
COMPARACIÓN ALGORITMO RECURSIVO CON ALGORITMO ITERATIVO

Programación de Sistemas

7 DE ABRIL DE 2017

Sebastián Barbas 14025
Telmo García-Verdugo 14172
Cristina Escibano 14130

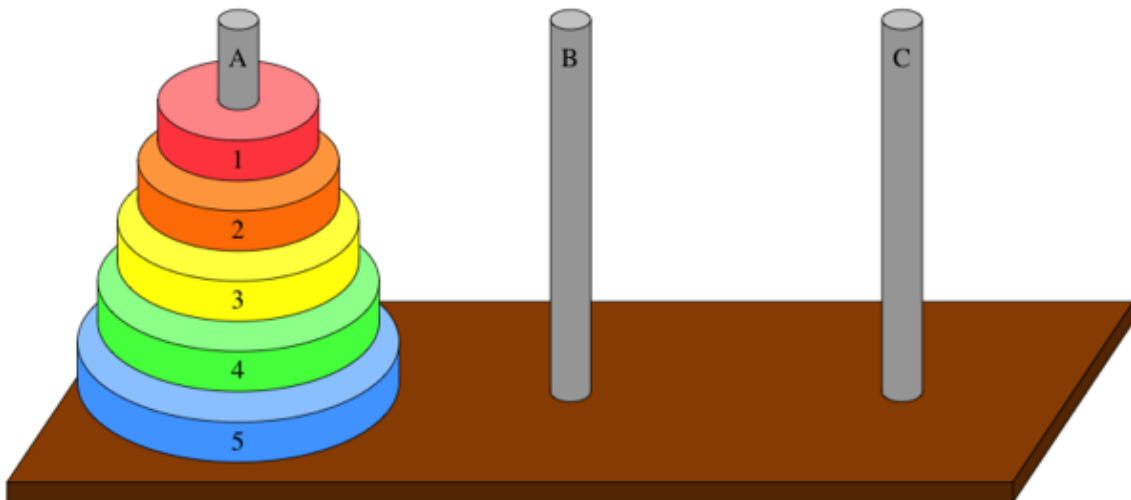
Introducción

En este proyecto vamos a estudiar las ventajas y desventajas en clave de tiempo de ejecución, memoria utilizada o número de movimientos entre el mismo algoritmo implementado en forma iterativa o en forma recursiva.

En primer lugar vamos a definir las diferencias entre estas dos formas de trabajo; funciones iterativas son aquellas que se implementan por medio de algoritmos que se caracterizan por ejecutarse mediante ciclos. Y, por otro lado, funciones recursivas son aquellas que se invocan a sí mismas en algún momento de su ejecución.

Para nuestro proyecto vamos a basarnos en el algoritmo de las Torres de Hanoi. Este rompecabezas tiene por objetivo cambiar todos los discos de la primera estaca a la tercera siguiendo unas reglas:

- Sólo se puede mover un disco cada vez.
- Un disco de mayor tamaño no puede descansar sobre uno más pequeño que él mismo.
- Sólo puedes desplazar el disco que se encuentre arriba en cada varilla



Funciones empleadas

Para elaborar nuestro programa hemos implementado el algoritmo tanto en iterativo como en recursivo mediante funciones que, posteriormente, utilizaremos en el programa principal. Se han añadido instrucciones en el código de los ficheros, no están puestos en las fotos para una mayor legibilidad.

