# Laboratorio 6

Juan Diego Sique Martínez

Septiembre, 2018

## 1. Primer inciso

El algoritmo que propongo para resolver el problema de las fracciones egipcias es el siguiente.

**Data:** Dos números enteros **n** y **d**
**Result:** Las fracciones egipcias que conforman una fracción
initialization;
f = $\lceil \frac{n}{d} \rceil$ ;
n = n × f − d ;
d = d × d';
Imprimir("1/", f);
**if** $n \neq 0$ **then**
    Imprimir("+") ;
    FracciónEgipcia(n, d);
**end**

**Algorithm 1:** Fracción Egipcia

El algoritmo es codicioso ya que se intenta hacer el mínimo de pasos y utiliza la solución heurística y mejor para el momento para hallar la respuesta esperada, de una manera sobresaliente.

## 2. Segundo inciso

### 2.1. «Algoritmo codicioso»

```
Data: Una estructura de datos A y un entero pMax
Result: La cantidad monetaria máxima a robarse en relación al peso
        máximo acarreable
initialization;
item = Máximo(A, valorUnitario);
if pMax - item.peso > 0 then
    pMax = pMax-item.peso;
    valor = item.valor;
    A.Remover(item);
    return valor + Codicioso(A, pMax) ;
else
    peso = item.peso + (pMax - item.peso) ;
    valor = peso × item.valorUnitario;
    return valor;
end
```

**Algorithm 2:** Codicioso

El algoritmo lo que hace es seleccionar los objetos cuyo valor unitario sea mayor, descuenta el peso del peso máximo que podemos acarrear y entra nuevamente a la función. En caso de no poder seleccionar objetos completos agarra una porción del objeto mejor valuado por unidad.

Se sirve de dos métodos «Máximo» y «Remover». «Máximo» obtiene el objeto con mayor valor unitario de nuestra estructura de datos A. «Remover» elimina un objeto de nuestra colección de datos en base a un índice.

### 2.2. «Programación dinámica»

Éste algoritmo posee la pecularidad de utilizar una estructura de datos auxiliar para guardar cálculos previos o menores que son reutilizados. En ésta ocuasión opté por una lista de longitud en proporción al valor del peso máximo acarreable durante el robo.

```
Data: Una estructura de datos A y un entero pMax
Result: La cantidad monetaria máxima a robarse en relación al peso
        máximo acarreable
initialization;
B = [ ] × 50 ;
for i = 1 to pMax do
    item = Máximo(A, valorUnitario);
    if B.Existe(i-1) then
        B[i] = item.valorUnitario + B[i-1];
        item.peso = item.peso - 1;
        if item.peso = 0 then
            A.Remover(item)
        end
    else
        B[i] = item.valorUnitario × i;
        item.peso = item.peso - 1;
        if item.peso = 0 then
            A.Remover(item)
        end
    end
end
return B[pMax];
```

**Algorithm 3:** Dinámico

El procedimiento se sirve de dos métodos «Máximo», «Existe» y «Remover». «Máximo» obtiene el objeto con mayor valor unitario de nuestra estructura de datos A. «Remover» elimina un objeto de nuestra colección de datos en base a un índice. «Existe» verifica si en el arreglo existe la posición del índice buscado.

En la lista de números llamada **B** los índices van del 1 al valor de **pMax** respectivamente.

En mi opinión, éste algoritmo es ineficiente ya que requiere de una estructura auxiliar y su tiempo de ejecución excede a una implmentación codiciosa para obtener un valor bastante atinado.