Capítulo 1

recursividad

1.1. Descripción y Motivación

Existen problemas que para resolverlos tenemos que ejecutar el mismo bloque de instrucciones varias veces, esto se puede lograr con ciclos iterativos o con recursividad. Todos los algoritmos iterativos pueden ser programados recursivamente y viceversa, aun que debemos aprender a elegir cual es la técnica correcta a utilizar. La implementación de un algoritmo iterativo consiste en repetir el cuerpo del bucle en cambio la implementacion de un algoritmo recursivo se basa en ejecutar repetidamente el mismo metodo. Los principales criterios a la hora de elegir entre programar algo iterativamente o recursivamente son: el rendimiento y la simpleza del codigo generado. Supongamos que debemos resolver el problema de sumar los primeros n numeros, dos algoritmos que solucionan este problema son los siguientes:

Iterativo

Listing 1.1: sumaIterativa.cpp

```
int sumaIterativa(int n){
    int resultado = 0;
    for(int i=1;i<=n;i++){
        resultado += i;
    }
    return resultado;
}
```

Recursivo

Listing 1.2: sumaRecursiva.cpp

```
int sumaRecursiva(int n){
    //Caso base
    if(n==1){
        return 1;
    }else{
        return n + sumaRecursiva(n-1);
    }
}
```

Algo muy importante a tener en cuenta en los algoritmos recursivos es el caso base, al igual que en los algoritmos iterativos se debe saber cuando detener la ejecución en los algoritmos recursivos necesitamos saber en donde detenernos. En realidad los dos algoritmos que mostramos tienen una ligera diferencia aun que dan el mismo resultado. En nuestro algoritmo iterativo sumamos desde 0 hasta n de la siguiente manera: $0 + 1 + 2 + 3 + ... + n$, pero en el recursivo sumamos desde n hasta 0: $n + (n - 1) + (n - 2)... + 0$. Si quisieramos que tuvieran un comportamiento mas similar podriamos programar el algoritmo recursivo de la siguiente manera:

Listing 1.3: sumaRecursiva2.cpp

```
int sumaRecursiva(int actual, int n){
    //Caso base
    if(actual==n){
        return actual;
    }else{
        return actual + sumaRecursiva(actual+1,n);
    }
}
```

El caso base esta muy ligado a la manera en que hacemos la recursividad, por lo general la recursividad se hace disminuyendo los parametros del problema, pero no siempre es asi como vimos en el segundo ejemplo, al igual que podemos hacer algoritmos iterativos con el contador ascendente o descendente y tenemos que generar la condicion de detener en base a este, en la recursividad también lo hacemos asi.
Quizas el ejemplo mas claro de recursividad es factorial de n. Ya que la solucion de factorial de n es $n * factorialde(n-1)$, y la solución de $factorialde(n-1) es (n-1) * factorialde(n-2)$ y asi consecutivamente. Por ejemplo factorial de 5 es:
$f(5) = 5 * f(4)$
$f(4) = 4 * f(3)$
$f(3) = 3 * f(2)$
$f(2) = 2 * f(1)$
$f(1) = 1$
por lo tanto
$f(2) = 2 * f(1) = 2 * 1 = 2$
$f(3) = 3 * f(2) = 3 * 2 = 6$
$f(4) = 4 * f(3) = 4 * 6 = 24$
$f(5) = 5 * f(4) = 5 * 12 = 120$
Mas o menos de esa manera funciona la recursividad en código, se van guardando cada llamada al metodo en una cola, al retornar regresa al metodo que la llamo. asi
$f(5) -> f(4) -> f(3) -> f(2) -> f(1)$
Iterativo

Listing 1.4: factorialIterativo.cpp

```
int factorial(int n){
    if(n==0)return 1;
    int resultado = 1;
    for(int i=n;i>=1;i--){
        resultado*=i;
    }
    return resultado;
}
```

Recursivo

Listing 1.5: factorialRecursivo.cpp

```
int factorial(int n){
    //Caso base
    if(n==0){
        return 1;
    }
    if(n==1){
        return 1;
    }
    return n * factorial(n-1);
}
```

