

Anexos con hilos

Busqueda ABB

```
/**
 * Titulo Busqueda con un arbol binario
 *
 * Este codigo busca un numero dentro de un arbol binario de busqueda
 *
 * @date 9/2021
 * @version 1
 * @author "Los ultimos"
 */
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <pthread.h>
#include "tiempo.h"
//Estructura del arbol binario
struct node{
    int data;
    struct node* left;
    struct node* right;
};

int isempty(struct node*);
int search(struct node*, int);
struct node* push(int, struct node*);
//*****
//VARIABLES GLOBALES
//*****
int NumThreads; //Número de threads
int size; //tamaño de problema
int key; //valor a buscar
struct node** arrayTree;
int keyfound = 0; //bandera que dice si se encontro el elemento

/**
```

```

* Procesar
*
* Esta funcion procesa cada uno de los hilos creados a partir de
* un numthreads dado y se le asigna indice dependiendo del hilo,
* a partir de ahi busca el elemento en el numero de arbol del
* numero de hilo, pero solo ese arbol, si lo encuentra mediante
* keyfound dice que se encontro y ya ningun otro hilo hace mas
* busquedas despues de esto, si no lo encuentra keyfound nunca
* cambia y al terminar el procesamiento de hilos vamos a ver
* que jamas se encontro la llave en ningun hilo ni arbol.
*
* @param un numero de hilo casteado a void*
* @return void*
*/
void* procesar(void* id)
{
    int n_thread = (int)id;
    int inicio,fin,i,a;

    //Revisar la parte de los datos a procesar
    inicio = (n_thread * size) / NumThreads;
    if(n_thread == NumThreads-1)
        fin = size - 1;
    else
        fin = ((n_thread+1) * size) / NumThreads - 1;

    int indexfound = search(arrayTree[n_thread], key);
    if (indexfound != -1) {
        printf(
            "El hilo %d encontro el numero %d.\n",
            n_thread, key);
        keyfound = 1;
    }
}
/**
* isempty
*

```

```

* Evalua un arbol binario y devuelve si este es vacio o no.
*
* Esta funcion compara si el apuntador que le estamos pasando
* es vacio, comparando este con nulo, si es asi, devuelve un 1
* que es equivalente a verdadero, si no es asi, o sea el
* apuntador no es nulo, devuelve un 0, indicando que no es vacio
*
* @param un arbol binario.
* @return 1 si el arbol es vacio, 0 si no lo es.
*/
int isempty(struct node* root){return root == NULL;}

/**
 * push
 *
 * Inserta un elemento en el arbol binario.
 *
 * Esta funcion recibe un arbol binario y un elemento y busca
 * si es vacio el arbol, si lo es lo inserta, si no va comparando
 * con el elemento del nodo hasta ver donde lo debe de insertar
 * dependiendo si es mas grande o mas pequeño en relacion a los
 * elementos de los nodos, de esta manera los elementos mas pequeños
 * van en el subarbol izquierdo, y los mas grandes en el derecho.
 *
 * @param un arbol binario y el elemento a insertar
 * @return un arbol binario con el nuevo elemento insertado.
 */
struct node* push(int dataToInsert, struct node* root)
{
    if (isempty(root)) {
        struct node* new_node = malloc(sizeof(struct node));
        new_node->data = dataToInsert;
        new_node->left = NULL;
        new_node->right = NULL;
        return new_node;
    }
    else if(dataToInsert < root->data)
        root->left = push(dataToInsert, root->left);

