UNIVERSIDAD AUTONOMA DE BAJA CALIFORNIA

ALGORITMOS Y ESTRUCTURAS DE DATOS

PRACTICA #2. RECURSION

ALUMNO: CAUDILLO SANCHEZ DIEGO

MATRICULA: 1249199

GRUPO: 551

DOCENTE: ALMA LETICIA PALACIOS

FECHA DE ENTREGA: 01/MARZO/2019

Competencia

Establecer las diferencias, ventajas y las soluciones recursivas e iterativas mediante la implementación de algoritmos usando ambas técnicas para identificar las problemáticas en donde la recursividad sea la solución a los problemas reales.

Introducción

Una función o procedimiento que se puede llamar a sí mismo se llama *recursivo*. La *recursión* (**recursividad**) es una herramienta muy potente en algunas aplicaciones, sobre todo de cálculo. La recursión puede ser utilizada como una alternativa a la repetición o estructura repetitiva. El uso de la recursión es particularmente idóneo para la solución de aquellos problemas que pueden definirse de modo natural en términos recursivos. La escritura de un procedimiento o función recursiva es similar a sus homónimos no recursivos; sin embargo, para evitar que la recursión continúe indefinidamente es preciso incluir una condición de terminación. La razón de que existan lenguajes que admiten la recursividad se debe a la existencia de estructuras específicas tipo *pilas* (*stack*, en inglés) para este tipo de procesos y memorias dinámicas. Las direcciones de retorno y el estado de cada subprograma se guardan en estructuras tipo pilas.

Muchas funciones matemáticas se definen recursivamente. Un ejemplo de ello es el *factorial* de un número entero *n.*

La función se define de la siguiente manera:

$$n! = \begin{cases} 1 \ si \quad n = 0 \quad 0! = 1 \\ nx(n-1)x(n-2)x \ ... \ x3x2x1 \quad si \ n > 0 \ n.\ (n-1)\ .\ (n-2)....3.2.1 \end{cases}$$

Si se observa la fórmula anterior cuando *n > 0*, es fácil definir *n!* en función de *(n–1)!* Por ejemplo, *5!*

```
5! = 5x4x3x2x1          = 120
4! = 4x3x2x1            = 24
3! = 3x2x1              = 6
2! = 2x1                = 2
1! = 1x1                = 1
0! = 1                  = 1
```

Problema

Una hembra canina no esterilizada tendrá en su primer año fértil dos celos y como producto al menos n cachorros por camada de los cuales la mitad son hembras. Supongamos que en cada camada hay 8 perritos, esto es 16 en un año, y la mitad, 8 serán hembras las cuales no estarán esterilizadas. Transcurrido un año, la hembra original tendrá otros 16 cachorros y sus 8 primeras descendientes repetirán el patrón y tendrán cada una otros 16 cachorros, haciendo un total de (8 x 16 =128) + 16=144. Al cabo de 5 años los descendientes de los descendientes habrán procreado 74899 caninos. Esta progresión se verifica en la siguiente relación:

1 año: 16
2 año: 16 +128 =144
3 año: 16+128+ 1024 = 1168
4 año: 16+128+1024+8192=9360
5 año: 16+128+1024+8192+65536=74899

Tabla 1.