Se debe tener cuidado al usar recursividad en no calcular muchas veces la misma solución, por ejemplo con el algoritmo de fibbonacci. su formula recursiva es $f(n) = f(n-1) + f(n-2)$. Si ejecutamos por ejemplo $f(5)$ sucederia lo siguiente:

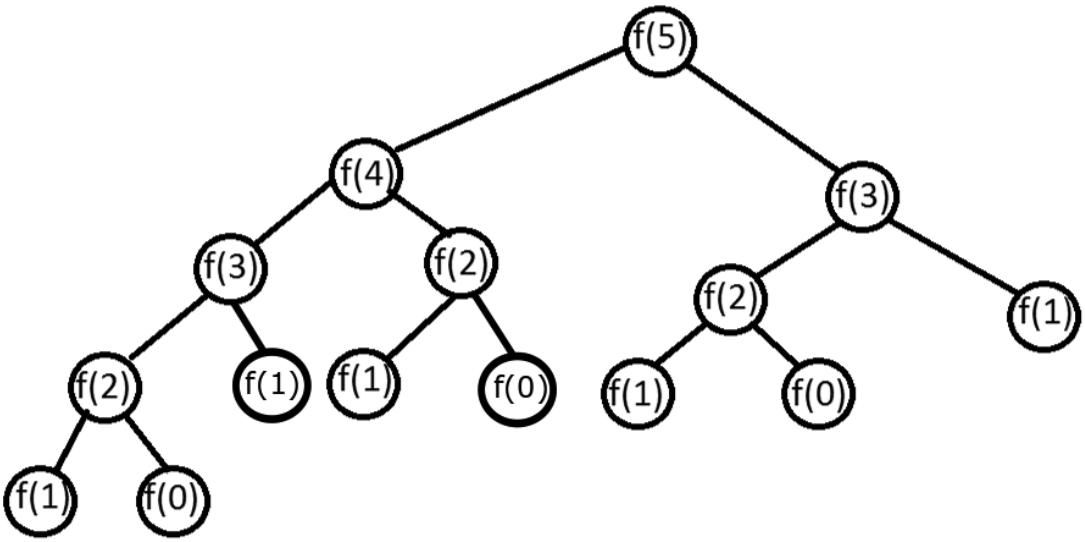

Figura 1.1: fibonacci.png

Podemos observar que recalculamos mucho, esto se puede resolver aplicando tecnicas de DP, pero eso lo veremos en otro capitulo.

1.2. Ejemplos

Ya vimos uno de los ejemplos mas tipicos de recursividad, el del factorial, en esta sección veremos el algoritmo de fibonacci, el algoritmo de Euclides (para hallar el máximo común divisor) y un algoritmo para solucionar las torres de hanoi.

Empecemos por el algoritmo de fibonacci, por definición la suseción de fibonacci comienza de la siguiente forma : 0,1,1,2,3,5,8 ... , cada elemento es la suma de sus dos anteriores. Más formalmente:

$f(n) = f(n-1) + f(n-2)$

Veamos primero como seria el algoritmo de fibonacci sin hacer uso de la recursión.

Listing 1.6: fibonacciIterativo.cpp

```
int fibonacci(int n){
    if(n==0)return 0;
    if(n==1)return 1;
    int a = 0;
    int b = 1;
    int c = a+b;
    for(int i=2;i<=n;i++){
        c = a+b;
        a = b;
        b = c;
    }
    return c;
}
```

Y ahora como seria usando recursión

Listing 1.7: fibonacciRecursivo.cpp

```
int fibonacci(int n){
    if(n==0)return 0;
    if(n==1)return 1;
    return fibonacci(n-1) + fibonacci(n-2);
}
```

Mucho más simple, ¿no lo creen?. Ahora veamos el algoritmo de euclides
Iterativo

Listing 1.8: euclidesIterativo.cpp

```
int euclides(int a,int b){
    int temporal = a;
    while(a>0){
        temporal = a;
        a = b%a;
        b = temporal;
    }
    return b;
}
```

Recursivo

Listing 1.9: euclidesRecursivo.cpp

```
int euclides(int a,int b){
    if(b==0)return a;
    return euclides(b,a%b);
}
```

Por último mi ejemplo favorito para demostrar el potencial de la recursividad,las torres de hannoi. Si no conoces este juego, te recomiendo que primero busques en google “torres de hannoi online”, te saldran multiples opciones para jugarlo, es bastante simple e interesante.