```

```

        else
            root->right = push(dataToInsert, root->right);
        return root;
    }

    /**
     * search
     *
     * esta funcion busca un numero en un arbol binario
     *
     * va comparando el elemento de la raiz con el numero a buscar
     * y si es mas grande el nodo entonces recorre a su subarbol
     * izquierdo y sino al derecho. asi hasta encontrar el numero
     * o llegar a un nodo vacio, al final nos dice si el numero
     * se encontro o no. si keyfound esta en 1, ni busca porque
     * ese numero ya se encontro.
     *
     * @param un arbol binario y la llave a buscar
     * @return void
     */
    int search(struct node* root, int key)
    {
        if(keyfound == 1)
            return -1;
        struct node* aux = root;
        while(!isempty(aux) && key != aux->data)
            aux = (key < aux->data) ? aux->left : aux->right;
        if(isempty(aux))
            return -1;
        else
            return 1;
    }

    int main(int argc, char* argv[]){
        double utime0, stime0, wtime0, utime1, stime1, wtime1;
        size = atoi(argv[1]);
        key = atoi(argv[2]);
        NumThreads = atoi(argv[3]);
    }

```

```

arrayTree = (struct node **)malloc(NumThreads * sizeof(struct node));
int i, element, j;
for (j = 0; j < NumThreads; j++) {
    for (i = 0; i < size / NumThreads; i++) {
        scanf("%d", &element);
        arrayTree[j] = push(element, arrayTree[j]);
    }
}

printf("Busqueda arbol binario (key:%d size:%d threads:%d)\n\n", key, size, NumThreads);
//*****
// Iniciar el conteo del tiempo para las evaluaciones de rendimiento
//*****
uswtime(&utime0, &stime0, &wtime0);
pthread_t *thread = malloc(NumThreads * sizeof(pthread_t));
//*****
// Procesar desde cada hilo "procesar"
//*****
// Crear los threads con el comportamiento "segmentar"
for (i = 1; i < NumThreads; i++) {
    if (pthread_create(&thread[i], NULL, procesar, (void *)i) != 0) {
        perror("El thread no pudo crearse");
        exit(-1);
    }
}

// El main ejecuta el thread 0
procesar(0);
// Esperar a que terminen los threads (Saludar)
for (i = 1; i < NumThreads; i++)
    pthread_join(thread[i], NULL);

if (keyfound == 0)
    printf("Ningun hilo encontro el numero %d en el arreglo.\n", key);
//*****
// Evaluar los tiempos de ejecución
//*****
uswtime(&utime1, &stime1, &wtime1);
// Cálculo del tiempo de ejecución del programa
printf("Tiempo real: %.10e s\n", wtime1 - wtime0);

```

```
    return 0;  
}
```

Busqueda binaria

```
/**
 * Titutlo Busqueda binaria
 *
 * este codigo busca un elemento en un arreglo usando el algoritmo de busqueda b
 *
 * @date 9/2021
 * @version 1
 * @author "Los ultimos"
 */

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include "tiempo.h"

int binaria(int, int);
//*****
//VARIABLES GLOBALES
//*****
int NumThreads;      //Número de threads
int size; //tamaño de problema
int key; //valor a buscar
int* array; // arreglo a manejar
int keyfound = 0; //bandera que dice si se encontro el elemento
/**
 * Procesar
 *
 * Esta funcion procesa cada uno de los hilos creados a partir
 * de un numthreads dado y se le asigna cierto rango dependiendo
 * del hilo, a partir de ahi busca el elemento en el arreglo
 * usando el algoritmo dado pero solo busca en ese rango que esta
 * dado por limite inferior y superior, si lo encuentra nos va a
 * imprimir que hilo se encontro y en que posicion y keyfound se
 * pone en 1 indicando que ya no se hagan mas busquedas, si no,
 * la bandera de keyfound nunca va a cambiar y al terminar el
 * procesamiento de hilos vamos a ver que jamas se encontro la
```

```

    * llave en ningun hilo.
    *
    * @param un numero de hilo casteado a void*
    * @return void*
    */
void* procesar(void* id)
{
    int n_thread = (int)id;
    int inicio,fin,i,a;

    //Revisar la parte de los datos a procesar
    inicio = (n_thread * size) / NumThreads;
    if(n_thread == NumThreads-1)
        fin = size-1;
    else
        fin = ((n_thread+1) * size) / NumThreads - 1;

    int indexfound = binaria(inicio, fin);
    if (indexfound != -1) {
        printf("El hilo %d encontro el numero %d en la posicion %d.\n", n_thread, keyfound, indexfound);
        keyfound = 1;
    }
}

/**
 * binaria
 *
 * Realiza el algoritmo de busqueda binaria sobre un arreglo ordenado
 *
 * Este algoritmo parte en mitades el arreglo ordenado, de
 * tal manera que con cada comparacion elimina una mitad del
 * arreglo, haciendo asi la busqueda mas rapida, va calculando
 * la mitad entre 2 limites del arreglo y si la llave es menor,
 * recorre el limite superior, si es mayor entonces recorre el
 * limite inferior y si es igual devuelve esa posicion que es
 * la mitad entre ambos limites, al final si el limite inferior
 * es mayor al superior o viceversa quiere decir que ya analizo
 * todas las posiciones posibles no encontro la llave, por lo
 * tanto el numero no esta en el arreglo.

