  . Tiempo empleado para tareas en casa en formato *h:mm* (obligatorio): 1:30 | Entregable  para  Laboratorio  la07 g |
| --- | --- |

Tema 08. Concurrencia en Colecciones de Java

Tema 09. El Problema de la Coordinaci´on en Java

Se dispone de un programa secuencial que calcula y muestra la palabra m´as usada en un vector de tiras de caracteres, y se desea desarrollar una soluci´on paralela a dicho problema. En esta pr´actica, debes emplear las hebras normales (sin *Thread Pools*) y puedes emplear una distri buci´on c´ıclica del trabajo.

Adem´as, debes ir con cuidado en las versiones concurrentes para evitar que dos hebras intenten a˜nadir la misma palabra al mismo tiempo.

No olvides comprobar que todas las versiones paralelas funcionan correctamente. Para ello debes comparar los resultados (tanto la palabra como su n´umero de veces) de las versiones paralelas con los de la versi´on secuencial. En las comprobaciones es conveniente que emplees el fichero f0.txt, dado que su contenido puede generar m´as errores.

Para realizar una comparaci´on justa, debes emplear en todas las versiones los mismos valores de capacidad inicial y factor de carga. As´ı, deber´ıas usar 1000 como capacidad inicial y 0.75F como factor de carga.

A continuaci´on se muestra el c´odigo secuencial del que se dispone. Como puedes ver, el c´alculo se cuencial se realiza dos veces, pero s´olo se cuenta el tiempo y se muestran resultados para la segunda ejecuci´on. Ello se debe a que la primera ejecuci´on es mucho m´as costosa (dado que debe cargar un mont´on de datos en antememoria), pero, obviamente, solo hace falta realizarlo al principio del progra ma, por lo que no debes repetirlo para cada implementaci´on paralela.

import j a v a . i o . ∗ ;

import j a v a . u t i l . ∗ ;

*// ============================================================================* c l a s s EjemploPalabraMasUsada *{*

*// ============================================================================*

*// −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−* public s t a t i c void main ( S t ri n g a r g s [ ] ) *{*

long t1 , t 2 ;

double t t ;

int numHebras ;

S t ri n g nombreFichero , p al a b r aA c t u al ;

Vector*<*S t ri n g*>* v e c t o r Li n e a s ;

HashMap*<*S t ri n g , I n t e g e r *>* hmCuentaPalabras ;

*// Comprobacion y e x t r a c c i o n de l o s argumen tos de en t r a d a .*

i f ( a r g s . l e n g t h != 2 ) *{*

System . e r r . p r i n t l n ( ”Uso : j a v a programa *<*numHebras*> <*f i c h e r o *>*” ) ;

System . e x i t ( −1 ) ;

*}*

2

try *{*

numHebras = I n t e g e r . p a r s e I n t ( a r g s [ 0 ] ) ;

nombreFichero = a r g s [ 1 ] ;

*}* catch ( NumberFormatException ex ) *{*

numHebras = −1;

nombreFichero = ” ” ;

System . out . p r i n t l n ( ”ERROR: Argumentos nume ric o s i n c o r r e c t o s . ” ) ; System . e x i t ( −1 ) ;

*}*

*// Lec t u r a y c a rg a de l i n e a s en ” v e c t o r L i n e a s ” .*

v e c t o r Li n e a s = r e a d F i l e ( nombreFichero ) ;

System . out . p r i n t l n ( ”Numero de l i n e a s l e i d a s : ” + v e c t o r Li n e a s . s i z e ( ) ) ; System . out . p r i n t l n ( ) ;

*//*

*// Implemen tac ion s e c u e n c i a l s i n t em p o r i z a r .*

*//*

hmCuentaPalabras = new HashMap*<*S t ri n g , I n t e g e r *>*( 1 0 0 0 , 0. 7 5F ) ; for ( int i = 0 ; i *<* v e c t o r Li n e a s . s i z e ( ) ; i++ ) *{*

*// Procesa l a l i n e a ” i ” .*

S t ri n g [ ] p al a b r a s = v e c t o r Li n e a s . g e t ( i ) . s p l i t ( ” *\\*W+” ) ;

for ( int j = 0 ; j *<* p al a b r a s . l e n g t h ; j++ ) *{*

*// Procesa cada p a l a b r a de l a l i n e a ” i ” , s i e s d i s t i n t a de bl a n c o .* p al a b r aA c t u al = p al a b r a s [ j ] . t rim ( ) ;

i f ( p al a b r aA c t u al . l e n g t h ( ) *>* 0 ) *{*

c o n t a b i l i z a P a l a b r a ( hmCuentaPalabras , p al a b r aA c t u al ) ;

*}*

*}*

*}*

*//*

*// Implemen tac ion s e c u e n c i a l .*

*//*

t 1 = System . nanoTime ( ) ;

hmCuentaPalabras = new HashMap*<*S t ri n g , I n t e g e r *>*( 1 0 0 0 , 0. 7 5F ) ; for ( int i = 0 ; i *<* v e c t o r Li n e a s . s i z e ( ) ; i++ ) *{*

