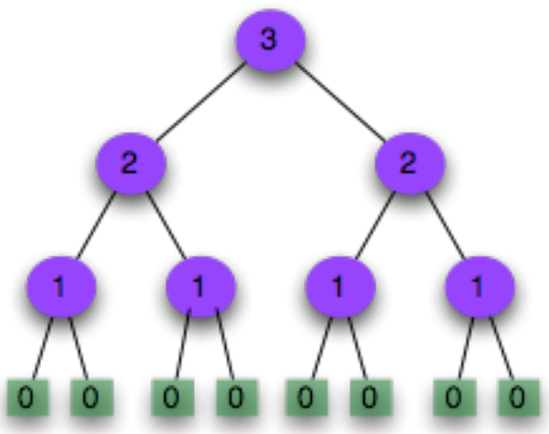
Recursividad

- mat-151

Ejercicio de recursión: dibujando una regla

- Queremos dibujar las marcas de diferentes tamaños de una regla.
- Marcas grandes cada 1/2 cm, marcas más pequeñas cada 1/4 cm ... hasta una resolución dada.
- Suponemos que tenemos a nuestra disposición un procedimiento: **`mark(x,h)`** para hacer una marca de **h** unidades de altura en la posición **x**.
- Si la resolución deseada es **$1/2^n$** cm, reescalamos para poner una marca en cada punto entre **0** y **2^n**, con los puntos extremos no incluidos.
- La marca media debe tener **n** unidades de alto, las marcas en las mitades izquierda y derecha serán de **n-1** unidades de alto, etc.

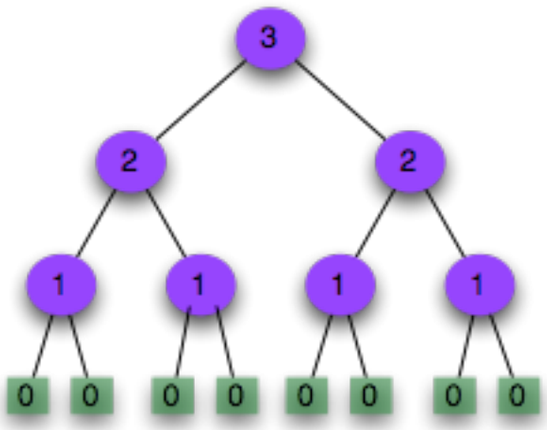
```
void rule ( int l, int r, int h )
{
    int m = (l+r)/2;
    if ( h > 0 )
        {
            rule( l, m, h-1 );
            mark( m, h );
            rule( m, r, h-1 );
        }
}
```



Version recursiva

```
rule(0, 8, 3)
    rule(0, 4, 2)
        rule(0, 2, 1)
            rule(0, 1, 0)
            mark(1, 1)
            rule(1, 2, 0)
        mark(2, 2)
        rule(2, 4, 1)
            rule(2, 3, 0)
            mark(3, 1)
            rule(3, 4, 0)
    mark(4, 3)
    rule(4, 8, 2)
        rule(4, 6, 1);
            rule(4, 5, 0)
            mark(5, 1)
            rule(5, 6, 0)
        mark(6, 2)
        rule(6, 8, 1)
            rule(6, 7, 0)
            mark(7, 1)
            rule(7, 8, 0)
```

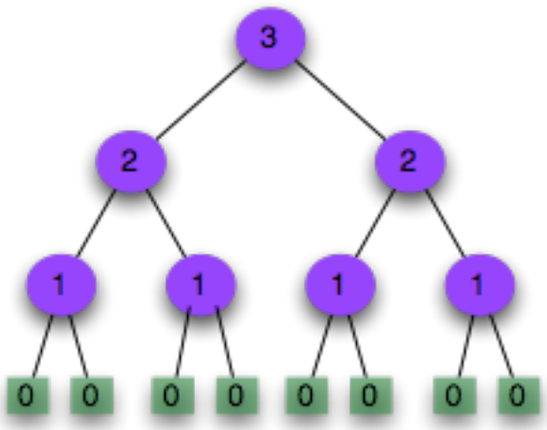
```
void rule ( int l, int r, int h )
{
    int m = (l+r)/2;
    if ( h > 0 )
        {
            rule( l, m, h-1 );
            mark( m, h );
            rule( m, r, h-1 );
        }
}
```



Version recursiva

```
rule(0, 8, 3)
    rule(0, 4, 2)
        rule(0, 2, 1)
            rule(0, 1, 0)
            mark(1, 1)
            rule(1, 2, 0)
        mark(2, 2)
        rule(2, 4, 1)
            rule(2, 3, 0)
            mark(3, 1)
            rule(3, 4, 0)
    mark(4, 3)
    rule(4, 8, 2)
        rule(4, 6, 1);
            rule(4, 5, 0)
            mark(5, 1)
            rule(5, 6, 0)
        mark(6, 2)
        rule(6, 8, 1)
            rule(6, 7, 0)
            mark(7, 1)
            rule(7, 8, 0)
```

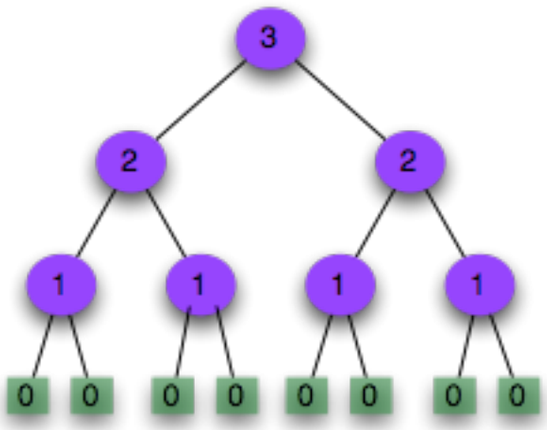
```
void rule ( int l, int r, int h )
{
    int m = (l+r)/2;
    if ( h > 0 )
        {
            rule( l, m, h-1 );
            mark( m, h );
            rule( m, r, h-1 );
        }
}
```




```
rule(0, 8, 3)
    rule(0, 4, 2)
        rule(0, 2, 1)
            rule(0, 1, 0)
            mark(1, 1)
            rule(1, 2, 0)
        mark(2, 2)
        rule(2, 4, 1)
            rule(2, 3, 0)
            mark(3, 1)
            rule(3, 4, 0)
    mark(4, 3)
    rule(4, 8, 2)
        rule(4, 6, 1);
            rule(4, 5, 0)
            mark(5, 1)
            rule(5, 6, 0)
        mark(6, 2)
        rule(6, 8, 1)
            rule(6, 7, 0)
            mark(7, 1)
            rule(7, 8, 0)
```

Version recursiva

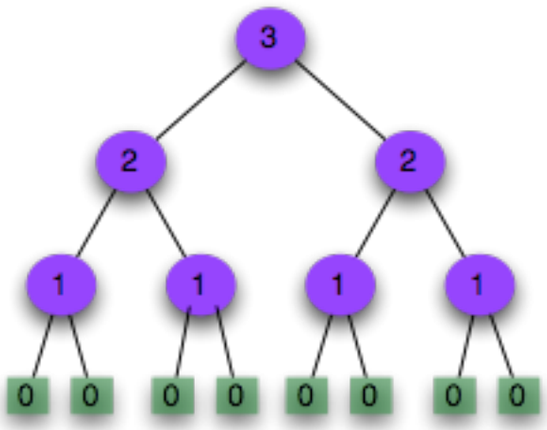
```
void rule ( int l, int r, int h )
{
    int m = (l+r)/2;
    if ( h > 0 )
        {
            rule( l, m, h-1 );
            mark( m, h );
            rule( m, r, h-1 );
        }
}
```




```
rule(0, 8, 3)
    rule(0, 4, 2)
        rule(0, 2, 1)
            rule(0, 1, 0)
            mark(1, 1)
            rule(1, 2, 0)
        mark(2, 2)
        rule(2, 4, 1)
            rule(2, 3, 0)
            mark(3, 1)
            rule(3, 4, 0)
    mark(4, 3)
    rule(4, 8, 2)
        rule(4, 6, 1);
            rule(4, 5, 0)
            mark(5, 1)
            rule(5, 6, 0)
        mark(6, 2)
        rule(6, 8, 1)
            rule(6, 7, 0)
            mark(7, 1)
            rule(7, 8, 0)
```