```



```

*
* @param un limite inferior y un limite superior del arreglo a buscar
* @return la posicion donde se encontro el numero o si
* no se encontro un -1
*/
int binaria(int l, int r)
{
    if(keyfound == 1)
        return -1;
    int h;
    while(l <= r)
    {
        h = l + (r - l)/2;
        if(key < array[h])
            r = h - 1;
        else if(key > array[h])
            l = h + 1;
        else
            return h+1;
    }
    return -1;
}

int main(int argc, char *argv[])
{
    double utime0, stime0, wtime0, utime1, stime1, wtime1;
    size = atoi(argv[1]);
    key = atoi(argv[2]);
    NumThreads = atoi(argv[3]);
    array = (int *)malloc(size * sizeof(int));
    int i;
    for (i = 0; i < size; i++)
        scanf("%d", array+i);
    printf("Busqueda binaria (key:%d size:%d threads:%d).\n\n",
        key, size, NumThreads);

    //*****
    // Iniciar el conteo del tiempo para las evaluaciones de rendimiento
    //*****
    uswtime(&utime0, &stime0, &wtime0);

```

```

pthread_t *thread = malloc(NumThreads * sizeof(pthread_t));
//*****
// Procesar desde cada hilo "procesar"
//*****
// Crear los threads con el comportamiento "segmentar"
for (i = 1; i < NumThreads; i++) {
    if (pthread_create(&thread[i], NULL, procesar, (void *)i) != 0) {
        perror("El thread no pudo crearse");
        exit(-1);
    }
}
// El main ejecuta el thread 0
procesar(0);
// Esperar a que terminen los threads (Saludar)
for (i = 1; i < NumThreads; i++)
    pthread_join(thread[i], NULL);

if (keyfound == 0)
    printf("Ningun hilo encontro el numero %d en el arreglo.\n", key);
//*****
// Evaluar los tiempos de ejecución
//*****
uswtime(&utime1, &stime1, &wtime1);
// Cálculo del tiempo de ejecución del programa
printf("Tiempo real: %.10e s\n", wtime1 - wtime0);
free(array);
return 0;
}

