**UNIVERSIDAD AUTONOMA** **DE BAJA CALIFORNIA**

**Algoritmos y estructura de datos**

**Practica #4. Análisis de algoritmos empírico y recursión**

**Alumno**

Caudillo Sánchez Diego

**Matricula**

1249199

**Grupo**

551

**Docente**

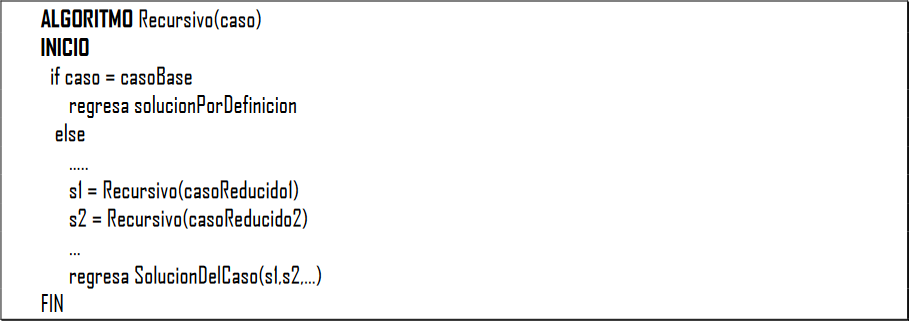
Alma Leticia Palacios Guerrero

**Fecha** **de entrega**

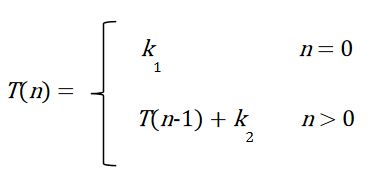
15/Mar/2019

**Introducción**

Un algoritmo recursivo consiste en obtener la solución total de un problema a partir de la solución de casos más sencillos de ese mismo problema. Estos casos se resuelven invocando al mismo algoritmo, el problema se simplifica hasta llegar al caso más sencillo de todos llamado caso base, cuya solución se da por definición. El aspecto de un algoritmo recursivo es el siguiente:



El tiempo de ejecución de un algoritmo recursivo se expresa por medio de una ecuación de recurrencia, la cual es una función T(n). En las recurrencias se separa el tiempo de ejecución del caso básico del tiempo de ejecución del caso recursivo. En este último caso, se utiliza la misma función T(n) para representar el tiempo de ejecución de la llamada recursiva, por ejemplo: En el caso del factorial, si consideramos que el caso base tiene un tiempo de ejecución k1y que el resto de los casos tienen un tiempo de ejecución T(n-1) + k2podemos substituir n por n-1 para obtener T(n-1).Así, el factorial queda expresado con la siguiente ecuación de recurrencia:



**Competencia**

Identificar los factores de diseño y entorno de ejecución que afectan el desempeño de un algoritmo con el propósito de proponer soluciones de software que optimicen el uso de los recursos.

**Problema**

Crear un programa que realice la búsqueda binaria de manera iterativa y recursiva.

**Código**

/\*

DESCRIPCION

Crear un programa en el cual se declare un vector con palabras, para despues hacer una

busqueda binaria, la cual consiste en tener todos los datos ordenados en orden alfabetico

y mediante un un pivote, el cual inicia en el centro del vector y apartir de esa posicion

el pivote se desplaza a la derecha o izquierda, dependiendo si el caso base no es igual a

a la palabra que se este buscando dentro del vector.

NOTA

•Los datos DEBEN ser cadenas de caracteres, no numéricos.

• No se pide que capture las cadenas, inicialice los arreglos con palabras en orden alfabético.

▪ Las funciones deben presentar en todo momento en pantalla los datos sobre los que se está realizando

la Búsqueda.