Proponer algoritmo para la solución

Codificación

```c
/*Librerias a utilizar*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

/*Estructura de datos con la informacion para determinar la proyeccion de una
camdada*/
typedef struct
{
    int camada;
    int proyeccion;
    int vecesAnio;
}animal_t;

/*Cabecera de las funciones*/
void capturaDatos(animal_t* perro);
int determinarProyeccionCiclico(animal_t perro, int* time_ciclico);
void determinarProyeccionRecursivo(animal_t perro,int x, int * ans, int*
time_recursivo);

int main(int argc, char const *argv[])
```

```c
{
    animal_t perro;
    int total = 0, counter = 1, time_recursivo = 0, time_ciclico = 0; //
variables para la función recursiva.
    capturaDatos(&perro);       // captura de los datos
    determinarProyeccionRecursivo(perro,counter, &total, &time_recursivo); //
llamada a la función recursiva
    printf("************************************************\n");
    printf("Proyeccion en %d a%cos (Ciclico): %d\n",perro.proyeccion,
164,determinarProyeccionCiclico(perro, &time_ciclico)); //impresion de los datos
    printf("Proyeccion en %d a%cos (Recursivo): %d\n",perro.proyeccion, 164,
total/2) ;
    /*impresión de los tiempos de cada función*/
    printf("Tiempo en funci%cn recursivo = %ld ms\n", 164, time_ciclico);
    printf("Tiempo en funci%cn recursivo = %ld ms\n", 164, time_recursivo);
    return 0;
}
/*
    DESCRIPCION
        Funcion la cual permite capturar los datos de sobre la informacion de un
perro al tener una camada.
    PARAMETROS:
        perro: estructura de datos con la informacion para determinar la
proyeccion la camada de un perro.
*/
void capturaDatos(animal_t* perro)
{
    /*captura cada una de la información necesaria para guardarla en la
estructura y despues poder usarla
    en las funciones siguientes.*/
    printf("Tama%co de la camada: ", 164); scanf("%d", &perro->camada);
    printf("Veces al a%co: ", 164); scanf("%d", &perro->vecesAnio);
    printf("A%cos de proyecci%cn: ", 164, 162); scanf("%d", &perro->proyeccion);
}
/*
    DESCRIPCION:
        Función la cual determina la proyección a futuro de las camadas. La
información
        para resolver el problema se encuentra en la estructura de datos que se
envia
        como parametro. La funcion determina la proyección mediante ciclos
iterativos for
        y do-while.
    PARAMETROS:
```

```
        perro: se encuatra la información sobre la cantidad de veces que se tiene
una camada
        al año, el tamaño de la camada, asi como la años de proyección
        time_ciclo: variable donde se guarda el tiempo de ejecución de la función
*/
int determinarProyeccionCiclico(animal_t perro, int* time_ciclico)
{
    int i, total = 0;
    /* el ciclo do-while se ejecuta mientras la proyeccion se mayor a 0.
    Ya que la funcion primero calcula la cantidad de perros del ultimo año hacia
el primero,
    y va creciendo exponencialmente mientras la variable de control sea mayor a
0. */
    do
    {
        for(i = 0; i < perro.proyeccion ; i++)
            total += pow((perro.camada * perro.vecesAnio),i+1);
        perro.proyeccion--; // se decremente el valor de la proyección con el fin
de que no se cicle infinitamente.
    } while (perro.proyeccion > 0);
    *time_ciclico = clock()/CLOCKS_PER_SEC;
    return total/2; // regresa la mitad, ya que las instrucciones pide este
requerimiento.
}
/*
    DESCRIPCION:
        Función la cual determina la proyección a futuro de las camadas. La
información
        para resolver el problema se encuentra en la estructura de datos que se
envia
        como parametro. La funcion determina la proyección mediante recursion,
quiere decir
        que se llama así mismo y modifica los datos para que a lasiguiente
llamada se vayan
        iterando, todo esto con una condicion que controla que el ciclo no se
cicle infinitamente
        o hasta que haya un stack overflow (des).
    PARAMETROS:
        perro: estrcutura de datos en la cual se encuentra la informacion del
perro para determinar
        la proyeccion deseada.
        counter: variable de control que permite controlar el numero de llamadas
(recursion) a la
        funcion para llegar al resultado deseado.
        total: vatiable en la cual se guarda el resultado de la proyeccion.
```

```
        time_recursivo: variable donde se guarda el tiempo de ejecución de la
función
*/
void determinarProyeccionRecursivo(animal_t perro,int counter, int* total, int
*time_recursivo)
{
    if(perro.proyeccion == 0) *total = 1;    // Si el año de proyeccion es 0,
inmediatamente el
    // resultado es 1 ya que solo se esta contando a la hembra
    else
    {
        if(counter <= perro.proyeccion) //mientras la variable de control no
llegue al año de proyección
        //esperado, seguirá llamando a la función así misma
        {
            *total += pow(perro.camada*perro.vecesAnio,counter)+ *total; // En
cada llamada se le suma el resultado anterior,
            // de otro modo el resultado fuera incorrecto, devolviendo así, la
cantidad de perros de solo el primero año, en
            // caso de que sea mas de un año de proyección.
            determinarProyeccionRecursivo(perro, counter+1, total,
time_recursivo); // La funcioón llamandose así misma.
        }
    }*time_recursivo += clock()/CLOCKS_PER_SEC;
}
```