```

Busqueda exponencial

```
/**
 * Titutlo Busqueda lineal
 *
 * este codigo busca un elemento en un arreglo usando
 * el algoritmo de busqueda lineal
 *
 * @date 9/2021
 * @version 1
 * @author "Los ultimos"
 */

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include "tiempo.h"

int binaria(int, int);
int exponencial(int, int);
int min (int x, int y) {return (x <= y) ? x : y;}
//*****
//VARIABLES GLOBALES
//*****
int NumThreads; //Número de threads
int size; //tamaño de problema
int key; //valor a buscar
int* array; // arreglo a manejar
int keyfound = 0; //bandera que dice si se encontro el elemento

/**
 * Procesar
 *
 * Esta funcion procesa cada uno de los hilos creados a partir
 * de un numthreads dado y se le asigna cierto rango dependiendo
 * del hilo, a partir de ahi busca el elemento en el arreglo
 * usando el algoritmo dado pero solo busca en ese rango que esta
 * dado por limite inferior y superior, si lo encuentra nos va a
```

```

* imprimir que hilo se encontro y en que posicion y keyfound se
* pone en 1 indicando que ya no se hagan mas busquedas, si no,
* la bandera de keyfound nunca va a cambiar y al terminar el
* procesamiento de hilos vamos a ver que jamas se encontro la
* llave en ningun hilo.
*
* @param un numero de hilo casteado a void*
* @return void*
*/
void* procesar(void* id)
{
    int n_thread = (int)id;
    int inicio,fin,i,a;

    //Revisar la parte de los datos a procesar
    inicio = (n_thread * size) / NumThreads;
    if(n_thread == NumThreads-1)
        fin = size-1;
    else
        fin = ((n_thread+1) * size) / NumThreads - 1;

    int indexfound = exponencial(inicio, fin);
    if (indexfound != -1) {
        printf("El hilo %d encontro el numero %d en la posicion %d.\n",
            n_thread, key, indexfound);
        keyfound = 1;
    }
}

/**
* binaria
*
* Realiza el algoritmo de busqueda binaria sobre un arreglo ordenado
*
* Este algoritmo parte en mitades el arreglo ordenado, de
* tal manera que con cada comparacion elimina una mitad del
* arreglo, haciendo asi la busqueda mas rapida, va calculando
* la mitad entre 2 limites del arreglo y si la llave es menor,
* recorre el limite superior, si es mayor entonces recorre el

```

```

* limite inferior y si es igual devuelve esa posicion que es
* la mitad entre ambos limites, al final si el limite inferior
* es mayor al superior o viceversa quiere decir que ya analizo
* todas las posiciones posibles no encontro la llave, por lo
* tanto el numero no esta en el arreglo.
*
* @param un limite inferior y un limite superior del arreglo a buscar
* @return la posicion donde se encontro el numero o si
* no se encontro un -1
*/
int binaria(int l, int r)
{
    int h;
    while(l <= r)
    {
        h = l + (r - l)/2;
        if(key < array[h])
            r = h - 1;
        else if(key > array[h])
            l = h + 1;
        else
            return h+1;
    }
    return -1;
}

/**
* exponencial
*
* Realiza el algoritmo de busqueda exponencial sobre un arreglo ordenado
*
* Este algoritmo recibe un limite inferior y revisa si el numero
* esta en ese indice si no lo esta entonces se le asigna una
* variable de indice que es el limite inferior +1 y el indice
* aumenta a razon de potencias de 2, o sea  $2^n + \text{limite inferior}$ 
* para conservar su aumento de potencia de 2, checa si el elemento
* en ese indice es menor a la llave y sigue aumentando siempre y
* cuando estemos dentro del limite, una vez que el numero del
* indice es mayor a la llave o rebasamos el indice permitido,

```

```

* aplica una busqueda binaria a solo esa seccion con parametros
* del i antes de su ultimo aumento y el minimo del limite superior
* y la i en la que vamos y obtenemos el indice.
*
* @param un limite inferior y un limite superior
* @return una busqueda binaria entre el i/2 y el minimo entre i
* y el limite superior
*/
int exponencial(int l, int r)
{
    if(keyfound == 1)
        return -1;
    if(array[l] == key)
        return l+1;
    int exponent2 = 1;
    int i = exponent2 + 1;
    while(i <= r && array[i] < key){
        exponent2 *= 2;
        i = exponent2 + 1;
    }
    return binaria(l+exponent2/2, min(i, r));
}
int main(int argc, char *argv[])
{
    double utime0, stime0, wtime0, utime1, stime1, wtime1;
    size = atoi(argv[1]);
    key = atoi(argv[2]);
    NumThreads = atoi(argv[3]);
    array = (int *)malloc(size * sizeof(int));
    int i;
    for (i = 0; i < size; i++)
        scanf("%d", array+i);
    printf("Busqueda exponencial (key:%d size:%d threads:%d).\n\n",
        key, size, NumThreads);
    //*****
    //Iniciar el conteo del tiempo para las evaluaciones de rendimiento
    //*****
    uswtime(&utime0, &stime0, &wtime0);