Función del algoritmo recursivo

```

C:\Users\Cristina\Desktop\ESCUELA\asignaturas\Programacion de Sistemas\trabajo1\Nueva idea\hanoi3.c - Notepad++
Archivo  Editar  Buscar  Vista  Codificación  idioma  Configuración  Herramientas  Macro  Ejecutar  Plugins  Ventana  ?
hanoi3.c  Trabajo2.c  mov.c  mov.h  hanoi3.c
1  #include<stdio.h>
2  #include "mov.h"
3  #include<time.h>
4
5  void hanoi3(int n,int inic,int temp,int fin, long int *llam, long int *ops){
6
7      (*llam)++;
8      (*ops)++;
9      if (n == 1) {
10         printf("mover disco de la estaca %d a la estaca %d\n",inic,fin);
11     }
12     (*ops)++;
13     if(n>1){
14         hanoi3(n-1,inic,fin,temp, llam, ops);
15         (*ops)++;
16
17         printf("mover disco de la estaca %d a la estaca %d\n",inic,fin);
18
19         hanoi3(n-1,temp,inic,fin, llam, ops);
20         (*ops)++;
21     }
22 }

```

Función del algoritmo iterativo

```

C:\Users\Cristina\Desktop\ESCUELA\asignaturas\Programacion de Sistemas\trabajo1\Nueva idea\hanoi3.c - Notepad++
Archivo  Editar  Buscar  Vista  Codificación  idioma  Configuración  Herramientas  Macro  Ejecutar  Plugins  Ventana  ?
hanoi3.c  hanoi3.c
1  #include <stdio.h>
2  #include "mov.h"
3  void iterativo(long int n, long int *ops){
4      long int x,a,b;
5      printf("\n");
6      (*ops)++;
7
8      for (x = 1; x < (1 << n); x++){
9          (*ops)=(*ops)+2;
10         if(n%2){
11             printf( "mover disco de la estaca %i a la estaca %i\n", (x&x-1)%3, ((x|x-1)+1)%3 );
12             (*ops) = (*ops) + 7;
13         }
14         else {
15             a=(x&x-1)%3;
16             b=((x|x-1)+1)%3;
17             (*ops) = (*ops) + 8; /*7 operaciones del a y b más una del if de abajo*/
18             if (a == 0) {
19                 a = 0;
20                 (*ops)++;
21             }
22             else {
23                 (*ops)++;
24                 if (a == 1) {
25                     a = 2;
26                     (*ops)++;
27                 }
28                 else {
29                     a = 1;
30                     (*ops)++;
31                 }
32             }
33             (*ops)++;
34             if (b == 0) {
35                 b = 0;
36                 (*ops)++;
37             }
38             else {
39                 (*ops)++;
40                 if (b == 1) {
41                     b = 2;
42                     (*ops)++;
43                 }
44             }
45         }
46         printf( "mover disco de la estaca %i a la estaca %i\n",a, b );
47         (*ops)++; /* Para la operación x++ */
48     }
49 }

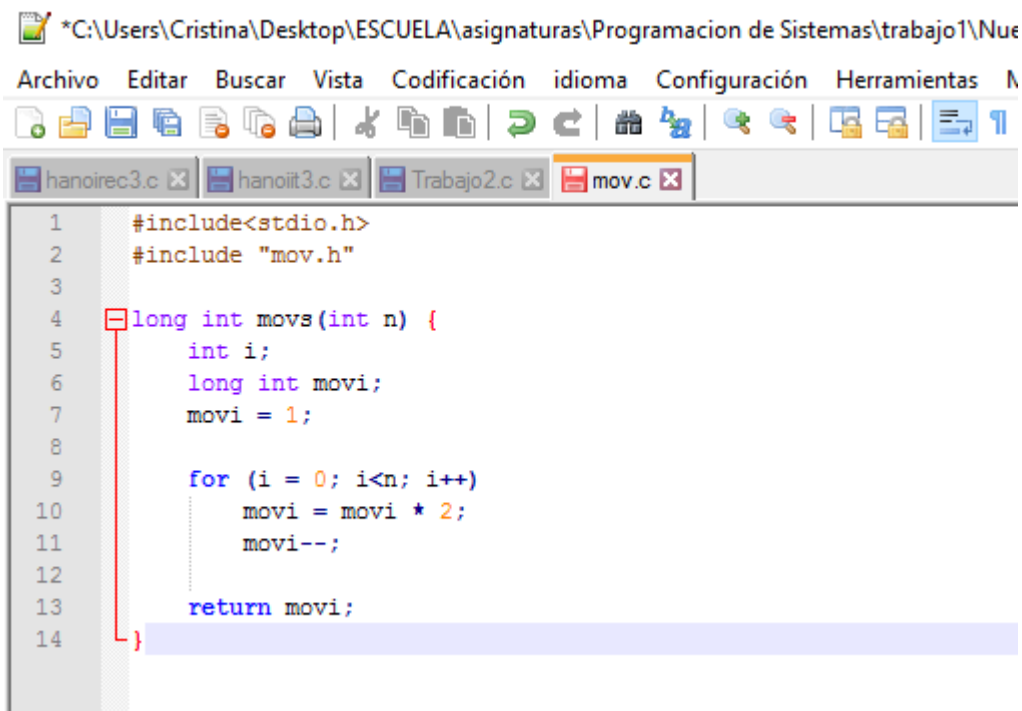
```