Version recursiva

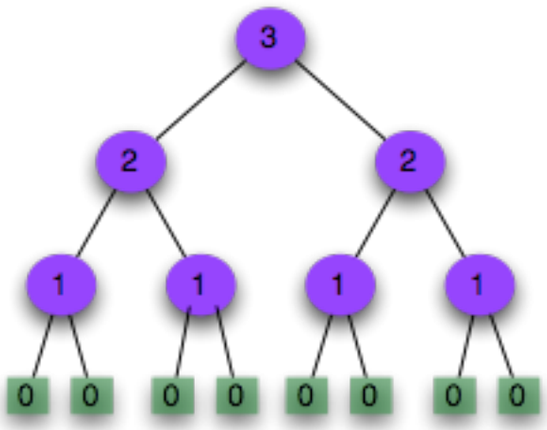
```
void rule ( int l, int r, int h )
{
    int m = (l+r)/2;
    if ( h > 0 )
        {
            rule( l, m, h-1 );
            mark( m, h );
            rule( m, r, h-1 );
        }
}
```




```
rule(0, 8, 3)
    rule(0, 4, 2)
        rule(0, 2, 1)
            rule(0, 1, 0)
            mark(1, 1)
            rule(1, 2, 0)
        mark(2, 2)
        rule(2, 4, 1)
            rule(2, 3, 0)
            mark(3, 1)
            rule(3, 4, 0)
    mark(4, 3)
    rule(4, 8, 2)
        rule(4, 6, 1);
            rule(4, 5, 0)
            mark(5, 1)
            rule(5, 6, 0)
        mark(6, 2)
        rule(6, 8, 1)
            rule(6, 7, 0)
            mark(7, 1)
            rule(7, 8, 0)
```

Version recursiva

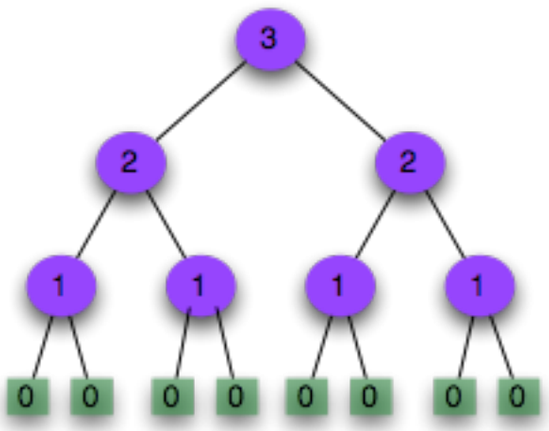
```
void rule ( int l, int r, int h )
{
    int m = (l+r)/2;
    if ( h > 0 )
        {
            rule( l, m, h-1 );
            mark( m, h );
            rule( m, r, h-1 );
        }
}
```




```
rule(0, 8, 3)
    rule(0, 4, 2)
        rule(0, 2, 1)
            rule(0, 1, 0)
            mark(1, 1)
            rule(1, 2, 0)
        mark(2, 2)
        rule(2, 4, 1)
            rule(2, 3, 0)
            mark(3, 1)
            rule(3, 4, 0)
    mark(4, 3)
    rule(4, 8, 2)
        rule(4, 6, 1);
            rule(4, 5, 0)
            mark(5, 1)
            rule(5, 6, 0)
        mark(6, 2)
        rule(6, 8, 1)
            rule(6, 7, 0)
            mark(7, 1)
            rule(7, 8, 0)
```

Version recursiva

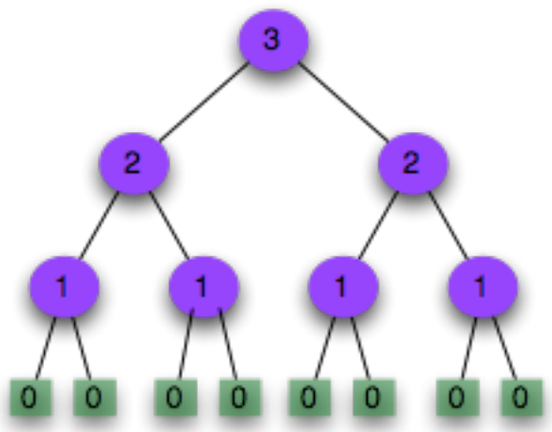
```
void rule ( int l, int r, int h )
{
    int m = (l+r)/2;
    if ( h > 0 )
        {
            rule( l, m, h-1 );
            mark( m, h );
            rule( m, r, h-1 );
        }
}
```



Version recursiva

```
rule(0, 8, 3)
    rule(0, 4, 2)
        rule(0, 2, 1)
            rule(0, 1, 0)
            mark(1, 1)
            rule(1, 2, 0)
        mark(2, 2)
        rule(2, 4, 1)
            rule(2, 3, 0)
            mark(3, 1)
            rule(3, 4, 0)
    mark(4, 3)
    rule(4, 8, 2)
        rule(4, 6, 1);
            rule(4, 5, 0)
            mark(5, 1)
            rule(5, 6, 0)
        mark(6, 2)
        rule(6, 8, 1)
            rule(6, 7, 0)
            mark(7, 1)
            rule(7, 8, 0)
```

```
void rule ( int l, int r, int h )
{
    int m = (l+r)/2;
    if ( h > 0 )
        {
            rule( l, m, h-1 );
            mark( m, h );
            rule( m, r, h-1 );
        }
}
```




```
rule(0, 8, 3)
    rule(0, 4, 2)
        rule(0, 2, 1)
            rule(0, 1, 0)
            mark(1, 1)
            rule(1, 2, 0)
        mark(2, 2)
        rule(2, 4, 1)
            rule(2, 3, 0)
            mark(3, 1)
            rule(3, 4, 0)
    mark(4, 3)
    rule(4, 8, 2)
        rule(4, 6, 1);
            rule(4, 5, 0)
            mark(5, 1)
            rule(5, 6, 0)
        mark(6, 2)
        rule(6, 8, 1)
            rule(6, 7, 0)
            mark(7, 1)
            rule(7, 8, 0)
```

Version recursiva

Ejercicio de recursión: dibujando una regla

- Equivale a resolver el problema de las torres de Hanoi (a contnuación ...)
- Es el mismo problema básico de dividir para vencer: tenemos un problema de dimensión 2^n y lo dividimos en dos problemas de 2^{n-1}.
- ¿Hay una manera no recursiva para calcular el largo de la ***i-ésima*** marca para una ***i*** dada?

```
void ruler ( int l, int r, int h )
{
    for ( int t=1, j=1; t <= h; j+=j, t++ )
        for( int i=0; l+j+i <= r; i += j+j)
            mark(l+j+i, t);
}
```

Version no-recursiva

```
void ruler ( int l, int r, int h )
{
    for ( int t=1, j=1; t <= h; j+=j, t++ )
        for( int i=0; l+j+i <= r; i += j+j)
            mark(l+j+i, t);
}
```

Version no-recursiva

```
void ruler ( int l, int r, int h )
{
    for ( int t=1, j=1; t <= h; j+=j, t++ )
        for( int i=0; l+j+i <= r; i += j+j)
            mark(l+j+i, t);
}
```

Version no-recursiva

```
void ruler ( int l, int r, int h )
{
    for ( int t=1, j=1; t <= h; j+=j, t++ )
        for( int i=0; l+j+i <= r; i += j+j)
            mark(l+j+i, t);
}
```