```

```

pthread_t* thread = malloc(NumThreads * sizeof(pthread_t));
//*****
//Procesar desde cada hilo "procesar"
//*****
//Crear los threads con el comportamiento "segmentar"
for (i = 1; i < NumThreads; i++)
{
    if (pthread_create (&thread[i], NULL, procesar,(void*)i) != 0 )
    {
        perror("El thread no pudo crearse");
        exit(-1);
    }
}
//El main ejecuta el thread 0
procesar(0);
//Esperar a que terminen los threads (Saludar)
for (i = 1; i < NumThreads; i++) pthread_join (thread[i], NULL);

if(keyfound == 0)
    printf("Ningun hilo encontro el numero %d en el arreglo.\n", key);
//*****
// Evaluar los tiempos de ejecución
//*****
uswtime(&utime1, &stime1, &wtime1);
// Cálculo del tiempo de ejecución del programa
printf("Tiempo real: %.10e s\n", wtime1 - wtime0);
free(array);
return 0;
}

```

Busqueda de Fibonacci

```
/**
 * Titutlo Busqueda de fibonacci
 *
 * este codigo busca un elemento en un arreglo usando el algoritmo de busqueda d
 *
 * @date 9/2021
 * @version 1
 * @author "Los ultimos"
 */
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include "tiempo.h"

//funcion que dice cual numero es menor
int min (int x, int y);
//funcion que buscar un numero en un arreglo
int busquedafibonacci (int, int);
//*****
//VARIABLES GLOBALES
//*****

int NumThreads;          //Número de threads
int size; //tamaño de problema
int key; //valor a buscar
int* arr; // arreglo a manejar
int keyfound = 0; //bandera que dice si se encontro el elemento

/**
 * Procesar
 *
 * Esta funcion procesa cada uno de los hilos creados a partir
 * de un numthreads dado y se le asigna cierto rango dependiendo
 * del hilo, a partir de ahi busca el elemento en el arreglo
 * usando el algoritmo dado pero solo busca en ese rango que esta
 * dado por limite inferior y superior, si lo encuentra nos va a
```



```

    * imprimir que hilo se encontro y en que posicion y keyfound se
    * pone en 1 indicando que ya no se hagan mas busquedas, si no,
    * la bandera de keyfound nunca va a cambiar y al terminar el
    * procesamiento de hilos vamos a ver que jamas se encontro la
    * llave en ningun hilo.
    *
    * @param un numero de hilo casteado a void*
    * @return void*
    */
void* procesar(void* id)
{
    int n_thread = (int)id;
    int inicio,fin,i,a;

    //Revisar la parte de los datos a procesar
    inicio = (n_thread * size) / NumThreads;
    if(n_thread == NumThreads-1)
        fin = size - 1;
    else
        fin = ((n_thread+1) * size) / NumThreads - 1;

    int indexfound = busquedafibonacci(inicio, fin);
    if (indexfound != -1) {
        printf(
            "El hilo %d encontro el numero %d en la posicion %d.\n",
            n_thread, key, indexfound);
        keyfound = 1;
    }
}

int main(int argc, char* argv[]) {
    double utime0, stime0, wtime0, utime1, stime1, wtime1;
    size = atoi(argv[1]);
    key = atoi(argv[2]);
    NumThreads = atoi(argv[3]);
    arr = (int *)malloc(size * sizeof(int));
    int i;
    for (i = 0; i < size; i++)
        scanf("%d", arr+i);
}