*// Procesa l a l i n e a ” i ” .*

S t ri n g [ ] p al a b r a s = v e c t o r Li n e a s . g e t ( i ) . s p l i t ( ” *\\*W+” ) ;

for ( int j = 0 ; j *<* p al a b r a s . l e n g t h ; j++ ) *{*

*// Procesa cada p a l a b r a de l a l i n e a ” i ” , s i e s d i s t i n t a de bl a n c o .* p al a b r aA c t u al = p al a b r a s [ j ] . t rim ( ) ;

i f ( p al a b r aA c t u al . l e n g t h ( ) *>* 0 ) *{*

c o n t a b i l i z a P a l a b r a ( hmCuentaPalabras , p al a b r aA c t u al ) ;

*}*

*}*

*}*

t 2 = System . nanoTime ( ) ;

t t = ( ( double ) ( t 2 − t 1 ) ) / 1. 0 e9 ;

System . out . p r i n t ( ” Implementacion s e c u e n c i a l : ” ) ;

imprimePalabraMasUsadaYVeces ( hmCuentaPalabras ) ;

System . out . p r i n t l n ( ” Tiempo ( s ) : ” + t t ) ;

System . out . p r i n t l n ( ”Num. elem s . t a bl a hash : ” + hmCuentaPalabras . s i z e ( ) ) ; System . out . p r i n t l n ( ) ;

*/*∗

*//*

*// Implemen tac ion p a r a l e l a 1 : Uso de synchron izedMap .*

*//*

*t 1 = System . nanoTime ( ) ;*

*// . . .*

3

*t 2 = System . nanoTime ( ) ;*

*t t = ( ( d o u bl e ) ( t 2 − t 1 ) ) / 1 . 0 e9 ;*

*System . ou t . p r i n t ( ” Implemen tac ion p a r a l e l a 1 : ” ) ;*

*imprimePalabraMasUsadaYVeces ( maCuentaPalabras ) ;*

*System . ou t . p r i n t l n ( ” Tiempo ( s ) : ” + t t + ” , Incremen to ” + . . . ) ; System . ou t . p r i n t l n ( ”Num. elems . t a b l a hash : ” + . . . ) ;*

*System . ou t . p r i n t l n ( ) ;*

*//*

*// Implemen tac ion p a r a l e l a 2 : Uso de H a s h t a bl e .*

*//*

*// . . .*

*//*

*// Implemen tac ion p a r a l e l a 3 : Uso de ConcurrentHashMap*

*//*

*// . . .*

*//*

*// Implemen tac ion p a r a l e l a 4 : Uso de ConcurrentHashMap*

*//*

*// . . .*

*//*

*// Implemen tac ion p a r a l e l a 5 : Uso de ConcurrentHashMap*

*//*

*// . . .*

*//*

*// Implemen tac ion p a r a l e l a 6 : Uso de ConcurrentHashMap*

*//*

*// . . .*

*//*

*// Implemen tac ion p a r a l e l a 7 : Uso de S treams*

*// t 1 = System . nanoTime ( ) ;*

*// Map<S t r i n g , Long> s tC u e n t aP al a b r a s = v e c t o r L i n e a s . p a r a l l e l S t r e a m ( ) // . f i l t e r ( s −> s != n u l l ) // . map( s −> s . s p l i t ( ”\\W+” ) ) // . fl a tM a p ( Arrays : : s tream ) // . map( S t r i n g : : t r im )*

*// . f i l t e r ( s −> ( s . l e n g t h ( ) > 0) ) // . c o l l e c t ( group ingBy ( s −> s , c o u n t i n g ( ) ) ) ; // t 2 = System . nanoTime ( ) ;*

*// . . .*

∗*/*

System . out . p r i n t l n ( ”Fin de programa . ” ) ;

*}*

*// −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−* public s t a t i c Vector*<*S t ri n g*>* r e a d F i l e ( S t ri n g fileN ame ) *{*

Bu f fe redRe ade r br ;

S t ri n g l i n e a ;

Vector*<*S t ri n g*>* data = new Vector*<*S t ri n g *>*( );

try *{*

br = new Bu f fe redRe ade r ( new Fil eR e a d e r ( fileN ame ) ) ;

while ( ( l i n e a = br . r e adLi n e ( ) ) != nu l l ) *{*

*// // System . ou t . p r i n t l n ( ” Le ida l i n e a : ” + l i n e a ) ;*

data . add ( l i n e a ) ;

*}*

4

br . c l o s e ( ) ;

*}* catch ( FileNotFoundException ex ) *{*

ex . p ri n t S t a c kT r a c e ( ) ;

*}* catch ( IOException ex ) *{*

ex . p ri n t S t a c kT r a c e ( ) ;

*}*

return data ;

*}*

*// −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−* public s t a t i c void c o n t a b i l i z a P a l a b r a (

HashMap*<*S t ri n g , I n t e g e r *>* cuen t aP al ab r a s ,

S t ri n g p al a b r a ) *{*

I n t e g e r numVeces = c u e n t aP al a b r a s . g e t ( p al a b r a ) ;

i f ( numVeces != nu l l ) *{*

c u e n t aP al a b r a s . put ( p al ab r a , numVeces+1 ) ;