Version no-recursiva

```
void ruler ( int l, int r, int h )
{
    for ( int t=1, j=1; t <= h; j+=j, t++ )
        for( int i=0; l+j+i <= r; i += j+j)
            mark(l+j+i, t);
}
```

Version no-recursiva

```
void ruler ( int l, int r, int h )
{
    for ( int t=1, j=1; t <= h; j+=j, t++ )
        for( int i=0; l+j+i <= r; i += j+j)
            mark(l+j+i, t);
}
```

Version no-recursiva

```
void ruler ( int l, int r, int h )
{
    for ( int t=1, j=1; t <= h; j+=j, t++ )
        for( int i=0; l+j+i <= r; i += j+j)
            mark(l+j+i, t);
}
```

Version no-recursiva

```
void ruler ( int l, int r, int h )
{
    for ( int t=1, j=1; t <= h; j+=j, t++ )
        for( int i=0; l+j+i <= r; i += j+j)
            mark(l+j+i, t);
}
```

Version no-recursiva

```
void ruler ( int l, int r, int h )
{
    for ( int t=1, j=1; t <= h; j+=j, t++ )
        for( int i=0; l+j+i <= r; i += j+j)
            mark(l+j+i, t);
}
```

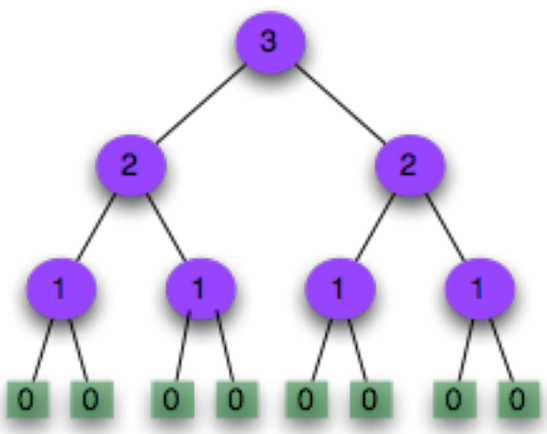
```
0 0 0 1
0 0 1 0       1
0 0 1 1
0 1 0 0       2
0 1 0 1
0 1 1 0       1
0 1 1 1
1 0 0 0       3
1 0 0 1
1 0 1 0       1
1 0 1 1
1 1 0 0       2
1 1 0 1
1 1 1 0       1
1 1 1 1
```

Version no-recursiva

Ejercicio de recursión: dibujando una regla

- El iterativo es tipo de acercamiento conocido como ***bottom-up***. No es recursivo pero es deducido a partir la recursión.
- Se trata de rearreglar el ***orden*** de los cálculos al dibujar la regla.
- Se puede modificar también el orden en la versión recursiva.

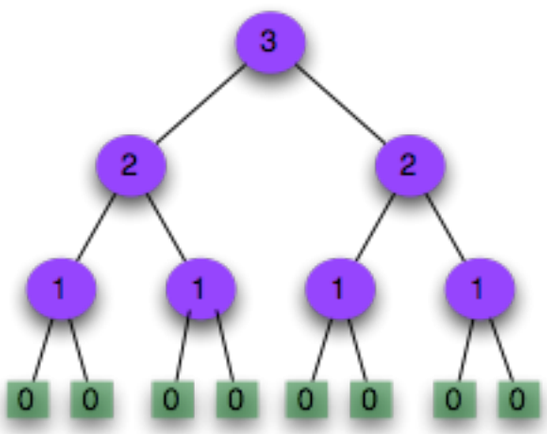
```
void rule( int l, int r, int h )
{
    int m = (l+r)/2;
    if( h > 0 )
        {
            mark(m, h);
            rule(l, m, h-1);
            rule(m, r, h-1);
        }
}
```




```
rule(0,8,3)
    mark(4,3)
    rule(0,4,2)
        mark(2,2)
        rule(0,2,1)
            mark(1,1)
            rule(0,1,0)
            rule(1,2,0)
        rule(2,4,1)
            mark(3,1)
            rule(2,3,0)
            rule(3,4,0)
rule(4,8,2)
    mark(6,2)
    rule(4,6,1)
        mark(5,1)
        rule(4,5,0)
        rule(5,6,0)
    rule(6,8,1)
        mark(7,1)
        rule(6,7,0)
        rule(7,8,0)
```

Version recursiva 2: postfijo vs. prefijo en expresiones aritméticas

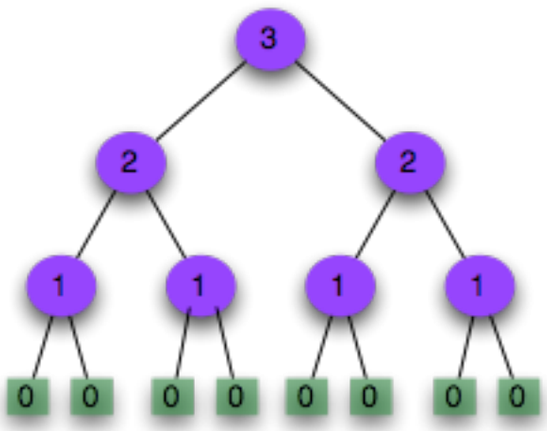
```
void rule( int l, int r, int h )
{
    int m = (l+r)/2;
    if( h > 0 )
        {
            mark(m, h);
            rule(l, m, h-1);
            rule(m, r, h-1);
        }
}
```




```
rule(0,8,3)
    mark(4,3)
    rule(0,4,2)
        mark(2,2)
        rule(0,2,1)
            mark(1,1)
            rule(0,1,0)
            rule(1,2,0)
        rule(2,4,1)
            mark(3,1)
            rule(2,3,0)
            rule(3,4,0)
rule(4,8,2)
    mark(6,2)
    rule(4,6,1)
        mark(5,1)
        rule(4,5,0)
        rule(5,6,0)
    rule(6,8,1)
        mark(7,1)
        rule(6,7,0)
        rule(7,8,0)
```

Version recursiva 2: postfijo vs. prefijo en expresiones aritméticas

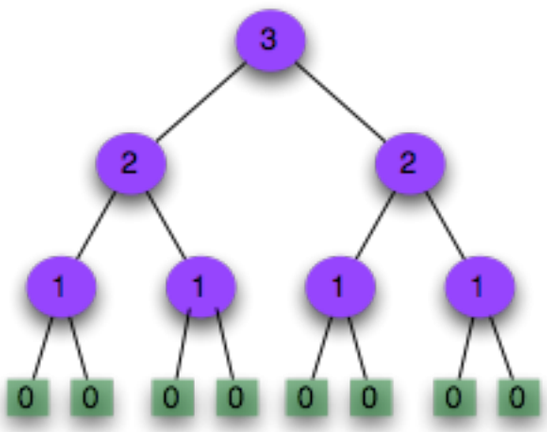
```
void rule( int l, int r, int h )
{
    int m = (l+r)/2;
    if( h > 0 )
        {
            mark(m, h);
            rule(l, m, h-1);
            rule(m, r, h-1);
        }
}
```




```
rule(0,8,3)
    mark(4,3)
    rule(0,4,2)
        mark(2,2)
        rule(0,2,1)
            mark(1,1)
            rule(0,1,0)
            rule(1,2,0)
        rule(2,4,1)
            mark(3,1)
            rule(2,3,0)
            rule(3,4,0)
rule(4,8,2)
    mark(6,2)
    rule(4,6,1)
        mark(5,1)
        rule(4,5,0)
        rule(5,6,0)
    rule(6,8,1)
        mark(7,1)
        rule(6,7,0)
        rule(7,8,0)
```