En este caso no pondre una solución iterativa puesto que no se me ocurre ninguna, excepto simulando el comportamiento de la recursividad con una cola, (para saber más detalles al respecto, te invito a profundizar en como funciona internamente la recursividad).
El caso base de esta solución consiste en tener unicamente dos piezas apiladas, saber donde estan apiladas, hacia donde se dirigen, y el otro palo sera nuestro auxiliar.
La solución al caso base es muy sencilla, unicamente debemos desplazar la ficha superior a nuestro palo auxiliar, la ficha base a nuestro palo destino y por ultimo la ficha superior a nuestro destino, y

asi logramos resolver la torre de hannoi de nuestro caso base.

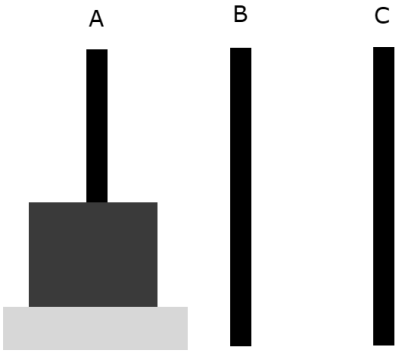

Figura 1.2: torre1.png

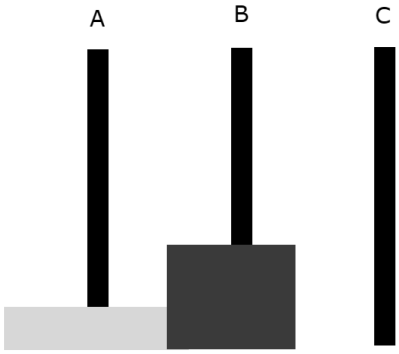

Figura 1.3: torre2.png

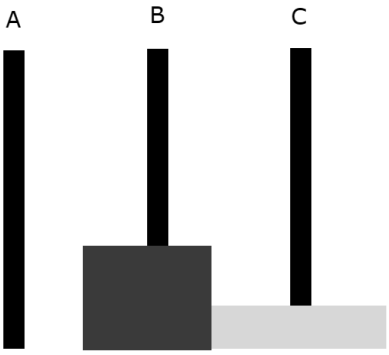

Figura 1.4: torre3.png

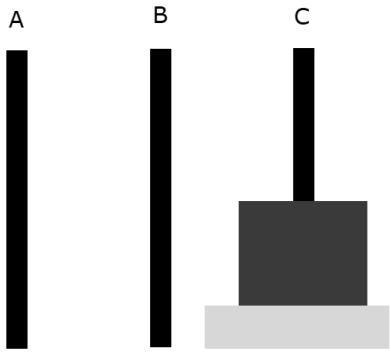

Figura 1.5: torre4.png

Pero que pasaria si fueran mas de dos fichas, he aqui donde viene la recursividad sucederia algo asi