*}* e l s e *{*

c u e n t aP al a b r a s . put ( p al ab r a , 1 ) ;

*}*

*}*

*// −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−* s t a t i c void imprimePalabraMasUsadaYVeces (

Map*<*S t ri n g , I n t e g e r *>* c u e n t aP al a b r a s ) *{*

Vector*<*Map. Entry*>* l i s t a =

new Vector*<*Map. Entry*>*( c u e n t aP al a b r a s . e n t r y S e t ( ) ) ;

S t ri n g palabraMasUsada = ” ” ;

int numVecesPalabraMasUsada = 0 ;

*// C al c ul a l a p a l a b r a mas usada .*

for ( int i = 0 ; i *<* l i s t a . s i z e ( ) ; i++ ) *{*

S t ri n g p al a b r a = ( S t ri n g ) l i s t a . g e t ( i ) . getKey ( ) ;

int numVeces = ( I n t e g e r ) l i s t a . g e t ( i ) . ge tV alue ( ) ;

i f ( i == 0 ) *{*

palabraMasUsada = p al a b r a ;

numVecesPalabraMasUsada = numVeces ;

*}* e l s e i f ( numVecesPalabraMasUsada *<* numVeces ) *{*

palabraMasUsada = p al a b r a ;

numVecesPalabraMasUsada = numVeces ;

*}*

*}*

*// Imprime r e s u l t a d o .*

System . out . p r i n t ( ” ( Palab ra : ’ ” + palabraMasUsada + ” ’ ” +

” v e c e s : ” + numVecesPalabraMasUsada + ” ) ” ) ;

*}*

*// −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−* s t a t i c void p rin tCuen t aP al ab r a sO rden ad a s (

HashMap*<*S t ri n g , I n t e g e r *>* c u e n t aP al a b r a s ) *{*

int i , numVeces ;

Li s t*<*Map. Entry*>* l i s t = new Vector*<*Map. Entry*>*( c u e n t aP al a b r a s . e n t r y S e t ( ) ) ;

*// Ordena por v a l o r .*

C o l l e c t i o n s . s o r t (

l i s t ,

new Comparator*<*Map. Entry *>*() *{*

public int compare ( Map. Entry e1 , Map. Entry e2 ) *{*

I n t e g e r i 1 = ( I n t e g e r ) e1 . ge tV alue ( ) ;

I n t e g e r i 2 = ( I n t e g e r ) e2 . ge tV alue ( ) ;

return i 2 . compareTo ( i 1 ) ;

*}*

5

*}*

) ;

*// Muestra c o n t e n i d o .*

i = 1 ;

System . out . p r i n t l n ( ” Veces Palab ra ” ) ;

System . out . p r i n t l n ( ”−−−−−−−−−−−−−−−−−” ) ;

for ( Map. Entry e : l i s t ) *{*

numVeces = ( ( I n t e g e r ) e . ge tV alue ( ) ) . i n tV al u e ( ) ;

System . out . p r i n t l n ( i + ” ” + e . getKey ( ) + ” ” + numVeces ) ;

i ++;

*}*

System . out . p r i n t l n ( ”−−−−−−−−−−−−−−−−−” ) ;

*}*

*}*

| 1 |
| --- |

Realiza una implementaci´on paralela con la colecci´on HashMap. Recuerda que esta colecci´on es no sincronizada.

Esta implementaci´on junto a las dos secuenciales se deber´a guardar en el fichero llamado Ejercicio. Para realizar an´alisis de costes equilibrados, debes asegurar que las versiones secuenciales no uti licen m´etodos sincronizados.

En el caso que necesites modificar el m´etodo contabilizaPalabra, crea una copia con otro nombre, para mantener el m´etodo que es utilizado en la versi´on secuencial.

Escribe a continuaci´on el c´odigo que realiza tal tarea:

**la definición de la clase MiHebra1**

Clase hebra

import java.util.HashMap;

import java.util.Vector;

class MiHebra1 extends Thread{

Vector<String> lineas;

int miId;

int numHebras;

Map<String,Integer> mapa;

public MiHebra1(int miId, int numHebras, Vector<String> linea, Map<String,Integer> mapa){

this.miId=miId;

this.numHebras=numHebras;

this.lineas=linea;

this.mapa=mapa;}

public void run() {

for (int i = miId; i < lineas.size(); i += numHebras) { String[] palabras = lineas.get(i).split("\\W+");

for (int j = 0; j < palabras.length; j++) {

String palabraActual = palabras[j].trim();

if (palabraActual.length() > 0) {

EjemploPalabraMasUsada.*contabilizaPalabra1*(mapa,

palabraActual);}}}}}

**El código a incluir en el programa principal que permite gestionar los objetos de esta clase.**

ContabilizaPalabra1

public static void contabilizaPalabra1(

Map<String,Integer> cuentaPalabras,

String palabra ) {

synchronized (cuentaPalabras) {

Integer numVeces = cuentaPalabras.get(palabra);

if (numVeces != null) {

cuentaPalabras.put(palabra, numVeces + 1);