Source file length: 934 lines: 46 Ln: 40 Col: 34 Sel

Variables de medida de parámetros

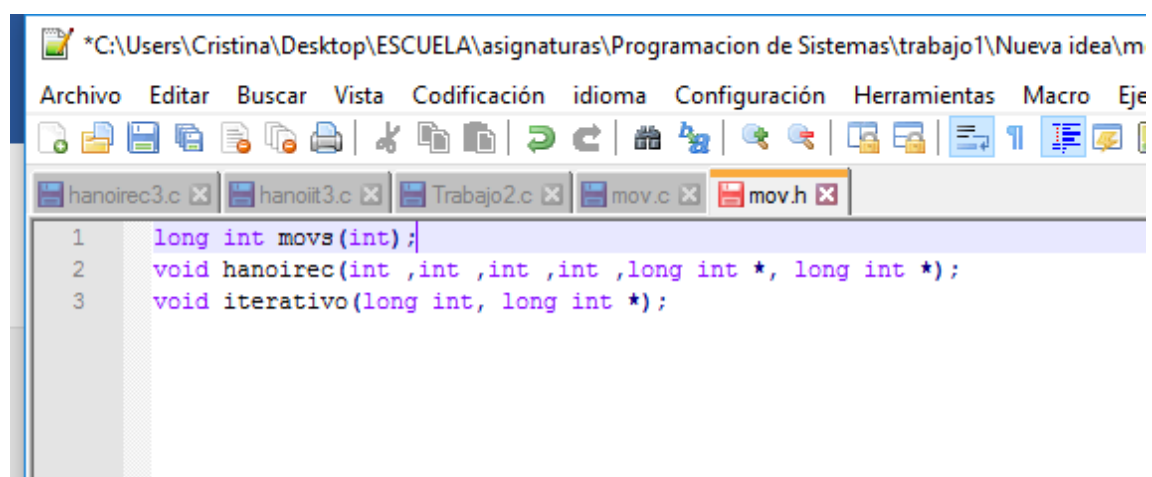
Para comparar los dos experimentos hay que definir variables de medida de las magnitudes a estudiar. En nuestro caso, para comprobar la eficiencia de las dos formas de trabajo hemos definido las siguientes:

- El número de movimientos '*movs*': para ello hemos creado un fichero llamada con el siguiente código.



```
*C:\Users\Cristina\Desktop\ESCUELA\asignaturas\Programacion de Sistemas\trabajo1\Nue
Archivo  Editar  Buscar  Vista  Codificación  idioma  Configuración  Herramientas  N
hanoi3.c x hanoi3.c x Trabajo2.c x mov.c x
1  #include<stdio.h>
2  #include "mov.h"
3
4  long int movs(int n) {
5      int i;
6      long int movi;
7      movi = 1;
8
9      for (i = 0; i<n; i++)
10         movi = movi * 2;
11         movi--;
12
13     return movi;
14 }
```