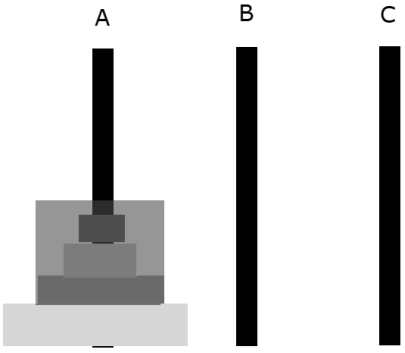

Figura 1.6: torre1-2.png

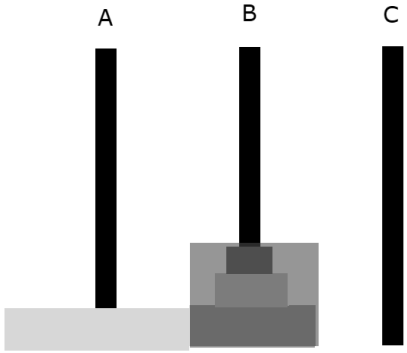

Figura 1.7: torre2-2.png

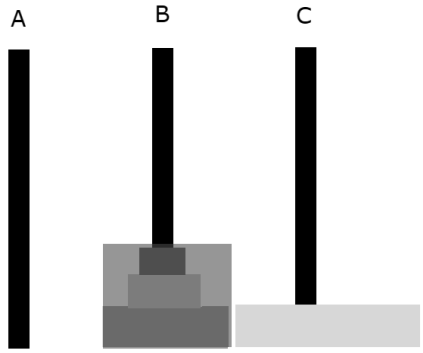

Figura 1.8: torre3-2.png

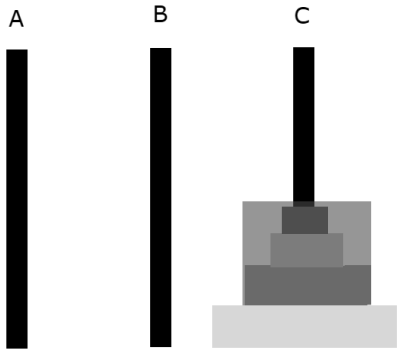

Figura 1.9: torre4-2.png

Internamente la recursion de las fichas sombreadas en rojo desde "torre1-2.png" hacia "torre2-2.png" funcionarian de la siguiente manera:

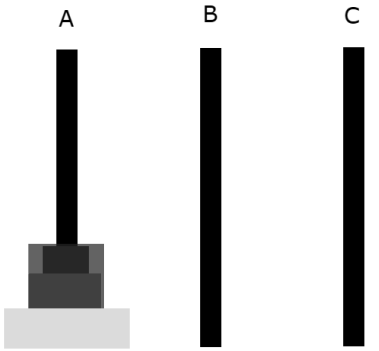

Figura 1.10: torre1-3.png

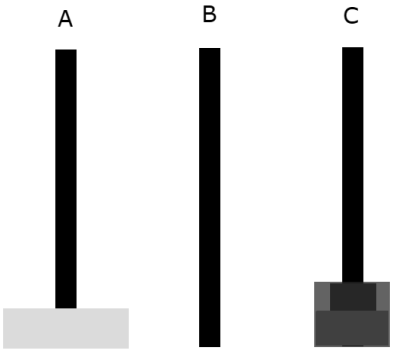

Figura 1.11: torre2-3.png

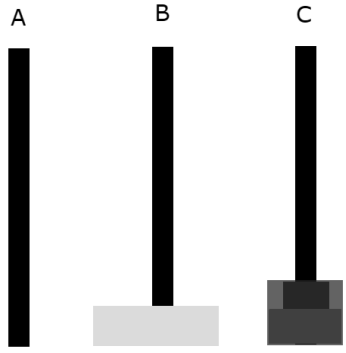

Figura 1.12: torre3-3.png

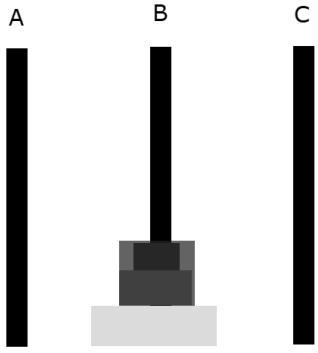

Figura 1.13: torre4-3.png

Y asi sucesivamente (recursivamente). La solución recursiva de la torre de hannoi consiste en llevar la parte superior (todas las piezas menos la base) hacia el palo auxiliar, mover la base al palo destino y finalmente mover la parte superior al palo destino. Cuando la parte superior es de mas de una pieza, se realiza la recursión cambiando invirtiendo el palo destino y el auxiliar. En código seria asi:

Recursivo

Listing 1.10: hannoi.cpp

```
1  void Hanoi(int disco, char origen, char intermedio, char destino){
2
3    if(disco == 1){
4      //caso base, solo movemos el disco a su destino
5      cout << "Mover disco " << disco << " desde " << origen << " hasta " <<
           destino << endl;
6    }else{
7      //movemos la parte superior al intermedio
8      Hanoi(disco-1, origen,destino,intermedio);
9      cout << "Mover disco " << disco << " desde " << origen << " hasta " <<
           destino << endl;
10     //movemos la parte superior al destino
11     Hanoi(disco-1,intermedio,origen,destino);
12   }
13 }
14
15 int main(){
16   int discos;
17   cout << "Ingrese la cantidad de discos: " << endl;
18   cin >> discos;
19   Hanoi(discos, 'A', 'B', 'C');
20
21   system("pause");
22 }
```