Anexos sin hilos

Busqueda ABB

```c
/**
 * Titulo Busqueda con un arbol binario
 *
 * Este codigo busca un numero dentro de un arbol binario de busqeda
 *
 * @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*);
void search(struct node*, int);
struct node* push(int,struct node*);

/**
 * 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;
}
```

```c
/**
 * 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.
 *
 * @param un arbol binario y la llave a buscar
 * @return void
 */
void search(struct node* root, int key)
{
    struct node* aux = root;
    while(!isempty(aux) && key != aux->data)
        aux = (key < aux->data) ? aux->left : aux->right;
    if(isempty(aux))
        printf("El elemento %d no se encontro en el arbol", key);
    else
        printf("El elemento %d se encontro en el arbol", key);
}

int main(int argc, char* argv[]){
    double utime0, stime0, wtime0, utime1, stime1, wtime1;
    int size = atoi(argv[1]);
    int key = atoi(argv[2]);
    struct node *root = NULL; // Inicializar el arbol binario
    int i, element;
    // Insertar los n de numeros en el arbol
    for (i = 0; i < size; i++) {
        scanf("%d", &element);
        root = push(element, root);
    }
    printf("Busqueda arbol binario (key:%d size:%d): ", key, size);
    //*****************************************************************
```

```
    // Iniciar el conteo del tiempo para las evaluaciones de rendimiento
    //*************************************************************;
    uswtime(&utime0, &stime0, &wtime0);
    //*************************************************************
    // Evaluar los tiempos de ejecución
    //*************************************************************
    search(root, key);
    uswtime(&utime1, &stime1, &wtime1);
    // Cálculo del tiempo de ejecución del programa
    printf(" %.10e s\n", wtime1 - wtime0);
    return 0;
}
```

Busqueda binaria

```c
/**
 * Titutlo Busqueda binaria
 *
 * este codigo busca un elemento en un arreglo usando el
 * algoritmo de busqueda binaria
 *
 * @date 9/2021
 * @version 1
 * @author "Los ultimos"
 */
#include <stdio.h>
#include <stdlib.h>
#include "tiempo.h"
int binaria(int [], int, int);
/**
 * 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 que
 * es mitad - 1, si es mayor entonces recorre el limite inferior que
 * es mitad + 1 y si es igual devuelve esa posicion que es la mitad
 * entre ambos limites + 1, al final si el limite inferior es mayor
 * al superior o viceversa quiere decir que ya analizo todas las
 * posiciones posibles y no encontro la llave, por lo tanto el
 * numero no esta en el arreglo.
 *
 * @param un arreglo ordenado su tamaño y la llave a buscar
 * @return la posicion donde se encontro el numero o si no se encontro un -1
*/
int binaria(int array[], int size, int key){
    int l = 0, r = size - 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[]){
    int i;
    double utime0, stime0, wtime0, utime1, stime1, wtime1;
    int size = atoi(argv[1]);
    int key = atoi(argv[2]);
    int *array = (int *)malloc(size * sizeof(int));
    for (i = 0; i < size; i++)
        scanf("%d", array + i);
    printf("Busqueda binaria (key:%d size:%d).\n\n ", key, size);
    //*****************************************************************
    // Iniciar el conteo del tiempo para las evaluaciones de rendimiento
    //*****************************************************************
    uswtime(&utime0, &stime0, &wtime0);
    //*****************************************************************
    // Evaluar los tiempos de ejecución
    //*****************************************************************
    int index = binaria(array, size, key);
    if (index != 1)
        printf("El numero %d se encontro en la posicion %d.\n", key, index);
    else
        printf("El numero %d no existe en el arreglo.\n", key);
    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

```c
/**
 * 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 "tiempo.h"

int binaria(int [],int, int, int);
int exponencial(int [], int, int);
int min (int x, int y) {return (x <= y) ? x : y;}
/**
 * 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 que
 * es mitad - 1, si es mayor entonces recorre el limite inferior que
 * es mitad + 1 y si es igual devuelve esa posicion +1 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 y no encontro la llave, por lo tanto el
 * numero no esta en el arreglo.
 *
 * @param un arreglo ordenado un limite inferior su tamaño y la llave a buscar
 * @return la posicion donde se encontro el numero o si no se encontro un -1
*/
```

```
int binaria(int array[],int l, int size, int key)
{
    int r = size - 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;
}
/**
 * exponencial
 *
 * Realiza el algoritmo de busqueda exponencial sobre un arreglo ordenado
 *
 * Este algoritmo revisa si el numero esta en la posicion 0 si no lo
 * esta entonces se le asigna una variable de indice que es 1 y el
 * indice aumenta a razon de potencias de 2, o sea 2^n checa si el
 * elemento en ese indice es menor a la llave y sigue aumentando
 * siempre y cuando estemos dentro del limite de tamaño, una vez
 * que el numero del indice es mayor a la llave o rebasamos el
 * tamaño permitido, aplica una binaria a solo esa seccion con parametros
 * del i antes de su ultimo aumento y el minimo del tamaño y la i en
 * la que vamos y obtenemos el indice.
 *
 * @param un arreglo ordenado su tamaño y la llave a buscar
 * @return una busqueda binaria entre i/2 y el minimo entre i y el size
*/
int exponencial(int array[], int size, int key)
{
    if(array[0] == key)
        return 0;
```