Version recursiva 2: postfijo vs. prefijo en expresiones aritméticas

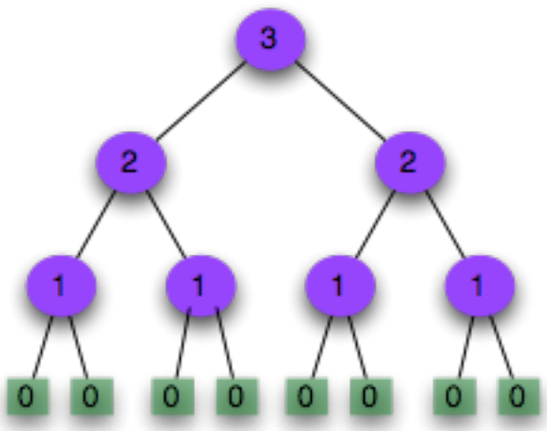
```
void rule( int l, int r, int h )
{
    int m = (l+r)/2;
    if( h > 0 )
        {
            mark(m, h);
            rule(l, m, h-1);
            rule(m, r, h-1);
        }
}
```




```
rule(0,8,3)
    mark(4,3)
    rule(0,4,2)
        mark(2,2)
        rule(0,2,1)
            mark(1,1)
            rule(0,1,0)
            rule(1,2,0)
        rule(2,4,1)
            mark(3,1)
            rule(2,3,0)
            rule(3,4,0)
    rule(4,8,2)
        mark(6,2)
        rule(4,6,1)
            mark(5,1)
            rule(4,5,0)
            rule(5,6,0)
        rule(6,8,1)
            mark(7,1)
            rule(6,7,0)
            rule(7,8,0)
```

Version recursiva 2: postfijo vs. prefijo en expresiones aritméticas

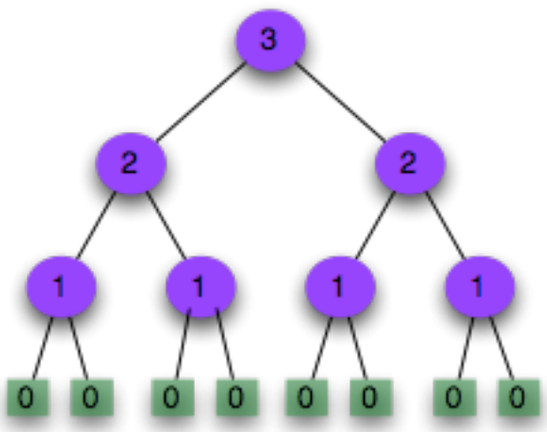
```c
void rule( int l, int r, int h )
{
    int m = (l+r)/2;
    if( h > 0 )
        {
            mark(m, h);
            rule(l, m, h-1);
            rule(m, r, h-1);
        }
}
```




```
rule(0,8,3)
    mark(4,3)
    rule(0,4,2)
        mark(2,2)
        rule(0,2,1)
            mark(1,1)
            rule(0,1,0)
            rule(1,2,0)
        rule(2,4,1)
            mark(3,1)
            rule(2,3,0)
            rule(3,4,0)
    rule(4,8,2)
        mark(6,2)
        rule(4,6,1)
            mark(5,1)
            rule(4,5,0)
            rule(5,6,0)
        rule(6,8,1)
            mark(7,1)
            rule(6,7,0)
            rule(7,8,0)
```

Version recursiva 2: postfijo vs. prefijo en expresiones aritméticas

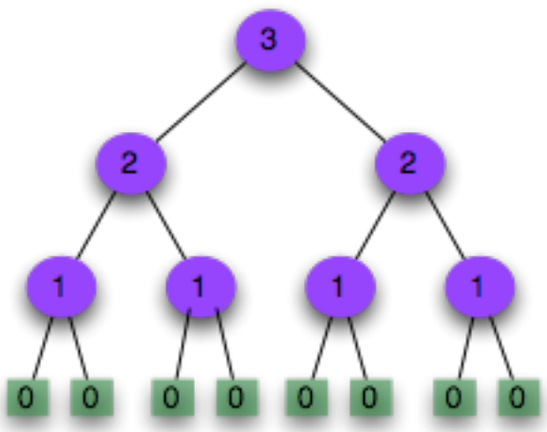
```
void rule( int l, int r, int h )
{
    int m = (l+r)/2;
    if( h > 0 )
        {
            mark(m, h);
            rule(l, m, h-1);
            rule(m, r, h-1);
        }
}
```




```
rule(0,8,3)
    mark(4,3)
    rule(0,4,2)
        mark(2,2)
        rule(0,2,1)
            mark(1,1)
            rule(0,1,0)
            rule(1,2,0)
        rule(2,4,1)
            mark(3,1)
            rule(2,3,0)
            rule(3,4,0)
    rule(4,8,2)
        mark(6,2)
        rule(4,6,1)
            mark(5,1)
            rule(4,5,0)
            rule(5,6,0)
        rule(6,8,1)
            mark(7,1)
            rule(6,7,0)
            rule(7,8,0)
```

Version recursiva 2: postfijo vs. prefijo en expresiones aritméticas

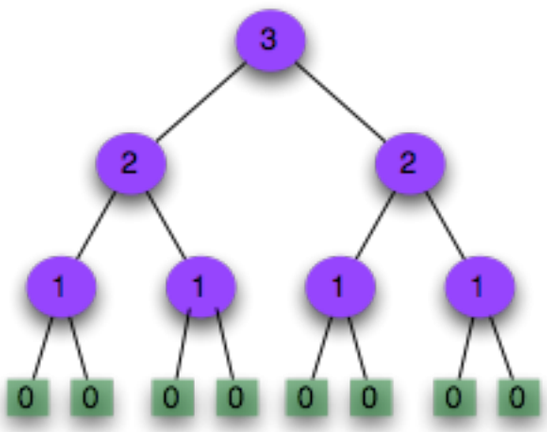
```
void rule( int l, int r, int h )
{
    int m = (l+r)/2;
    if( h > 0 )
        {
            mark(m, h);
            rule(l, m, h-1);
            rule(m, r, h-1);
        }
}
```




```
rule(0,8,3)
    mark(4,3)
    rule(0,4,2)
        mark(2,2)
        rule(0,2,1)
            mark(1,1)
            rule(0,1,0)
            rule(1,2,0)
        rule(2,4,1)
            mark(3,1)
            rule(2,3,0)
            rule(3,4,0)
    rule(4,8,2)
        mark(6,2)
        rule(4,6,1)
            mark(5,1)
            rule(4,5,0)
            rule(5,6,0)
        rule(6,8,1)
            mark(7,1)
            rule(6,7,0)
            rule(7,8,0)
```

Version recursiva 2: postfijo vs. prefijo en expresiones aritméticas

```
void rule( int l, int r, int h )
{
    int m = (l+r)/2;
    if( h > 0 )
        {
            mark(m, h);
            rule(l, m, h-1);
            rule(m, r, h-1);
        }
}
```