} else {

cuentaPalabras.put(palabra, 1);

}}}

Main

t1 = System.*nanoTime*();

Map<String, Integer> maCuentaPalabras = Collections.*synchronizedMap*(new HashMap<String,Integer>());

MiHebra1[] v = new MiHebra1[numHebras];

for(int i=0;i<numHebras;i++){

v[i] = new MiHebra1(i,numHebras,vectorLineas,maCuentaPalabras); v[i].start();

}

for (int i = 0; i < v.length; i++){

try {

v[i].join();

}catch (InterruptedException e) {

throw new RuntimeException(e);

}}

t2 = System.*nanoTime*();

tt = ( ( double ) ( t2 - t1 ) ) / 1.0e9;

System.*out*.print( "Implementacion paralela 1: " );

*imprimePalabraMasUsadaYVeces*(maCuentaPalabras);

System.*out*.println( " Tiempo(s): " + tt + " , Incremento " +tc/tt); System.*out*.println( "Num. elems. tabla hash: " + maCuentaPalabras.size() );

System.*out*.println();

| 2 |
| --- |

Realiza una implementación paralela con la colección **Hashtable**.

¿Será posible reutilizar la clase MiHebra 1 en este ejercicio? Razona tu respuesta.

Si se puede reutilizar la hebra1 simplmente cambiando los parametros que son HashMap a HashTable

Esta implementaci´on junto a las dos secuenciales se deber´a guardar en el fichero llamado

Ejer 2. Cuando se valide su ejecuci´on, a˜nade el c´odigo correspondiente al fichero Ejercicio.

Escribe a continuaci´on lel c´odigo que realiza tal tarea:

**la definición de la clase MiHebra 2**

import java.util.Hashtable;

import java.util.Vector;

public class MiHebra2 extends Thread{

Vector<String> lineas;

int miId;

int numHebras;

Hashtable<String,Integer> table;

public MiHebra2(int miId, int numHebras, Vector<String> linea, Hashtable<String,Integer> table){

this.miId=miId;

this.numHebras=numHebras;

this.lineas=linea;

this.table=table; }

public void run() {

for (int i = miId; i < lineas.size(); i += numHebras) { String[] palabras = lineas.get(i).split("\\W+");

for (int j = 0; j < palabras.length; j++) {

String palabraActual = palabras[j].trim();

if (palabraActual.length() > 0) {

EjemploPalabraMasUsada.*contabilizaPalabra2*(table,

palabraActual);

}}}}}

**el código a incluir en el programa principal que permite gestionar los objetos de esta clase. MiHebra2**

ContabilizaPalabra2

public static void contabilizaPalabra2(

Hashtable<String,Integer> cuentaPalabras,

String palabra ) {

Integer numVeces = cuentaPalabras.get(palabra);

if (numVeces != null) {

cuentaPalabras.put(palabra, numVeces + 1);

} else {

cuentaPalabras.put(palabra, 1);

}

}

Main

t1 = System.*nanoTime*();

Hashtable<String,Integer> hmCuentaPalabras2 = new

Hashtable<String,Integer>(1000);

MiHebra2 v2[]=new MiHebra2[numHebras];

for(int i=0;i<numHebras;i++){

v2[i]=new MiHebra2(i,numHebras,vectorLineas,hmCuentaPalabras2);

v2[i].start();

}

for (int i = 0; i < v2.length; i++){

try {

v2[i].join();

}catch (InterruptedException e) {

throw new RuntimeException(e);}}

t2 = System.*nanoTime*();

tt = ( ( double ) ( t2 - t1 ) ) / 1.0e9;

System.*out*.print( "Implementacion paralela 2: " );

*imprimePalabraMasUsadaYVeces*(hmCuentaPalabras2 );

System.*out*.println( " Tiempo(s): " + tt + " , Incremento " +tc/tt); System.*out*.println( "Num. elems. tabla hash: " + hmCuentaPalabras2.size() );

System.*out*.println();

| 3 |
| --- |

Realiza una implementación paralela con la colección **ConcurrentHashMap** con un cerrojo adicional, es decir, empleando las clausulas **synchronized o static synchronized**.

¿Será posible reutilizar las clases MiHebra 1 o MiHebra 2 en este ejercicio? Razona tu respuesta. Si, simplemente cambiando los parámetros HashMap a ConcurrentHashMap

Esta implementación junto a las dos secuenciales se debe´a guardar en el fichero llamado Ejer 3. Cuando se valide su ejecuci´on, a˜nade el c´odigo correspondiente al fichero Ejercicio.