```

```

printf("Busqueda de fibonacci (key:%d size:%d threads:%d):\n\n ",
      key, size, NumThreads);

//*****
// Iniciar el conteo del tiempo para las evaluaciones de rendimiento
//*****
uswtime(&utime0, &stime0, &wtime0);
pthread_t *thread = malloc(NumThreads * sizeof(pthread_t));
//*****
// Procesar desde cada hilo "procesar"
//*****
// Crear los threads con el comportamiento "segmentar"
for (i = 1; i < NumThreads; i++) {
    if (pthread_create(&thread[i], NULL, procesar, (void *)i) != 0) {
        perror("El thread no pudo crearse");
        exit(-1);
    }
}
// El main ejecuta el thread 0
procesar(0);
// Esperar a que terminen los threads (Saludar)
for (i = 1; i < NumThreads; i++)
    pthread_join(thread[i], NULL);

if(keyfound == 0)
    printf("Ningun hilo encontro el numero %d en el arreglo.\n", key);
//*****
// Evaluar los tiempos de ejecución
//*****
uswtime(&utime1, &stime1, &wtime1);
// Cálculo del tiempo de ejecución del programa
uswtime(&utime1, &stime1, &wtime1);
// Cálculo del tiempo de ejecución del programa
printf("Tiempo real: %.10e s\n", wtime1 - wtime0);
free(arr);
return 0;
}
/**

```

```

* min
*
* Esta funcion dados dos numeros, te regresa el menor de ellos
* haciendo una comparacion.
*
* @param un entero x y un entero y.
* @return el menor de ambos enteros
*/
int min (int x, int y) {return (x <= y) ? x : y;}
/**
* busqueda fibonacci
*
* Entra n el cual buscara dos numero de la serie de fibonacci
* que mas cerca a n y se sumaran, con ese numero entrara al
* segundo while hasta que la suma de los dos numeros de la
* serie de fibonacci sean menor a 1, una vez entrando al while
* se define i que se define entre la suma del rango +auxfm2 o
* n-1 el que sea menor es el que i tendra su valor despues entra
* a un if si el donde compara la posicion i con el numero a
* buscar, si es menor i a x se dezpalaan los valores y se
* restan al numero de la serie fibonacci. si no entra al if
* esta otra comparacion donde se compara que el numero de
* la posicion i sea mayor a x y dezplaza los valors y resta
* de los dos numeros de la serie fibonacci y si no entra a
* este if es por que el numero a comparar es el numero buscado
* por lo cual retorna i que es la posicion. si el numero a
* buscar no esta dentro del rango de limite inferior y superior,
* regresa que no lo encontro, esto lo sabemos porque tenemos un
* arreglo ordenado.
*
* @param un limite inferior y un limite superior del arreglo a buscar
* @return la posicion donde se encontro el numero o si no se
* encontro un -1
*/
int busquedafibonacci (int l, int r)
{
    if(keyfound == 1)
        return -1;

```

```

if(key < arr[l] || key > arr[r])
    return -1;
int auxf2,auxf1,auxfm;//auxf2 tomara el valor de n-1 y auxf1 n-2 y auxfm alme
int rango,i;
auxf2 = 0;
auxf1 = 1;
auxfm= auxf2 + auxf1;

while(auxfm < r){ //se inicia ciclo hasta que auxfm sea menor a n
    auxf2 = auxf1;
    auxf1 = auxfm;
    auxfm= auxf2 + auxf1;
}

rango = -1;// rango toma el valor de -1
//el ciclo entra hasta que ya no queden numeros a buscar ,compararemos el indice
while (auxfm > 1) {
    // verifica si auxfm2 es no es vacio
    i = min(rango + auxf2, r);

    //Si numbuscar es mayor que el valor de auxfm2 el arreglo se desplaza
    if (arr [i] < key) {
        auxfm = auxf1;
        auxf1 = auxf2;
        auxf2 = auxfm- auxf1;
        rango = i;
    }
    //en cambio si es mayor que el valor en arr[auxfm2], desplazamos el ar
    else if (arr [i] > key) {
        auxfm = auxf2;
        auxf1 = auxf1 - auxf2;
        auxf2 = auxfm- auxf1;
    }
}

// si el elemnto es encontrado se retorna la posicion i
else
    return i+1;
}

```

```
//se comprara el ultimo elemento con numbusc
    if (auxf1 && arr [rango + 1] == key){
        return rango + 1;
    }

//si no se encuentra se retorna -1
    return -1;
}
```