\*/

#include <stdio.h>

#include <string.h>

#include <time.h>

void printData(int posicion, char \*key, time\_t time, int iterations);

int binarySearchIterativo(char str[20][11], char\* key, int init, int str\_size, int \*iterations);

int binarySearchRecursivo(char str\_vec[20][11], char\* key, int init, int str\_size, int \*iterations);

void copyString(char str\_vector[20][11], char\* str, int index);

int main(int argc, char const \*argv)

{

char str\_vec[20][11] = {"aaaaaaaaaa","bbbbbbbbbb","cccccccccc","dddddddddd","eeeeeeeeee","ffffffffff",

"gggggggggg", "hhhhhhhhhh", "iiiiiiiiii", "jjjjjjjjjj", "kkkkkkkkkk", "llllllllll", "mmmmmmmmmm", "nnnnnnnnnn",

"oooo", "qqqqq", "rrrrr"};

char key[11];

int str\_size = sizeof(str\_vec)/sizeof(str\_vec[0]), init = 0, ans\_iter, ans\_recursivo, iterations = 0;

time\_t time;

printf("Palabra que desea buscar: "); gets(key);

// impresion de busqueda binaria de forma iterativa

time = clock(); //inicia de la ejecucion

ans\_iter = binarySearchIterativo(str\_vec, key, init, str\_size-1, &iterations);

puts(" >>>> Iterativo <<<<");

time = clock()-time; // fin de la jecucion

printData(ans\_iter, key, time, iterations); // impresion de los datos.

// impresion de busqueda binaria de forma recursiva

time = clock(); // inicio del tiempo

ans\_recursivo = binarySearchRecursivo(str\_vec, key, init, str\_size-1, &iterations);

puts(" >>>> Recursivo <<<<");

time = clock()-time; // fin del tiempo de ejecucion

printData(ans\_recursivo, key, time, iterations); // impresion de datos

return 0;

}

/\*

DESCRIPCION:

funcion que imprime los siguientes datos: la posicion donde se encontro (si es que fue asi),

tiempo de ejecucion de la funcion y el numero de iteraciones.

PARAMETROS

- posicion: entero en la posicion donde se encontro la cadena. Si es -1 significa que no se encontro.

- key: palabra que el usuario quiera encontrar.

- time: tiempo de ejecucion de la funcion.

- iterations: numero de iteraciones que realizo la busqueda.

\*/

void printData(int posicion, char \*key, time\_t time, int iterations)

{

double total\_time = ((double)time)/CLOCKS\_PER\_SEC;

/\* impresion de datos en caso de que no se encuentre la cadena en el arreglo\*/

if(posicion == -1)

{

printf("No se encontro la palabra [%s]\n\n", key);

printf("Tiempo de ejecucion: %f seconds\n", total\_time);

printf("Numero de iteraciones: %d\n\n", iterations);

}

/\* impresion de los datos cuando la busqueda haysa sido exitosa.\*/

else

{

printf("La palabra [%s] se encontro en la posicion [%d]\n\n", key, posicion);

printf("Tiempo de ejecucion: %f seconds\n", total\_time);

printf("Numero de iteraciones: %d\n\n", iterations);

}

}

/\*

DESCRIPCION:

funcion que busca una palabra dentro de un arreglo, mediante un algoritmo de busqueda el cual consiste

en cadenas ordenadas en orden alfabetico. Mediante un pivote el cual apunta en la posicion media del

arreglo y de ahi parte si es que se debe mover hacia izquierda o derecha dependiendo si la letra de la

palabra que se esta buscando es menor o mayor a la palabra posicionada en el medio del arreglo.

La funcion regresa la posicion de donde se haya encontrado la palabra, caso contrario retorna un -1.

PARAMETROS

- str\_vec: vector llena con cadenas.

- key: palabra la cual se quiere buscar.

- init: inicio del arreglo

- str\_size: tamaño de la cadena.

- iterations: cantidad de iteraciones que realiza la funcion al buscar una palabra.

\*/

int binarySearchIterativo(char str\_vec[20][11], char\* key, int init, int str\_size, int \*iterations)

{

char str\_aux[20] = {0}; // cadena auxiliar para comparar palabras.

while(init <= str\_size)

{

int pivot = init + (str\_size-init)/2; // se inicializa nuestro pivote, el cual apunta en medio del arreglo.