Escribe a continuaci´on el c´odigo que realiza tal tarea:

**la definici´on de la clase MiHebra 3**

import java.util.Hashtable;

import java.util.Vector;

import java.util.concurrent.ConcurrentHashMap;

public class MiHebra3 extends Thread{

Vector<String> lineas;

int miId;

int numHebras;

ConcurrentHashMap<String,Integer> table;

public MiHebra3(int miId, int numHebras, Vector<String> linea, ConcurrentHashMap<String,Integer> table){

this.miId=miId;

this.numHebras=numHebras;

this.lineas=linea;

this.table=table;

}

public void run() {

for (int i = miId; i < lineas.size(); i += numHebras) {

String[] palabras = lineas.get(i).split("\\W+");

for (int j = 0; j < palabras.length; j++) {

String palabraActual = palabras[j].trim();

if (palabraActual.length() > 0) {

EjemploPalabraMasUsada.*contabilizaPalabra3*(table,

palabraActual);

}

}

}

}

}

**el c´odigo a incluir en el programa principal que permite gestionar los objetos de esta clase.**

Main

t1 = System.*nanoTime*();

ConcurrentHashMap<String,Integer> hmCuentaPalabras3 = new ConcurrentHashMap<>(1000);

MiHebra3 v3[]=new MiHebra3[numHebras];

for(int i=0;i<numHebras;i++){

v3[i]=new MiHebra3(i,numHebras,vectorLineas,hmCuentaPalabras3); v3[i].start();

}

for (int i = 0; i < v3.length; i++){

try {

v3[i].join();

}catch (InterruptedException e) {

throw new RuntimeException(e);

}

}

t2 = System.*nanoTime*();

tt = ( ( double ) ( t2 - t1 ) ) / 1.0e9;

System.*out*.print( "Implementacion paralela 3: " );

*imprimePalabraMasUsadaYVeces*( hmCuentaPalabras3 );

System.*out*.println( " Tiempo(s): " + tt + " , Incremento " +tc/tt); System.*out*.println( "Num. elems. tabla hash: " +

hmCuentaPalabras3.size() );

System.*out*.println();

| 4 |
| --- |

Realiza una implementaci´on paralela con la colección **ConcurrentHashMap** y sin uso de cerrojos adicionales. En este caso, debes emplear los métodos **putIfAbsent, get y replace** para no tener que usar cerrojos adicionales.

¿Sería posible reutilizar alguna de las clases anteriores en este ejercicio? Razona tu respuesta.

Podriamos reutilizar la clase MiHebra simplemente cambiando el tipo de mapa que usamos

Esta implementaci´on junto a las dos secuenciales se debe´a guardar en el fichero llamado Ejer 4. Cuando se valide su ejecución, a˜nade el c´odigo correspondiente al fichero Ejercicio.

Escribe a continuaci´on el c´odigo que realiza tal tarea:

**la definici´on de la clase MiHebra 4**

Mihebra4

import java.util.Hashtable;

import java.util.Vector;

import java.util.concurrent.ConcurrentHashMap;

public class MiHebra4 extends Thread{

Vector<String> lineas;

int miId;

int numHebras;

ConcurrentHashMap<String,Integer> table;

public MiHebra4(int miId, int numHebras, Vector<String> linea, ConcurrentHashMap<String,Integer> table){

this.miId=miId;

this.numHebras=numHebras;

this.lineas=linea;

this.table=table;

}

public void run() {

for (int i = miId; i < lineas.size(); i += numHebras) {

String[] palabras = lineas.get(i).split("\\W+");

for (int j = 0; j < palabras.length; j++) {

String palabraActual = palabras[j].trim();

if (palabraActual.length() > 0) {

EjemploPalabraMasUsada.contabilizaPalabra4(table, palabraActual);

}

}

}

}

}

**el c´odigo a incluir en el programa principal que permite gestionar los objetos de esta clase.**

Main

t1 = System.*nanoTime*();

ConcurrentHashMap<String,Integer> hmCuentaPalabras4 = new ConcurrentHashMap<>(1000, 0.75F);

MiHebra4 v4[] = new MiHebra4[numHebras];

for(int i = 0;i < numHebras;i++){

v4[i]=new MiHebra4(i,numHebras,vectorLineas,hmCuentaPalabras4);

v4[i].start();

}

for (int i = 0; i < v4.length; i++){

try {

v4[i].join();

}catch (InterruptedException e) {

throw new RuntimeException(e);

}

}

t2 = System.*nanoTime*();

tt = ( ( double ) ( t2 - t1 ) ) / 1.0e9;

System.*out*.print( "Implementacion paralela 4: " );

*imprimePalabraMasUsadaYVeces*( hmCuentaPalabras4 );

System.*out*.println( " Tiempo(s): " + tt + " , Incremento " +tc/tt);

System.*out*.println( "Num. elems. tabla hash: " + hmCuentaPalabras4.size() );

System.*out*.println();

contabilizaPalabra4

public static void contabilizaPalabra4(

ConcurrentHashMap<String,Integer> cuentaPalabras,

String palabra ) {

boolean sustit;

Integer valorIni = cuentaPalabras.putIfAbsent(palabra, 1);

if (valorIni != null){

int valorAct = valorIni;

while (true){

sustit = cuentaPalabras.replace(palabra,valorAct, valorAct + 1);

if (sustit) break;

valorAct = cuentaPalabras.get(palabra);

}

}

}

| ATENCION: Los ejercicios anteriores deben realizarse en casa. Los siguientes, en el aula.´ |
| --- |

| 5 |
| --- |

Realiza una implementación paralela con la colección ConcurrentHashMap y **sin uso de cerrojos adicionales**. En este caso, debes emplear los **métodos putIfAbsent y get, y la clase AtomicInteger.**

¿Ser´ıa posible reutilizar alguna de las clases anteriores en este ejercicio? Razona tu respuesta.