Busqueda lineal

```
/**
 * Titutlo Busqueda lineal
 *
 * este codigo busca un elemento en un arreglo usando el algoritmo de busqueda l
 *
 * @date 9/2021
 * @version 1
 * @author "Los ultimos"
 */

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include "tiempo.h"

int busquedaLineal(int, int);
//*****
//VARIABLES GLOBALES
//*****
int NumThreads;      //Número de threads
int size; //tamaño de problema
int key; //valor a buscar
int* array; // arreglo a manejar
int keyfound = 0; //bandera que dice si se encontro el elemento

/**
 * Procesar
 *
 * Esta funcion procesa cada uno de los hilos creados a partir
 * de un numthreads dado y se le asigna cierto rango dependiendo
 * del hilo, a partir de ahi busca el elemento en el arreglo
 * usando el algoritmo dado pero solo busca en ese rango que esta
 * dado por limite inferior y superior, si lo encuentra nos va a
 * imprimir que hilo se encontro y en que posicion y keyfound se
 * pone en 1 indicando que ya no se hagan mas busquedas, si no,
 * la bandera de keyfound nunca va a cambiar y al terminar el
```

```

    * procesamiento de hilos vamos a ver que jamas se encontro la
    * llave en ningun hilo.
    *
    * @param un numero de hilo casteado a void*
    * @return void*
    */
void* procesar(void* id)
{
    int n_thread = (int)id;
    int inicio,fin,i,a;

    //Revisar la parte de los datos a procesar
    inicio = (n_thread * size) / NumThreads;
    if(n_thread == NumThreads-1)
        fin = size - 1;
    else
        fin = ((n_thread+1) * size) / NumThreads - 1;
    int indexfound = busquedaLineal(inicio, fin);
    if (indexfound != -1) {
        printf(
            "El hilo %d encontro el numero %d en la posicion %d.\n",
            n_thread, key, indexfound);
        keyfound = 1;
    }
}
/**
 *
 * busquedaLineal
 *
 * Esta funcion busca un numero dentro de una lista.
 *
 * Se ingresa la posicion inicial del arreglo y su tamaño, compara cada elemento
 * para verificar si existe el numero a buscar, en caso de que el numero si est
 * posicion mas 1 donde se encuentra el numero, en caso contrario regresa un -1.
 *
 * @param posición inicial y el limite superior del pedazo de arreglo a buscar
 * @return la posicion donde se encontro el numero o si no se encontro un -1
 */

```

```

int busquedaLineal(int l, int r)
{
    if(keyfound == 1)
        return -1;
    int i;
    for (i = l; i <= r; i++)
        if (array[i] == key)
            return i+1;
    return -1;
}

int main(int argc, char *argv[])
{
    double utime0, stime0, wtime0, utime1, stime1, wtime1;
    size = atoi(argv[1]);
    key = atoi(argv[2]);
    NumThreads = atoi(argv[3]);
    array = (int *)malloc(size * sizeof(int));
    int i;
    for (i = 0; i < size; i++)
        scanf("%d", array+i);
    printf("Busqueda lineal (key:%d size:%d threads:%d).\n\n", key, size,
        NumThreads);

    //*****
    // Iniciar el conteo del tiempo para las evaluaciones de rendimiento
    //*****

    uswtime(&utime0, &stime0, &wtime0);
    pthread_t *thread = malloc(NumThreads * sizeof(pthread_t));
    //*****
    // Procesar desde cada hilo "procesar"
    //*****
    // Crear los threads con el comportamiento "segmentar"
    for (i = 1; i < NumThreads; i++) {
        if (pthread_create(&thread[i], NULL, procesar, (void *)i) != 0) {
            perror("El thread no pudo crearse");
            exit(-1);
        }
    }
}

```



```

    // El main ejecuta el thread 0
    procesar(0);
    // Esperar a que terminen los threads (Saludar)
    for (i = 1; i < NumThreads; i++)
        pthread_join(thread[i], NULL);
    //*****
    // Evaluar los tiempos de ejecución
    //*****
    uswtime(&utime1, &stime1, &wtime1);
    // Cálculo del tiempo de ejecución del programa
    printf("Tiempo real: %.10e s\n", wtime1 - wtime0);
    free(array);
    return 0;
}

```