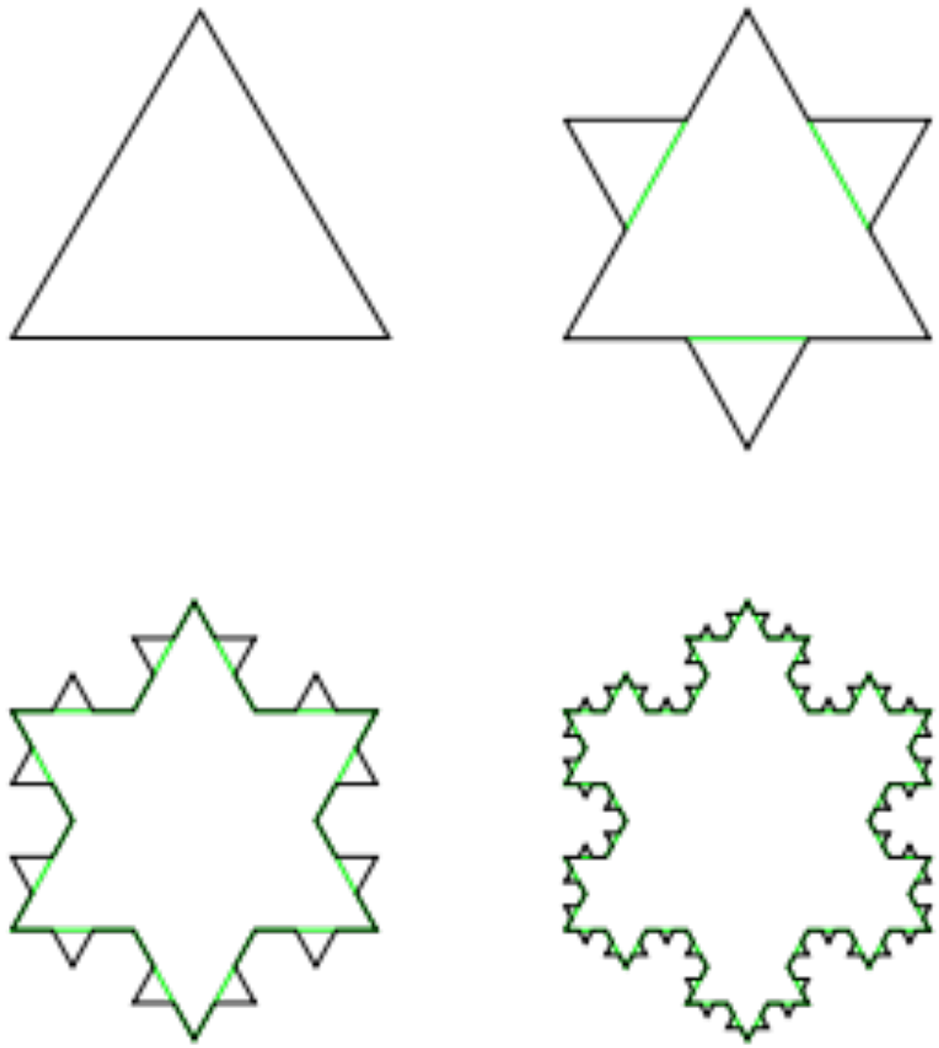

```
rule(0,8,3)
    mark(4,3)
    rule(0,4,2)
        mark(2,2)
        rule(0,2,1)
            mark(1,1)
            rule(0,1,0)
            rule(1,2,0)
        rule(2,4,1)
            mark(3,1)
            rule(2,3,0)
            rule(3,4,0)
    rule(4,8,2)
        mark(6,2)
        rule(4,6,1)
            mark(5,1)
            rule(4,5,0)
            rule(5,6,0)
        rule(6,8,1)
            mark(7,1)
            rule(6,7,0)
            rule(7,8,0)
```

Version recursiva 2: postfijo vs. prefijo en expresiones aritméticas

Ejercicio de recursión: dibujando una regla

- Intuitivamente, ¿qué algoritmo parece mejor cuando queremos dibujar una regla arbitrariamente larga? ¿ por qué ?
- Para algunos problemas el orden de resolución es irrelevante y solo nos interesa resolver el sub-problema antes que el problema principal.
- Considerar que el orden de resolución no solo es útil para determinar el algoritmo más eficiente, sino a veces es indispensable. (procesadores en paralelo)

Ejercicio de recursión: estrella de Koch



Encontrar el máximo de un vector sin ciclos, …

Encontrar el máximo de un vector sin ciclos, …

.. es decir de manera recursiva:

Encontrar el máximo de un vector sin ciclos, …

.. es decir de manera recursiva:

```
Item max(Item a[], int l, int r){
    if(l==r) return a[l];
    int m = (l+r)/2;
    Item u = max(a,l,m);
    Item v = max(a,m+1,r);
    return u>v ? u : v;
}
```

Contar la longitud de la lista de manera recursiva:

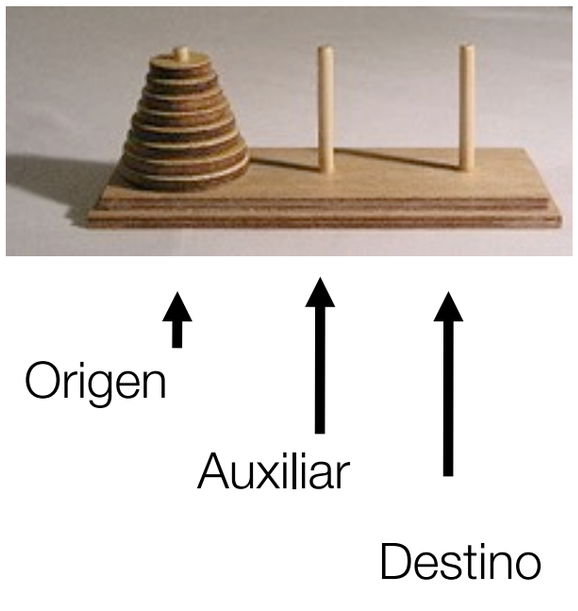
Contar la longitud de la lista de manera recursiva:

```
int count(link x){

    if(x==NULL) return 0;

    return 1 + count(x->next);

}
```

Convertir un numero de base 10 a base 2

```
void pasarbinario (int N){

    if (N < 2) {
            cout << N;
    }
    else {
            pasarbinario(N/2);
            cout << N%2;
    }
}
```

El problema de las Torres de Hanoi



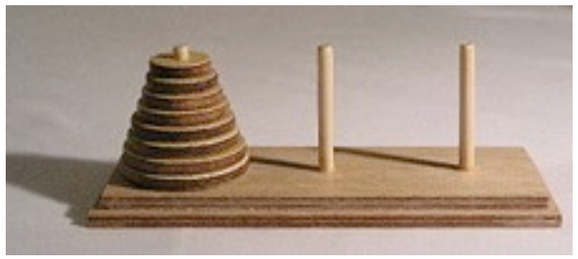
Ver :

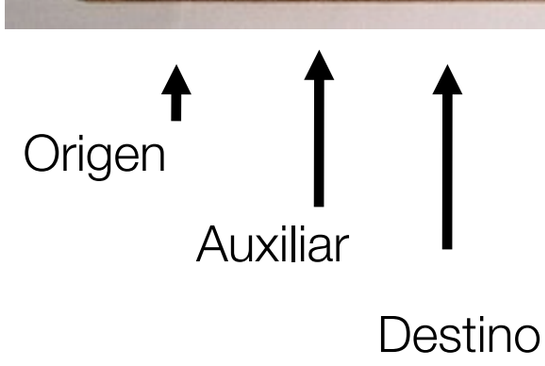
http://rodoval.com/heureka/hanoi/

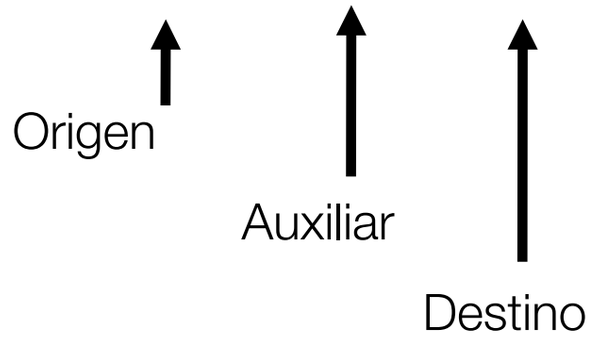
http://www.cut-the-knot.org/recurrence/hanoi.shtml

La leyenda dice que hay que mover 40 discos!
(de oro, por supuesto, en espigas de diamante)