Se puede reutilizar la clase MiHebra

Esta implementaci´on junto a las dos secuenciales se deber´a guardar en el fichero llamado Ejer 5. Cuando se valide su ejecución, a˜nade el código correspondiente al fichero Ejercicio.

Escribe a continuaci ́on el código que realiza tal tarea: la definición de la clase MiHebra 5 y el c´odigo a incluir en el programa principal que permite gestionar los objetos de esta clase.

MiHebra5

import java.util.Hashtable;

import java.util.Vector;

import java.util.concurrent.ConcurrentHashMap;

import java.util.concurrent.atomic.AtomicInteger;

public class MiHebra5 extends Thread{

Vector<String> lineas;

int miId;

int numHebras;

ConcurrentHashMap<String,AtomicInteger> table;

public MiHebra5(int miId, int numHebras, Vector<String> linea, ConcurrentHashMap<String, AtomicInteger> table){

this.miId=miId;

this.numHebras=numHebras;

this.lineas=linea;

this.table=table;

}

public void run() {

for (int i = miId; i < lineas.size(); i += numHebras) {

String[] palabras = lineas.get(i).split("\\W+");

for (int j = 0; j < palabras.length; j++) {

String palabraActual = palabras[j].trim();

if (palabraActual.length() > 0) {

EjemploPalabraMasUsada.*contabilizaPalabra5*(table, palabraActual);

}

}

}

}

}

Main

t1 = System.*nanoTime*();

ConcurrentHashMap<String, AtomicInteger> hmCuentaPalabras5 = new ConcurrentHashMap<>(1000, 0.75F);

MiHebra5 v5[] = new MiHebra5[numHebras];

for(int i = 0;i < numHebras;i++){

v5[i] = new MiHebra5(i,numHebras,vectorLineas,hmCuentaPalabras5);

v5[i].start();

}

for (int i = 0; i < v5.length; i++){

try {

v5[i].join();

}catch (InterruptedException e) {

throw new RuntimeException(e);

}

}

t2 = System.*nanoTime*();

tt = ( ( double ) ( t2 - t1 ) ) / 1.0e9;

System.*out*.print( "Implementacion paralela 5: " );

*imprimePalabraMasUsadaYVeces1*( hmCuentaPalabras5 );

System.*out*.println( " Tiempo(s): " + tt + " , Incremento " +tc/tt);

System.*out*.println( "Num. elems. tabla hash: " + hmCuentaPalabras5.size() );

System.*out*.println();

ContabilizaPalabra5

public static void contabilizaPalabra5(

ConcurrentHashMap<String,AtomicInteger> cuentaPalabras,

String palabra ) {

AtomicInteger a = cuentaPalabras.putIfAbsent(palabra, new AtomicInteger(1));

if ( a != null){

AtomicInteger aux = cuentaPalabras.get(palabra);

aux.incrementAndGet();

cuentaPalabras.put(palabra, aux);

}

}

| 6 |
| --- |

Realiza una implementación paralela idéntica a la del Ejer 5 pero con mayor número de niveles de concurrencia. Emplea 256 niveles.

¿Sería posible reutilizar alguna de las clases anteriores en este ejercicio? Razona tu respuesta.

Reusamos las clases de MiHebra5

Esta implementaci´on junto a las dos secuenciales se deberá guardar en el fichero llamado Ejer 6. Cuando se valide su ejecuci´on, a˜nade el c´odigo correspondiente al fichero Ejercicio.

Escribe a continuaci´on el c´odigo que realiza tal tarea:

**la definici´on de la clase MiHebra 6**

public class MiHebra6 extends Thread{

Vector<String> lineas;

int miId;

int numHebras;

ConcurrentHashMap<String,AtomicInteger> table;

public MiHebra6(int miId, int numHebras, Vector<String> linea, ConcurrentHashMap<String, AtomicInteger> table){

this.miId=miId;

this.numHebras=numHebras;

this.lineas=linea;

this.table=table;

}

public void run() {

for (int i = miId; i < lineas.size(); i += numHebras) {

String[] palabras = lineas.get(i).split("\\W+");

for (int j = 0; j < palabras.length; j++) {

String palabraActual = palabras[j].trim();

if (palabraActual.length() > 0) {

EjemploPalabraMasUsada.*contabilizaPalabra5*(table, palabraActual);

}

}

}

}

}

**el c´odigo a incluir en el programa principal que permite gestionar los objetos de esta clase.**

t1 = System.*nanoTime*();

ConcurrentHashMap<String, AtomicInteger> hmCuentaPalabras6 = new ConcurrentHashMap<>(1000, 0.75F, 256);