Evidencia de ejecución

```
PS C:\Users\caudi\Documents\UABC\Semestre 5-2\Algoritmos\Laboratorio\Practica 2> ./d
Tamaño de la camada: 11
Veces al año: 1
Años de proyección: 11
************************************************
Proyección en 11 años (Ciclico): 352816523
Proyección en 11 años (Recursivo): -1073741824
Tiempo en función ciclico = 3 ms
Tiempo en función recursivo = 36 ms
PS C:\Users\caudi\Documents\UABC\Semestre 5-2\Algoritmos\Laboratorio\Practica 2>
```

```
PS C:\Users\caudi\Documents\UABC\Semestre 5-2\Algoritmos\Laboratorio\Practica 2> ./d
Tamaño de la camada: 5
Veces al año: 2
Años de proyección: 12
************************************************
Proyección en 12 años (Ciclico): -456457879
Proyección en 12 años (Recursivo): -1073741824
Tiempo en función ciclico = 2 ms
Tiempo en función recursivo = 26 ms
PS C:\Users\caudi\Documents\UABC\Semestre 5-2\Algoritmos\Laboratorio\Practica 2>
```

```
PS C:\Users\caudi\Documents\UABC\Semestre 5-2\Algoritmos\Laboratorio\Practica 2> ./d
Tamaño de la camada: 8
Veces al año: 3
Años de proyección: 13
************************************************
Proyección en 13 años (Ciclico): -969700888
Proyección en 13 años (Recursivo): -1073741824
Tiempo en función ciclico = 3 ms
Tiempo en función recursivo = 42 ms
PS C:\Users\caudi\Documents\UABC\Semestre 5-2\Algoritmos\Laboratorio\Practica 2>
```

```
PS C:\Users\caudi\Documents\UABC\Semestre 5-2\Algoritmos\Laboratorio\Practica 2> ./d
Tamaño de la camada: 11
Veces al año: 2
Años de proyección: 4
************************************************
Proyección en 4 años (Ciclico): 128546
Proyección en 4 años (Recursivo): 128832
Tiempo en función ciclico = 13 ms
Tiempo en función recursivo = 65 ms
PS C:\Users\caudi\Documents\UABC\Semestre 5-2\Algoritmos\Laboratorio\Practica 2>
```

```
PS C:\Users\caudi\Documents\UABC\Semestre 5-2\Algoritmos\Laboratorio\Practica 2> ./d
Tamaño de la camada: 5
Veces al año: 3
Años de proyección: 3
************************************************
Proyección en 3 años (Ciclico): 1935
Proyección en 3 años (Recursivo): 1942
Tiempo en función ciclico = 2 ms
Tiempo en función recursivo = 8 ms
PS C:\Users\caudi\Documents\UABC\Semestre 5-2\Algoritmos\Laboratorio\Practica 2>
```

```
PS C:\Users\caudi\Documents\UABC\Semestre 5-2\Algoritmos\Laboratorio\Practica 2> ./d
Tamaño de la camada: 8
Veces al año: 4
Años de proyección: 2
************************************************
Proyección en 2 años (Ciclico): 544
Proyección en 2 años (Recursivo): 544
Tiempo en función ciclico = 2 ms
Tiempo en función recursivo = 6 ms
PS C:\Users\caudi\Documents\UABC\Semestre 5-2\Algoritmos\Laboratorio\Practica 2>
```

Desarrollo

Camada	Veces al año	Proyección	Tiempo de ejecución cíclico	Tiempo de ejecución recursivo
11	1	11	3 ms	36 ms
5	2	12	2 ms	26 ms
8	3	13	3 ms	42 ms
11	2	4	13 ms	65 ms
5	3	3	2 ms	8 ms
8	4	2	2 ms	6 ms

¿Cómo afecta el tamaño de la camada al tiempo en ambas implementaciones?

El tamaño de la camada no influye mucho en el tiempo, a partir de cierto punto. Ya que si la camada es demasiado grande el tiempo ejecución también crece bastante.

¿Cómo afecta el tamaño de la proyección al tiempo en ambas implementaciones?

La proyección es el factor que mas afecta al programa, ya que cada una iteración mas con respecto a la proyección la cantidad de perro va aumentando de manera potencial *n+1* y al programa le toma mas tiempo ejecutar las instrucciones.

¿Qué pasa si el tamaño de la camada es muy grande?

El tiempo no se ve afectado si la proyección es grande. Por otro lado, si la camada es muy grande, por ejemplo, mayor a 20, y la proyección es mayor a 7, entonces hay se ve el cambio exponencial con respecto al tiempo.

Conclusión

La práctica se concluye en que la recursividad es un recurso muy útil para ciertas tareas, ya que puede realizar iteraciones con menos tiempo de ejecución. En mi caso particular, esto no se vio reflejado así, ya que el programa es optimo con los ciclos *for* y *do-while*. El motivo fue que ambas funciones realizan el mismo tiempo de ejecución por cada llamada a ellos, y al ser iterativo y llamarse n veces, mientras que la condición fuera verdadera, el tiempo solo iba en aumento con respecto a la cantidad de veces que se llamaba (n*tiempo de ejecución de la función).

Fuentes

Luis Joyanes. (2008). *FUNDAMENTOS DE PROGRAMACIÓN. Algoritmos, estructura de datos y objetos.* Madrid: McGraw Hill.

Luis Joyanes & Ignacio Zahonero. (2004). *Algoritmos y estructura de datos. Una perspectiva en C*. Madrid: McGraw Hill.

Anónimo. (1997). *The Open Group Library*. Recuperado: febrero 25, 2019, de UNIX Sitio web: http://pubs.opengroup.org/onlinepubs/7908799/xsh/clock.html