\*iterations += 1; // cantidad de itraciones que realiza el programa.

if((int)key[0] == (int)str\_vec[pivot][0]) // si la llave y el arreglo en la posicion inicial del pivote coinciden

// en la primera letra entonces entra a la condicion para despues comparar toda la cadena.

{

copyString(str\_vec, str\_aux, pivot);

if( strcmp(key, str\_aux ) == 0) return pivot; // si la comparacion devuelve un 0, significa que ambas cadenas

// son iguales.

}

if(str\_vec[pivot][0] < key[0]) init = pivot+1; // si la letra inicial de la llave es mayor a la posicion inicial de

// pivote, entonces se busca en el lado derecho del arreglo.

else str\_size = pivot-1; // caso contrario se busca en el lado izquierdo.

}//\*exe\_time = clock()/CLOCKS\_PER\_SEC; // cantidad de tiempo en que se ejecuto el programa.

return -1; // Retorna un -1 en caso de no encontrarse en el arreglo.

}

/\*

DESCRIPCION:

Funcion que busca una palabra en un array de cadenas de manera recursiva. La funcion se vuelve a

llamar en caso de que no se encuentre en la posicion inicial, lo cual significa que tendra despues

de la primera iteracion se tendra que buscar en el lado derecho o izquierdo, dependiendo de la palabra

que se este buscando. La funcion devuelve la posicion donde se haya encontrado, caso contrario devuelve

un -1 en caso de que no se encuentre.

PARAMETROS

- str\_vec: vector llena con cadenas.

- key: palabra la cual se quiere buscar.

- init: inicio del arreglo

- str\_size: tamaño de la cadena.

- iterations: cantidad de iteraciones que realiza la funcion al buscar una palabra.

\*/

int binarySearchRecursivo(char str\_vec[20][11], char\* key, int init, int str\_size, int\* iterations)

{

char str\_aux[20] = {0}; // cadena auxiliar que se utiliza para comparar palabras.

if (str\_size >= init)

{

//\*exe\_time += clock()/CLOCKS\_PER\_SEC; // tiempo de ejecucion. A cada llamada se le suma el tiempo anterior.

int pivot = init + (str\_size - init) / 2; // pivote inicial donde se inicia la busqueda, la cual cambia depende

// si la letra esta a la derecha o izquierda

if (str\_vec[pivot][0] == key[0]) // se comprueba que si la letra esta en el medio inmediatamente se devuelve la posicion

// esta condicion es nuestro caso base de la recursion.

{

copyString(str\_vec, str\_aux, pivot); // copia una cadena basandose en la letra inicial, si son iguales regresa un 1.

if( strcmp(key, str\_aux) == 0) return pivot;

}

if (str\_vec[pivot][0] > key[0]) // en caso de que la letra se encuentre de lado izquierdo de nuestro pivote inicial

// retorna un llamado a la funcion, restandole un 1 al pivote para asi recorrer la busqueda hacia la izquierda.

return binarySearchRecursivo(str\_vec, key, init, pivot-1, iterations+1); // De otro modo la cadena se

else

return binarySearchRecursivo(str\_vec, key, pivot+1,str\_size, iterations+1); // busca hacia el lado derecho

// sumandole un 1 al pivote.

}return -1; // si recorre todo el arreglo y no encuentra la palabra, entonces devuelve un -1.

}

/\*

DESCRIPCION:

Funcion que copia una cadena del vector de cadenas. Copia respecto a la posicion que se le mande

como parametro.

PARAMETROS

- str\_vector: vector de cadenas de la cual se va extraer la palabra.

- str: lugar donde se va a guardar la cadena solicitada.

- index: indice de la cadena que va a copiar.