MiHebra5 v6[] = new MiHebra5[numHebras];

for(int i = 0;i < numHebras;i++){

v6[i] = new MiHebra5(i,numHebras,vectorLineas,hmCuentaPalabras6);

v6[i].start();

}

for (int i = 0; i < v5.length; i++){

try {

v6[i].join();

}catch (InterruptedException e) {

throw new RuntimeException(e);

}

}

t2 = System.*nanoTime*();

tt = ( ( double ) ( t2 - t1 ) ) / 1.0e9;

System.*out*.print( "Implementacion paralela 6: " );

*imprimePalabraMasUsadaYVeces1*( hmCuentaPalabras6 );

System.*out*.println( " Tiempo(s): " + tt + " , Incremento " +tc/tt);

System.*out*.println( "Num. elems. tabla hash: " + hmCuentaPalabras6.size() );

System.*out*.println();

| 7 |
| --- |

Realiza una implementaci´on paralela basada en Streams.

Esta implementaci´on junto a las dos secuenciales se deber´a guardar en el fichero llamado Ejer 7. Cuando se valide su ejecuci´on, a˜nade el c´odigo correspondiente al fichero Ejercicio.

Seguidamente se muestra el c´odigo implementa esta operaci´on mediante un Parallel Streams

Map*<*S t ri n g , Long*>* s tCuen t aP al ab r a s = v e c t o r Li n e a s . p a r all el S t r e am ( ) . f i l t e r ( s −*>* s != nu l l )

. map( s −*>* s . s p l i t ( ”*\\*W+” ) )

. flatMap ( Arrays : : stream )

. map( S t ri n g : : t rim )

. f i l t e r ( s −*>* ( s . l e n g t h ( ) *>* 0 ) )

. c o l l e c t ( groupingBy ( s −*>* s , c o u n ti n g ( ) ) ) ;

t1 = System.*nanoTime*();

Map<String,Long> stCuentaPalabras = vectorLineas.parallelStream()

.filter( s -> s != null )

.map( s -> s.split( "\\W+" ) )

.flatMap( Arrays::*stream* )

.map( String::trim )

.filter( s -> (s.length() > 0) )

.collect( *groupingBy* (s -> s, *counting*()));

t2 = System.*nanoTime*();

tt = ( ( double ) ( t2 - t1 ) ) / 1.0e9;

System.*out*.print( "Implementacion Strams: " );

*imprimePalabraMasUsadaYVeces3*(stCuentaPalabras);

System.*out*.println( " Tiempo(s): " + tt + " , Incremento " +tc/tt);

System.*out*.println( "Num. elems: " + stCuentaPalabras.size() );

System.*out*.println();

System.*out*.println( "Fin de programa." );

}

static void imprimePalabraMasUsadaYVeces3(Map<String,Long> cuentaPalabras ) {

Vector<Map.Entry> lista = new Vector<Map.Entry>( cuentaPalabras.entrySet() );

String palabraMasUsada = "";

Long numVecesPalabraMasUsada = 0L;

// Calcula la palabra mas usada.

for( int i = 0; i < lista.size(); i++ ) {

String palabra = ( String ) lista.get( i ).getKey();

Long numVeces = ( Long ) lista.get( i ).getValue();

if( i == 0 ) {

palabraMasUsada = palabra;

numVecesPalabraMasUsada = numVeces;

} else if( numVecesPalabraMasUsada < numVeces) {

palabraMasUsada = palabra;

numVecesPalabraMasUsada = numVeces;

}

}

// Imprime resultado.

System.*out*.print( "( Palabra: '" + palabraMasUsada + "' " +

"veces: " + numVecesPalabraMasUsada + " )" );

}

| 8 |
| --- |

Completa la siguiente tabla y justifica los resultados. Obt´en los resultados para 4 hebras en tu ordenador local y los resultados para 16 hebras en patan. Redondea los tiempos dejando s´olo tres decimales y redondea los incrementos dejando dos decimales.

En ambas pruebas debes emplear el fichero f3.txt. Este fichero debe generarse en el script de lanzamiento utilizando el siguiente comando:

c a t f 1 . t x t f 2 . t x t f 1 . t x t f 2 . t x t f 1 . t x t f 2 . t x t f 1 . t x t f 2 . t x t *>* f 3 . t x t y debe ser borrado al final del script.

4 hebras 16 hebras

|  | Tiempo | Incremento | Tiempo | Incremento |
| --- | --- | --- | --- | --- |
| Secuencial | 0.100 | — |  | — |
| Paralela con HashMap | 0.247 | 0.405 |  |  |
| Paralela con Hashtable | 0.184 | 0.546 |  |  |
| Paralela con ConcurrentHashMap y **con cerrojo** adicional | 0.157 | 0.639 |  |  |
| Paralela con ConcurrentHashMap y sin ce rrojo adicional mediante putIfAbsent, get y **replace**  **BUCLE DE REINTENTOS** | 0.139 | 0.721 |  |  |
| Paralela con ConcurrentHashMap y sin cerrojo adiciona, mediante putIfAbsent, get y **AtomicInteger** | 0.141 | 0.708 |  |  |

| Paralela con ConcurrentHashMap y sin ce rrojo adicional mediante putIn Absent, get y AtomicInteger y **con más niveles** | 0.073 | 1.366 |  |
| --- | --- | --- | --- |
| Parallel Stream | 0.190 | 0.528 |  |

Justifica los resultados obtenidos.