```
    int i = 1;
    while(i < size && array[i] < key)
        i *= 2;
    return binaria(array, i/2, min(i, size-1), key);
}
int main(int argc, char *argv[])
{
    int i;
    double utime0, stime0, wtime0, utime1, stime1, wtime1;
    int size = atoi(argv[1]);
    int key = atoi(argv[2]);
    int *array = (int *)malloc(size * sizeof(int));
    for (i = 0; i < size; i++)
        scanf("%d", array + i);
    printf("Busqueda exponencial (key:%d size:%d).\n\n", key, size);
    //*********************************************************************
    // Iniciar el conteo del tiempo para las evaluaciones de rendimiento
    //*********************************************************************
    uswtime(&utime0, &stime0, &wtime0);
    //*********************************************************************
    // Evaluar los tiempos de ejecución
    //*********************************************************************
    int index = exponencial(array, size, key);
    if (index != 1)
        printf("El numero %d se encontro en la posicion %d.\n", key, index);
    else
        printf("El numero %d no existe en el arreglo.\n", key);
    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

```c
/**
 * 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 "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 arr [], int x, int n);

int main(int argc, char* argv[])
{
    int i;
    double utime0, stime0, wtime0, utime1, stime1, wtime1;
    int size = atoi(argv[1]);
    int key = atoi(argv[2]);
    int *array = (int *)malloc(size * sizeof(int));
    for (i = 0; i < size; i++)
        scanf("%d", array + i);
    printf("Busqueda fibonacci (key:%d size:%d).\n\n", key, size);
    //*****************************************************************
    // Iniciar el conteo del tiempo para las evaluaciones de rendimiento
    //*****************************************************************
    uswtime(&utime0, &stime0, &wtime0);
    //*****************************************************************
    // Evaluar los tiempos de ejecución
    //*****************************************************************
    int index = busquedafibonacci(array, key, size);
```

```
    if (index != 1)
        printf("El numero %d se encontro en la posicion %d.\n", key, index);
    else
        printf("El numero %d no existe en el arreglo.\n", key);
    uswtime(&utime1, &stime1, &wtime1);
    // Cálculo del tiempo de ejecución del programa
    printf("Tiempo real: %.10e s\n", wtime1 - wtime0);
    free(array);
    return 0;
}

//entran dos numeros y retorna el menor de los dos
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 dezpalazan
 * los valores  y se resntan 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 valores 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 arreglo ordenado su tamaño y la llave a buscar
 * @return la posicion donde se encontro el numero o si no se encontro un -1
*/
int busquedafibonacci (int arr [], int x, int n)
{
    int auxf2,auxf1,auxfm;//auxf2 tomara el valor de n-1 y auxf1 n-2 y auxfm alm
```

```
    int rango,i;
    auxf2 = 0;
    auxf1 = 1;
    auxfm= auxf2 + auxf1;
    while(auxfm < n){ //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 elind
    while (auxfm > 1) {
        // verifica  si auxfm2 es no es vacio
        i = min(rango + auxf2, n - 1);
        //Si numbuscar es mayor que el valor de  auxfm2  el arreglo se dezplaza
        if (arr [i] <x) {
            auxfm= auxf1;
            auxf1 = auxf2;
            auxf2 = auxfm- auxf1;
            rango = i;
        }
        //en cambio si  es mayor que el valor en arr[auxfm2],  dezplazamos el ar
        else if (arr [i]> x) {
            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] == x){
        return rango + 1;
    }
    //si no se encuentra se retorna -1
    return -1;
}
```

Busqueda lineal

```c
/**
 * 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 "tiempo.h"
int busquedaLineal(int [], int, int);
/**
 *
 * busquedaLineal
 *
 * Esta funcion busca un numero dentro de una lista.
 *
 * Se ingresa su tamaño y compara cada elemento del arreglo desde
 * la posicion 0 con la llave, en caso de que el numero si este en
 * el arreglo la funcion devuelve la posicion mas 1 donde se encuentra
 * el numero, en caso contrario regresa un -1.
 *
 * @param un arreglo su tamaño y la llave a buscar
 * @return la posicion donde se encontro el numero o si no se encontro un -1
*/
int busquedaLineal(int array[], int size, int key)
{
    int i;
    for (i = 0; i < size; i++)
        if (array[i] == key)
            return i+1;
    return -1;
```

```
}
int main(int argc, char *argv[])
{
    int i;
    double utime0, stime0, wtime0, utime1, stime1, wtime1;
    int size = atoi(argv[1]);
    int key = atoi(argv[2]);
    int *array = (int *)malloc(size * sizeof(int));
    for (i = 0; i < size; i++)
        scanf("%d", array + i);
    printf("Busqueda lineal (key:%d size:%d).\n\n", key, size);
    //*****************************************************************
    // Iniciar el conteo del tiempo para las evaluaciones de rendimiento
    //*****************************************************************
    uswtime(&utime0, &stime0, &wtime0);
    //*****************************************************************
    // Evaluar los tiempos de ejecución
    //*****************************************************************
    int index = busquedaLineal(array, size, key);
    if (index != 1)
        printf("El numero %d se encontro en la posicion %d.\n", key, index);
    else
        printf("El numero %d no existe en el arreglo.\n", key);
    uswtime(&utime1, &stime1, &wtime1);
    // Cálculo del tiempo de ejecución del programa
    printf("Tiempo real: %.10e s\n", wtime1 - wtime0);
    free(array);
    return 0;
}
```