Y su respectivo fichero *.h* donde se implementan los prototipos de las funciones.



```
*C:\Users\Cristina\Desktop\ESCUELA\asignaturas\Programacion de Sistemas\trabajo1\Nueva idea\m
Archivo  Editar  Buscar  Vista  Codificación  idioma  Configuración  Herramientas  Macro  Eje
hanoi3.c x hanoi3.c x Trabajo2.c x mov.c x mov.h x
1  long int movs(int);
2  void hanoi3(int ,int ,int ,int ,long int *, long int *);
3  void iterativo(long int, long int *);
```

- El número de llamadas a la función '*llam*': contabiliza las veces que entramos en la función recursiva. En el caso de utilizar la función iterativa siempre será la unidad.
- Espacio en memoria '*tamn*': Se hace un tratamiento diferente para recursivo y para iterativo. En el caso recursivo implementamos:

$$tamn=llam*(4*(sizeof(int))+sizeof(int *));$$

donde multiplicamos variable *llam* (número de veces que entramos en la función recursiva) por el espacio que ocupa en memoria cada vez que se llama a dicha función.

En el caso iterativo al no crearse nuevas variables solo se necesita:

$$tamn=2*(sizeof(long int));$$

- Número de comparaciones u operaciones que realiza cada función '*opera*': dentro de las funciones recursiva e iterativa hemos implementado un contador que incrementa el número exacto de operaciones o comparaciones que se realizan en cada expresión.