La implementación más eficiente será la paralela con ConcurrentHasMap y sin cerrojos, dónde usamos los métodos putIfAbsent, get y un AtomicInteger (en vez del bucle de reintentos), y al tener un nivel de concurrencia mayor permite realizar simultáneamente un mayor número de operaciones. Pero tener un nivel de concurrencia tán elevado supondrá que las operaciones globales tardarán más tiempo.

| 9 |
| --- |

Completa la siguiente tabla y justifica los resultados. Obtén los resultados para 4 hebras en tu ordenador local y los resultados para 16 hebras en patan. Redondea los tiempos dejando s´olo tres decimales y redondea los incrementos dejando dos decimales.

En ambas pruebas debes emplear el fichero f4.txt. Este fichero debe generarse en el script de lanzamiento utilizando el siguiente comando:

c a t f 1 . t x t f 2 . t x t f 1 . t x t f 2 . t x t f 1 . t x t f 2 . t x t f 1 . t x t f 2 . t x t *>* **f 3 . t x t**

c a t f 3 . t x t f 3 . t x t f 3 . t x t f 3 . t x t f 3 . t x t f 3 . t x t f 3 . t x t f 3 . t x t *>* **f 4 . t x t**

y debe ser borrado al final del script.

4 hebras 16 hebras

|  | Tiempo | Incremento | Tiempo | Incremento |
| --- | --- | --- | --- | --- |
| Secuencial | 0.697 | — |  | — |
| Paralela con HashMap | 1.008 | 0.692 |  |  |
| Paralela con Hashtable | 0.809 | 0.862 |  |  |
| Paralela con ConcurrentHashMap y con ce rrojo adicional | 0.706 | 0.988 |  |  |
| Paralela con ConcurrentHashMap y sin ce rrojo adicional mediante putIfAbsent, get y replace | 0.395 | 1.764 |  |  |

| Paralela con ConcurrentHashMap y sin ce rrojo adiciona, mediante putIfAbsent, get y AtomicInteger | 0.351 | 1.985 |  |
| --- | --- | --- | --- |
| Paralela con ConcurrentHashMap y sin ce rrojo adicional mediante putIfAbsent, get y AtomicInteger y con m´as niveles | 0.288 | 2.421 |  |
| Parallel Stream | 0.604 | 1.155 |  |

Justifica los resultados obtenidos.

La implementación más eficiente será la misma que en el caso anterior, pero ahora obtiene un incremento mayor ya que el fichero es más grande

| 1  0 |
| --- |

Completa la siguiente tabla y justifica los resultados. Obt´en los resultados para 4 hebras en tu ordenador local y los resultados para 16 hebras en patan. Redondea los tiempos dejando s´olo tres decimales y redondea los incrementos dejando dos decimales.

En ambas pruebas debes emplear el fichero f5.txt. Este fichero debe generarse en el script de lanzamiento utilizando el siguiente comando:

c a t f 1 . t x t f 2 . t x t f 1 . t x t f 2 . t x t f 1 . t x t f 2 . t x t f 1 . t x t f 2 . t x t *>* **f 3 . t x t**

c a t f 3 . t x t f 3 . t x t f 3 . t x t f 3 . t x t f 3 . t x t f 3 . t x t f 3 . t x t f 3 . t x t *>* **f 4 . t x t**

c a t f 4 . t x t f 4 . t x t f 4 . t x t f 4 . t x t f 4 . t x t f 4 . t x t f 4 . t x t f 4 . t x t *>* f 5 . t x t

y debe ser borrado al final del script.

4 hebras 16 hebras

|  | Tiempo | Incremento Tiempo | Incremento |
| --- | --- | --- | --- |
| Secuencial | 5.109 | — | — |
| Paralela con HashMap | 7.660 | 0.666 |  |
| Paralela con Hashtable | 6.739 | 0.758 |  |

| Paralela con ConcurrentHashMap y con ce rrojo adicional | 5.015 | 1.018 |  |
| --- | --- | --- | --- |
| Paralela con ConcurrentHashMap y sin ce rrojo adicional mediante putIfAbsent, get y replace | 2.190 | 2.513 |  |
| Paralela con ConcurrentHashMap y sin ce rrojo adiciona, mediante putIfAbsent, get y AtomicInteger | 1.929 | 2.648 |  |
| Paralela con ConcurrentHashMap y sin ce rrojo adicional mediante putIfAbsent, get y AtomicInteger y con m´as niveles | 1.916 | 2.666 |  |
| Parallel Stream | 2.002 | 2.551 |  |

Justifica los resultados obtenidos.

La implementación con el ConcurrentHashMap (con nivel de concurrencia 256) sin cerrojo, usando los métodos putIfAbsent y get y empleando un AtomicInteger, seguirá siendo el más eficiente