El problema de las Torres de Hanoi







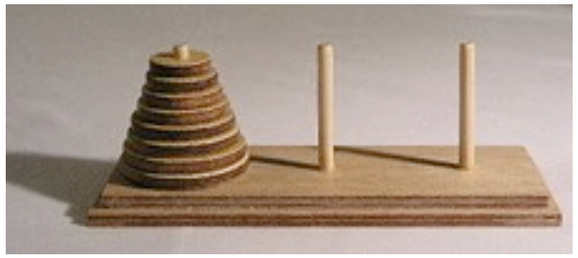
Ver :

http://rodoval.com/heureka/hanoi/

http://www.cut-the-knot.org/recurrence/hanoi.shtml

La leyenda dice que hay que mover 40 discos!
(de oro, por supuesto, en espigas de diamante)

El problema de las Torres de Hanoi



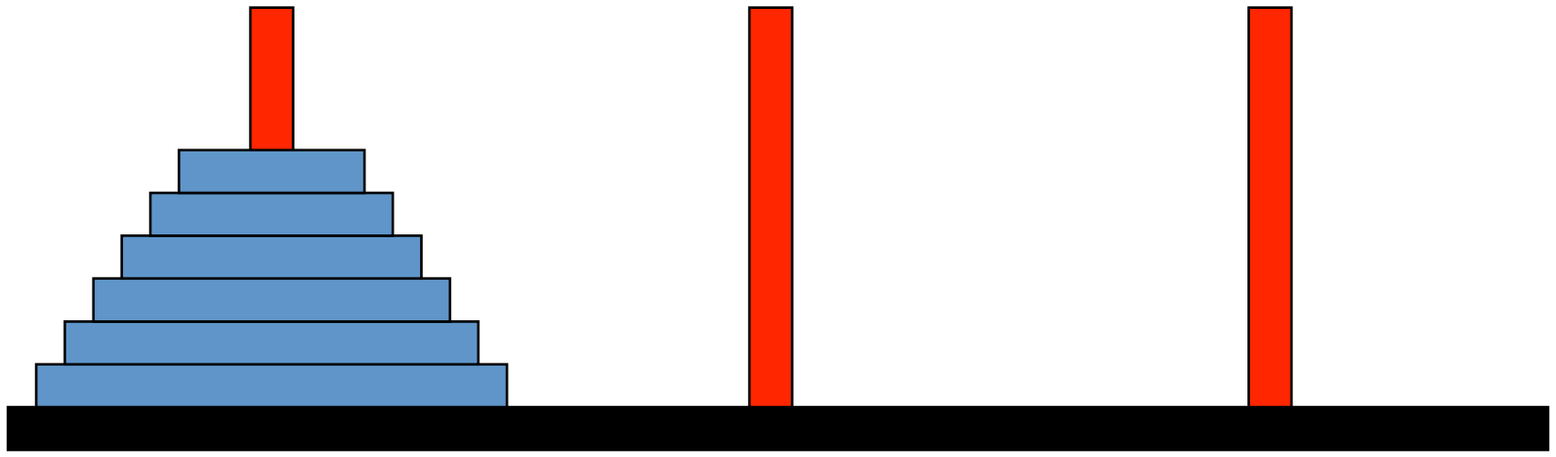
Ver :

http://rodoval.com/heureka/hanoi/

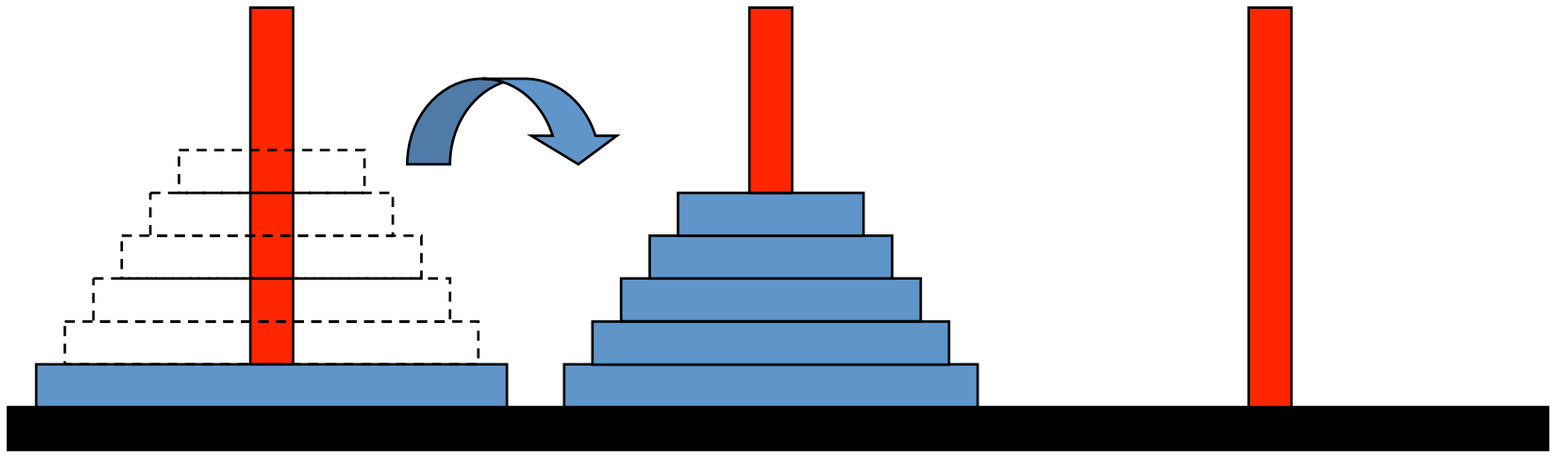
http://www.cut-the-knot.org/recurrence/hanoi.shtml

La leyenda dice que hay que mover 40 discos!
(de oro, por supuesto, en espigas de diamante)