Programa principal

Tras crear las funciones explicadas creamos en programa principal.

```

1  #define _CRT_SECURE_NO_WARNINGS
2  #include<stdio.h>
3  #include<time.h>
4  #include<string.h>
5  #include "mov.h"
6
7  int main() {
8
9      long int tamn=0, llam=0, opera=0;
10     int n, i;
11     double tiempo = 0;
12     char c[20], d;
13     clock_t t1, t2;
14
15     printf("Este programa se encarga de resolver el problema de la Torre de i. Este problema consiste en pasar un numero de
16     bloques, ordenado de forma decreciente en tamaño desde una estaca 1, hasta una estaca 3, con ayuda de otra complementaria. Un
17     bloque de mayor tamaño nunca puede estar encima de otro de menor tamaño, pero si a la inversa \n");
18
19     do {
20         printf("\n Como quieres que se solucione el problema:\n Si lo quiere por iteracion teclee: iterativo \n Si quiere por
21         recursividad teclee: recursividad ");
22         printf("\n Si quiere salir del programa teclee: salir \n");
23         gets(c);
24         printf("\n Vuelve a darle al intro");
25         scanf("%c", &d);
26
27         if (!strcmp(c, "recursividad")) {
28             printf("\n Solucion recursividad. Cuantos bloques hay (numero mayor que 0):");
29
30             do scanf("%d%c", &n, &d);
31             while (n <= 0);
32
33             t1 = clock();
34             hanoi3c(n,0,1,2,&llam, &opera);
35             t2 = clock();
36
37             tamn=llam*(4*(sizeof(int))+2*sizeof(long int *));
38
39             tiempo = (double)(t2 - t1) / CLOCKS_PER_SEC;
40
41             printf("\n El numero de movimientos es %ld y tarda %f milisegundos y ocupa %ld bytes, realizando %ld llamadas y hace
42             %ld comparaciones u operaciones", movs(n), 1000*tiempo, tamn, llam, opera);
43
44             llam = 0;
45             opera = 0;
46         }
47         else {
48             if (!strcmp(c, "iterativo")) { /*Lo negamos porque si la comparacion es cierta, devuelve un 0, si es falso devuelve -1*/
49                 printf("\n Solucion iterativa. Cuantos bloques hay (numero mayor que 0): ");
50
51                 do scanf("%d%c", &n, &d);
52                 while (n <= 0);
53
54                 t1 = clock();
55                 iterativo(n, &opera);
56                 t2 = clock();
57
58                 tamn=4*(sizeof(long int));
59                 tiempo = (double)(t2 - t1) / CLOCKS_PER_SEC;
60
61                 printf("\n El numero de movimientos es %d y tarda %f milisegundos y ocupa %d bytes haciendo solo una llamada y %ld
62                 comparaciones u operaciones", movs(n), 1000*tiempo, tamn, opera);
63
64                 opera = 0;
65             }
66             else(if (strcmp(c, "salir")))
67                 printf("Cadena no valida");
68         }
69     } while (strcmp(c,"salir"));
70     return 0;
71 }

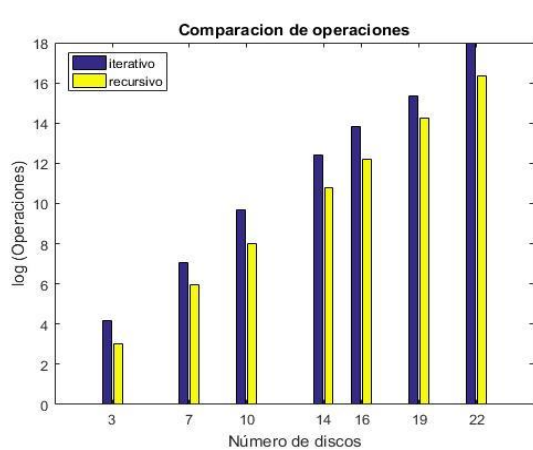
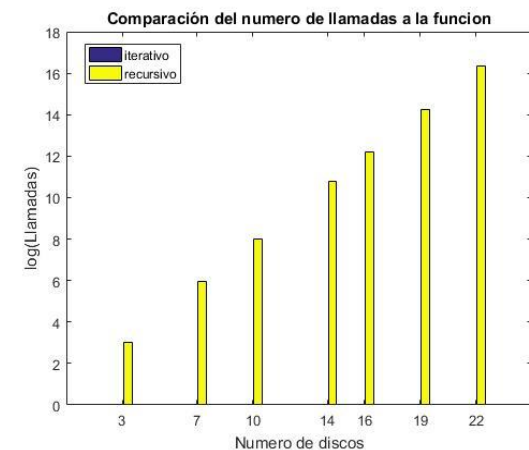
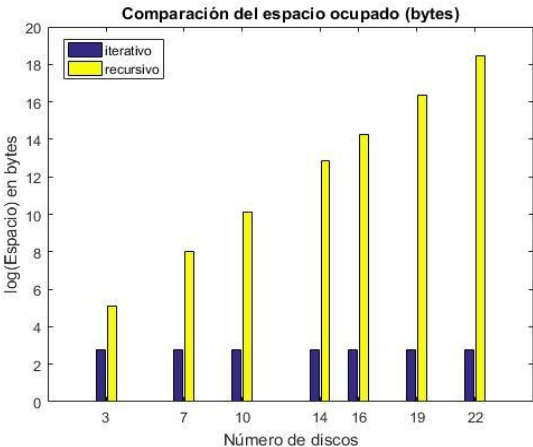
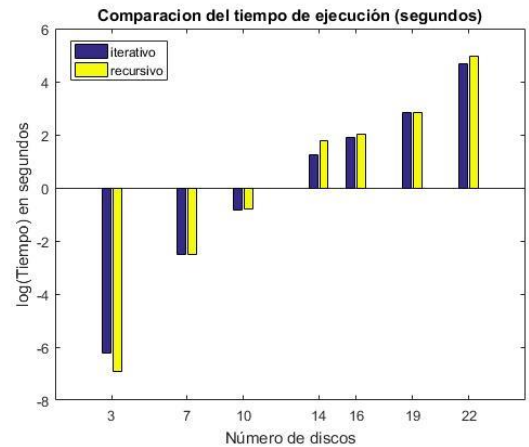
```

Resaltamos que el `strcmp` devuelve un 0 cuando la comparación es cierta y devuelve un -1 en caso contrario. En el caso de la línea 43 de código queremos que entre en el `if` si es cierto, por lo tanto, implementamos el `strcmp` negado.