\*/

void copyString(char str\_vector[20][11], char\* str,int index)

{

int i = 0;

/\*copia la cadena respecto al pivote\*/

while(str\_vector[index][i] != 0)

{

str[i] = str\_vector[index][i];

i++;

}

}

**Desarrollo**

Implemente la búsqueda binaria recursiva e iterativa

Utilizando el mismo conjunto de datos para ambas búsquedas **elabore una tabla** con los siguientes incisos:

1. El tiempo para el peor de los casos
2. El tiempo para el mejor de los casos
3. El tiempo para cualquier otro caso
4. La cantidad de iteraciones realizadas en ambas búsquedas para cada caso
5. Ejecute los incisos *a)* al *c)* con cadenas del mismo tamaño
6. Ejecute los incisos *a)* al *c)* con cadenas de doble tamaño ¿Afecta esto el tiempo de ejecución?
7. Determine nuevamente los incisos a) al d) pero ejecutando las funciones en una computadora distinta y con distintas cargas en el sistema.

**Tabla Recursión**

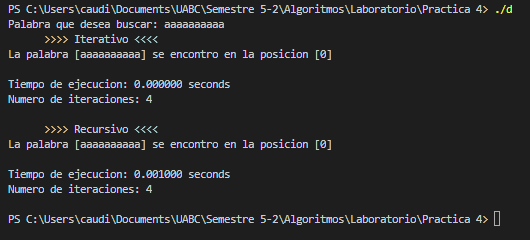
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Tamaño de la cadena | Peor de los casos  (tiempo) | Iteraciones | Mejor de los casos  (tiempo) | Iteraciones | Otro | Iteraciones |
| 5 | 0.001 sec | 4 | 0.000000 sec | 1 | 0.004 sec | 5 |
| 10 | 0.001 sec | 5 | 0.000000 sec | 1 | 0.004 sec | 5 |

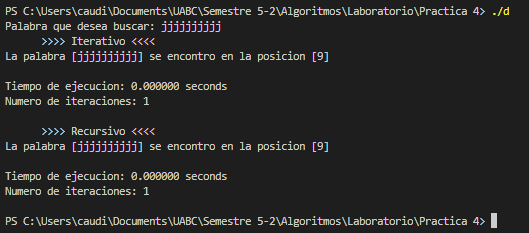
**Tabla Iterativo**

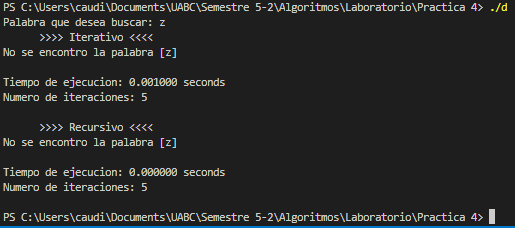
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Tamaño de la cadena | Peor de los casos  (tiempo) | Iteraciones | Mejor de los casos  (tiempo) | Iteraciones | Otro | Iteraciones |
| 5 | 0.000 sec | 5 | 0.000000 sec | 1 | 0.004 sec | 5 |
| 10 | 0.001 sec | 4 | 0.000000 sec | 1 | 0.004 sec | 5 |

* **¿Cuál implementación es más rápida?**
  + A decir verdad, en mi computadora los cambios son muy mínimos, en algunos casos, el tiempo de ejecución son casi iguales. En algunas ocasiones el iterativo suele ser mejor, especialmente en el peor de los casos.
* **¿Afecta el tamaño de los datos al tiempo de ejecución?**
  + En realidad, no, los tiempos han de ser muy pequeños que la función no los toma en cuenta ya que son mínimos.
* **¿Afecta si tiene otras aplicaciones abiertas el tiempo de ejecución?**
  + En mi computadora al parecer no afecta, cuando se tiene Spotify, Word, Firefox y visual studio, el tiempo sigue siendo el mismo.
* **¿Qué cambios observa cuando ejecuta el programa en otra computadora?**

**Evidencia**







**Conclusión**

Esta practica se concluye con que dependiendo de la velocidad del procesador las instrucciones van hacer ejecutadas. En mi caso de mi computadora no se diferenciaba mucho sobre el tamaño de caracteres de la cadena. Ya que ejecutaba al mismo tiempo la búsqueda. Esta vez el tiempo de ejecución en ambas funciones fueron muy similares, no tuvieron mucha diferencia en ambas.

**Fuente**

María del Carmen Gómez & Jorge Cervantes. (2004). *Introducción al Análisis al Diseño de Algoritmos.* México: Publidisa.