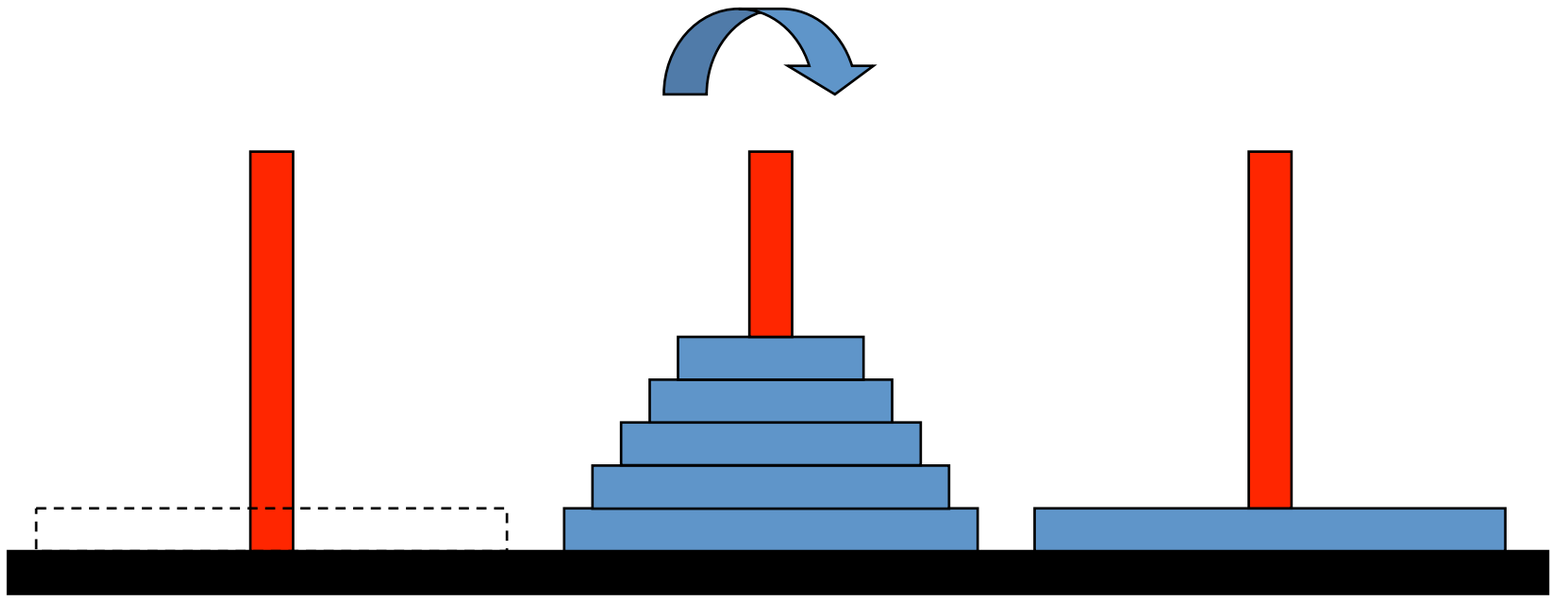
Paso a paso



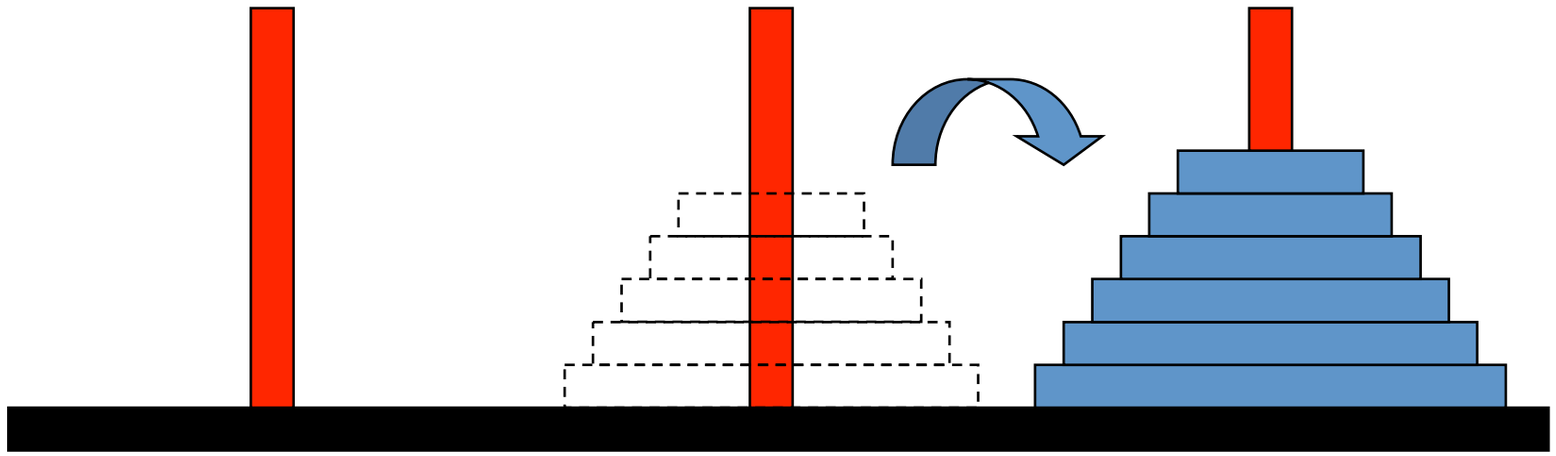
Paso a paso



Paso a paso



Paso a paso



El problema de las Torres de Hanoi

```
void hanoi(int discos, char ini, char dest, char aux) {
    if (discos == 1) cout << ini << " --> " << dest << endl;
    else {
        hanoi(discos - 1, ini, aux, dest);
        cout << ini << " --> " << dest << endl;
        hanoi(discos - 1, aux, dest, ini);
    }
}
```

```
void hanoi(int discos) {
    hanoi(discos, 'A', 'C', 'B');
}
```

El problema de las Torres de Hanoi

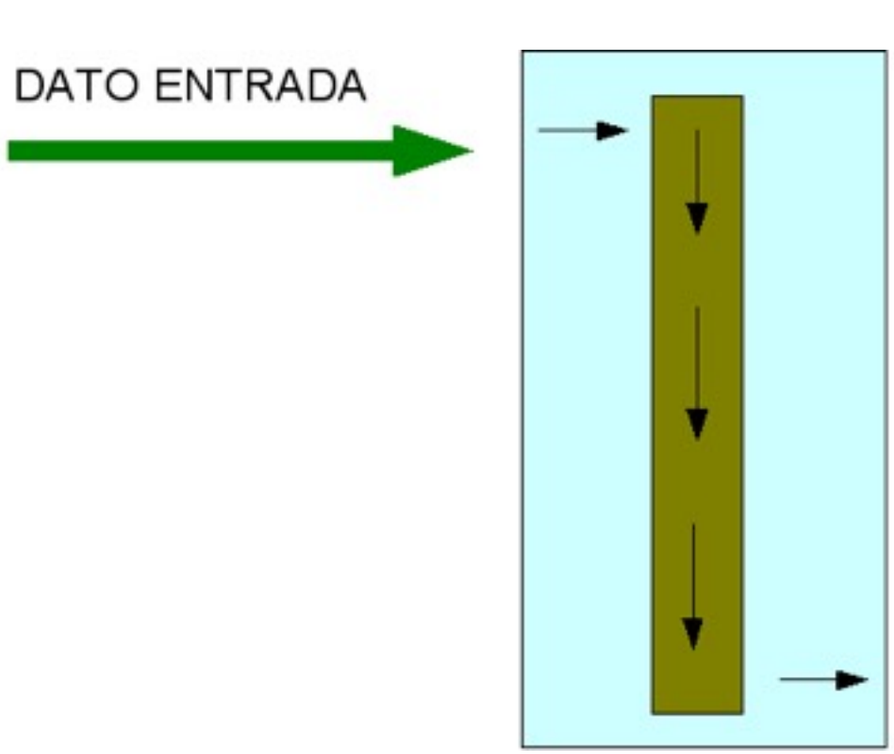
```
void hanoi(int discos, char ini, char dest, char aux) {
    if (discos == 1) cout << ini << " --> " << dest << endl;
    else {
        hanoi(discos - 1, ini, aux, dest);
        cout << ini << " --> " << dest << endl;
        hanoi(discos - 1, aux, dest, ini);
    }
}
```

```
void hanoi(int discos) {
    hanoi(discos, 'A', 'C', 'B');
}
```

Se hacen 2^N-1 movimientos, para 40 discos uno por segundo toma 348 centurias.

FIFO

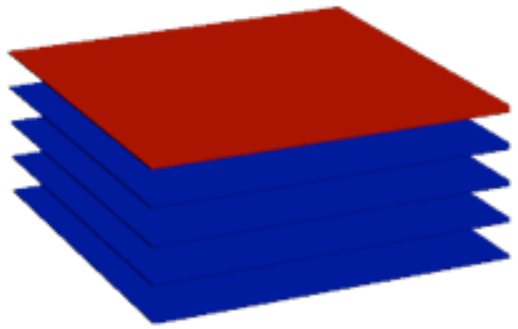
FIFO es el acrónimo inglés de First In, First Out (primero en entrar, primero en salir) o primero en llegar, primero en ser servido.



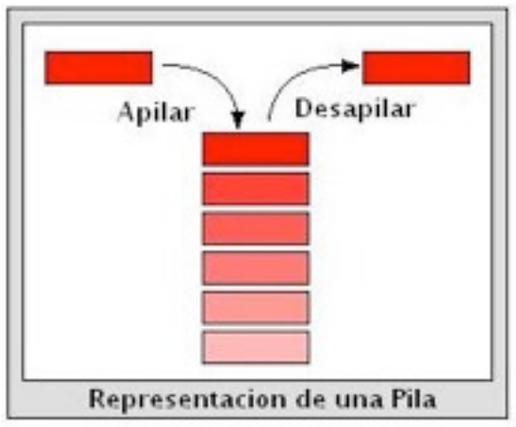

```
class kfifo {
        link head;
        link tail;
        unsigned int len;
    public:
        . . .
};
```

LIFO

El término LIFO es el acrónimo inglés de Last In First Out (último en entrar, primero en salir).
Guarda analogía con una pila de platos, en la que los platos van poniéndose uno sobre el otro, y si se quiere sacar uno, se saca primero el último que se puso.