Pruebas realizadas

Tras compilar todos los ficheros indicados anteriormente realizamos las siguientes pruebas, y comparamos gráficamente.

| NUEMRO DE DISCOS | ITERATIVO | | | | | RECURSIVO | | | | |
|------------------|----------------|--------------|-----------------|----------|-------------|----------------|--------------|-----------------|----------|-------------|
| | Nº movimientos | Tiempo (min) | Espacio (bytes) | Llamadas | Operaciones | Nº movimientos | Tiempo (min) | Espacio (bytes) | Llamadas | Operaciones |
| 3 | 7 | 2 | 16 | 1 | 64 | 7 | 1 | 168 | 7 | 20 |
| 7 | 127 | 82 | 16 | 1 | 1 e3 | 127 | 80 | 3048 | 127 | 380 |
| 10 | 1 e3 | 428 | 16 | 1 | 15 e3 | 1 e3 | 444 | 24 e3 | 1 e3 | 3 e3 |
| 14 | 16 e3 | 3 e3 | 16 | 1 | 251 e3 | 16 e3 | 442 | 393 e3 | 16 e3 | 49 e3 |
| 16 | 65 e3 | 6 e3 | 16 | 1 | 1 e6 | 65 e3 | 6 e3 | 1 e6 | 65 e3 | 196 e3 |
| 19 | 524 e3 | 17 e3 | 16 | 1 | 4 e6 | 534 e3 | 17 e3 | 12 e6 | 52 e4 | 1 e6 |
| 22 | 4 e6 | 107 e3 | 16 | 1 | 64 e6 | 4 e6 | 14 e4 | 100 e6 | 4 e6 | 12 e6 |



A medida que se aumenta el número de discos el tiempo que tarda en ejecutarse el programa es mayor, como cabe esperar. Al principio nos llamó la atención que, aunque el programa recursivo realiza muchas llamadas y el iterativo solo una la diferencia de tiempo no es significativamente diferente. Tras observar el número de operaciones que se realizan vemos que el programa iterativo lleva a cabo muchas más comparaciones u operaciones que el recursivo, por lo tanto, entendemos que la diferencia de tiempo sea pequeña.

Vemos que en lo referido al espacio ocupado en memoria el iterativo es mucho más eficiente, no solo por el hecho de que solo ocupe 16 bytes frente a 168 bytes, 3048 bytes, etc; sino porque podemos saber a priori el espacio en memoria que vamos a utilizar, en cambio en el recursivo no podemos estimar previamente cuanto espacio necesitaremos y además es mucho mayor.

A pesar de que tras las comparaciones realizadas el programa iterativo parece bastante mejor que el recursivo, es cierto que es mucho menos intuitivo de programar.

Reparto de los roles de trabajo

En un primer momento trabajamos los tres juntos en las horas de prácticas de la asignatura y definimos las líneas generales del proyecto. Empezamos a implementar código básico del funcionamiento de las Torres de Hanoi y comenzamos a solventar los diferentes problemas que nos iban surgiendo.

Cuando estaba todo más o menos estructurado dividimos las partes, Telmo se encargó en profundidad de la función de iterativo y de investigar como contabilizar el tiempo, Sebastián implemento la función *main* e investigó como contabilizar la memoria utilizada y las operaciones realizadas en cada caso. Cristina implementó la función recursiva, llevo a cabo las diferentes pruebas y elaboro el documento.

Aunque es verdad que hemos dividido el trabajo, no ha habido necesidad de crear roles definidos, hemos trabajado eficientemente los tres sin que nadie haya tenido que organizar los pasos que debía seguir cada uno. Además, aunque cada uno elaborara una parte hemos ido compartiendo todos los archivos para que los tres supiéramos como se iba realizando el proyecto, aportando ideas y corrigiendo posibles errores.