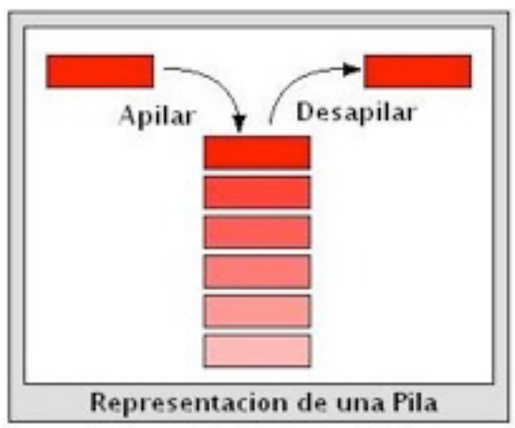
LIFO es el algoritmo utilizado para implementar pilas.



Pilas

Una pila cuenta con 2 operaciones imprescindibles: apilar y desapilar, a las que en las implementaciones modernas de las pilas se suelen añadir más de uso habitual.

- **Crear:** se crea la pila vacía.
- **Apilar:** se añade un elemento a la pila.(push)
- **Desapilar:** se elimina el elemento frontal de la pila.(pop)
- **Cima:** devuelve el elemento que esta en la cima de la pila. (top o peek)
- **Vacía:** devuelve cierto si la pila está vacía o falso en caso contrario.



Representacion de una Pila

Tipo de Dato Abstracto de una pila (*stack*)

```
Class stack{
        lista lista;
    public:
        push(item i);
        pop();
        item top();
        imprime();
    // otros métodos miembro
};
```

Graficar la secuencia del problema de las torres de Hanoi usando (en linux)

```
#include<cstdio>

//positioning in linux and Os X
void gotoxy(int x, int y)
{
    printf("%c[%d;%df",0x1B,y,x);
}


int main()
{

     gotoxy(10,10);
     printf("  IIIIIIIIII\n");
     gotoxy(10,11);
     printf("IIIIIIIIIIII\n");
     return 0;
}
```

Graficar la secuencia del problema de las torres de Hanoi usando (en Windows):

```
#include <cstdio>
#include <windows.h>

void GotoXY(int x, int y) // en windows{
  COORD coord;
  coord.X = x;
  coord.Y = y;
  SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);
}

int main(){
        GotoXY(10,10);
        printf("  IIIIIIIII\n");
        GotoXY(10,11);
        printf("IIIIIIIIIII\n");
        return 0;
}
```

Complejidad de soluciones recursivas

- Ecuaciones de recurrencia

Complejidad de soluciones recursivas

- Ecuaciones de recurrencia

En un algoritmo recursivo, la función T(n) que establece su tiempo de ejecución viene dada por una ecuación E(n) de recurrencia, donde en la expresión aparece la propia función T.

$T(n) = E(n)$, *y en* $E(n)$ *aparece la propia función* T.

Complejidad de soluciones recursivas

- Ejemplo 1, factorial

```
int Fact (int n) {

     if (n <= 1)
          return(1);
     else
          return( n * Fact(n-1));
}
```

Complejidad de soluciones recursivas

- Ejemplo 1, factorial

```
int Fact (int n) {

     if (n <= 1)
          return(1);
     else
          return( n * Fact(n-1));
}
```

$$T(n) = \begin{cases} c_1, & si\ n \leq 1 \\ T(n-1) + c_2, & si\ n > 1 \end{cases}$$

Complejidad de soluciones recursivas

- Ejemplo 1, factorial

```
int Fact (int n) {

     if (n <= 1)
          return(1);
     else
          return( n * Fact(n-1));
}
```

$$T(n) = \begin{cases} c_1, & si\ n \leq 1 \\ T(n-1) + c_2, & si\ n > 1 \end{cases}$$

c_1: tiempo de ejecución del caso trivial n <=1.

Si n>1, el tiempo requerido por Fact puede dividirse en dos partes:

- T(n-1): La llamada recursiva a Fact con n-1.
- c2: El tiempo de evaluar la condición (n>1) y la multiplicación de n*Fact(n-1)

Complejidad de soluciones recursivas

- Estrategia

Sustituir las recurrencias por su igualdad hasta llegar a cierta $T(n_0)$ que sea conocida.

Complejidad de soluciones recursivas

- Estrategia

Sustituir las recurrencias por su igualdad hasta llegar a cierta $T(n_0)$ que sea conocida.

$T(n) = T(n-1) + c_2$

$= (T(n-2) + c_2) + c_2 \quad = T(n-2) + 2*c_2 =$

$= (T(n-3) + c_2) + 2*c_2 \quad = T(n-3) + 3*c_2 =$

....

$= T(n-k) + k *c_2$

Cuando $k = n–1$, tenemos que $T(n) = T(1) + c_2*(n-1)$, y es $O(n)$.

Complejidad de soluciones recursivas

- Estrategia

Sustituir las recurrencias por su igualdad hasta llegar a cierta $T(n_0)$ que sea conocida.

$T(n) = T(n-1) + c_2$ k=1

$= (T(n-2) + c_2) + c_2 \quad = T(n-2) + 2*c_2 =$ k=2

$= (T(n-3) + c_2) + 2*c_2 \quad = T(n-3) + 3*c_2 =$ k=3

....

$= T(n-k) + k *c_2$

Cuando $k = n–1$, tenemos que $T(n) = T(1) + c_2*(n-1)$, y es $O(n)$.

Complejidad de soluciones recursivas

- Ejemplo 2, Como el de las torres de Hanoi

```
int Recursiva1 (int n) {

 if (n <= 1)
    return(1);
 else
    return(Recursiva1(n-1)+ Recursiva1(n-1));
}
```

Complejidad de soluciones recursivas

- Ejemplo 2, Como el de las torres de Hanoi

```
int Recursiva1 (int n) {

 if (n <= 1)
    return(1);
  else
    return(Recursiva1(n-1)+ Recursiva1(n-1));
}
```

$$T(n) = \begin{cases} 1, & si\ n \leq 1 \\ 2 \cdot T(n-1)+1, & si\ n > 1 \end{cases}$$

Complejidad de soluciones recursivas

$$T(n) = \begin{cases} 1, & si\ n \leq 1 \\ 2 \cdot T(n-1) + 1, & si\ n > 1 \end{cases}$$

$T(n) = 2*T(n-1) + 1 =$

$= 2*(2*T(n-2) + 1) + 1 = 2^2*T(n-2) + (2^2-1) =$

. . . .

$= 2^k*T(n-k) + (2^k-1)$

Para $k = n-1$, $T(n) = 2^{n-1}*T(1) + 2^{n-1}-1$, y por tanto $T(n)$ es $O(2^n)$.

Complejidad de soluciones recursivas

$$T(n) = \begin{cases} 1, & si\ n \leq 1 \\ 2 \cdot T(n-1) + 1, & si\ n > 1 \end{cases}$$

$T(n) = 2*T(n-1) + 1 =$ k=1

$= 2*(2*T(n-2) + 1) + 1 = 2^2*T(n-2) + (2^2-1) =$ k=2

. . . .

$= 2^k*T(n-k) + (2^k-1)$

Para $k = n-1$, $T(n) = 2^{n-1}*T(1) + 2^{n-1}-1$, y por tanto $T(n)$ es $O(2^n)$.