PUCP

1INF27

Ejercicios de Pilas y Colas

2024

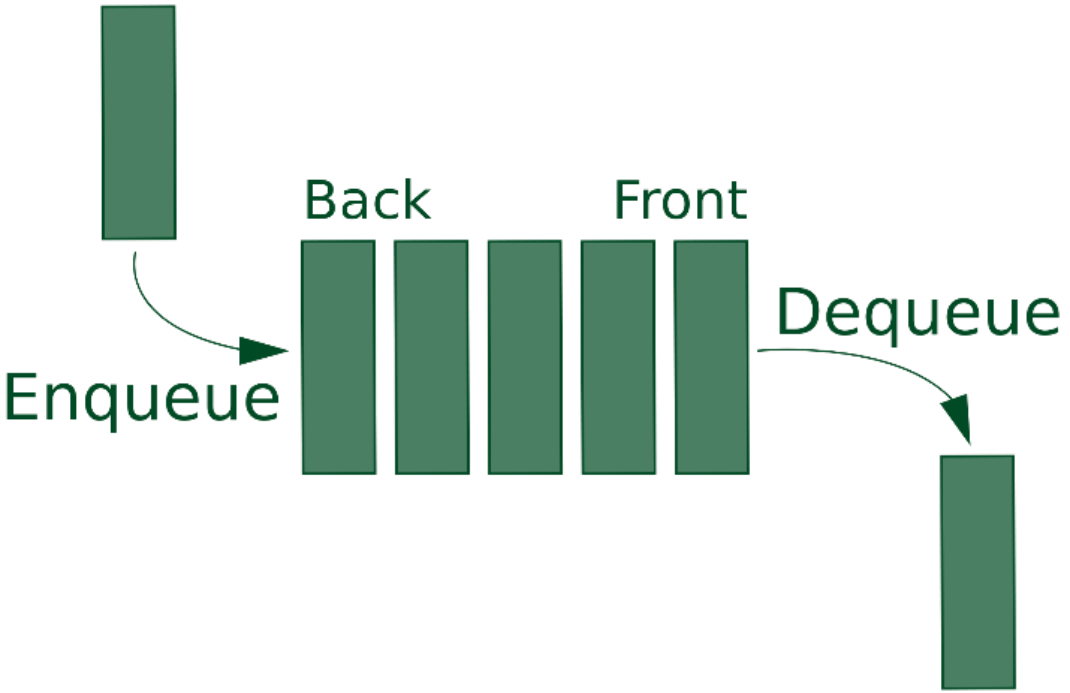
Profesores:

Cueva, R. | **Allasi**, D. | **Roncal**, A. | **Huamán**, F.

0581 | 0582 | 0583 | 0584

COLAS – Operaciones

- **Encolar()**: añade un elemento al final de la cola
- **Desencolar()**: elimina un elemento del principio de la cola
- **Primero()**: examina el elemento situado al principio de la cola
- **EsColaVacia()**: determina si la cola está vacía
- **Tamaño()**: determina el número de elementos de la cola



COLAS

EJERCICIOS

Colas para cifrado

Utilizando una clave repetitiva se puede codificar y decodificar un mensaje. Esta técnica utiliza una lista de valores clave para desplazar los distintos caracteres en diferentes cantidades. Si el mensaje es más largo que la lista de valores clave, simplemente volvemos a comenzar con la lista de claves desde el principio. Por ejemplo, si los valores claves son:

3 1 7 4

Entonces el primer carácter se desplaza tres posiciones, el segundo carácter se desplazará una, etc. Después de desplazar el cuarto carácter cuatro posiciones, comenzamos a utilizar de nuevo la lista.

Colas para cifrado

La figura muestra el mensaje "*hola mundo*" codificado utilizando esta clave repetitiva. Observe que está técnica codifica una misma letra asignándole diferentes caracteres cada vez, dependiendo de cual sea su posición dentro del mensaje y, por tanto, de que valor de clave se utilice para codificarla.

Mensaje codificado:	K p s e # n | r g p
Clave:	3 1 7 4 3 1 7 4 3 1
Mensaje decodificado:	h o l a m u n d o

Colas para cifrado

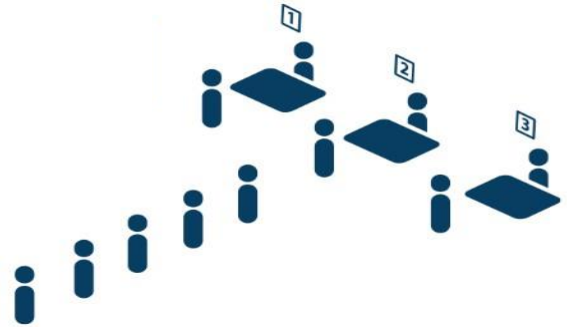
El programa utiliza una clave repetitiva para codificar y decodificar un mensaje. La clave formada por valores enteros se almacena en una cola. Después de utilizar un valor de la clave, se le vuelve a poner al final de la cola, para que la clave se repita continuamente según sea necesario para los mensajes de gran longitud. La clave utiliza valores positivos como negativos.

```
COLAS PARA CIFRADO
Codificado: kpse#n|rgp
Decodificado: hola mundo
```

Simulación de banco

Imagina que en un banco los clientes hacen fila para ser atendidos por un solo cajero. Cada cliente tiene un tiempo de servicio, que representa cuánto tiempo tardará el cajero en atender a ese cliente. El programa debe simular este proceso usando una cola, donde los clientes ingresan a la fila y se atienden en orden de llegada (FIFO).

- Cada cliente tiene un ID y un tiempo de servicio (en minutos).
- Los clientes ingresan a la cola y son atendidos por el cajero uno por uno.
- El programa debe mostrar el tiempo total de espera de los clientes, así como el tiempo que cada cliente tuvo que esperar hasta ser atendido.

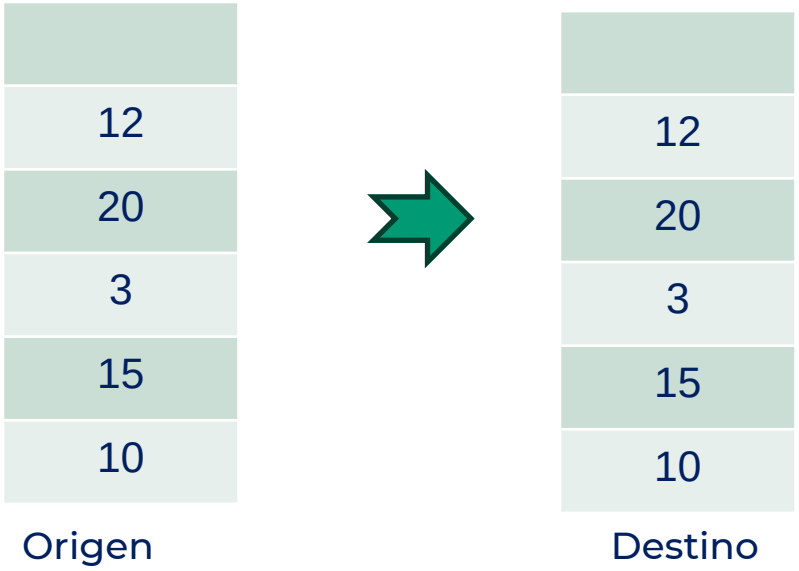


PILAS

EJERCICIOS

Pasar elementos manteniendo orden de ubicación

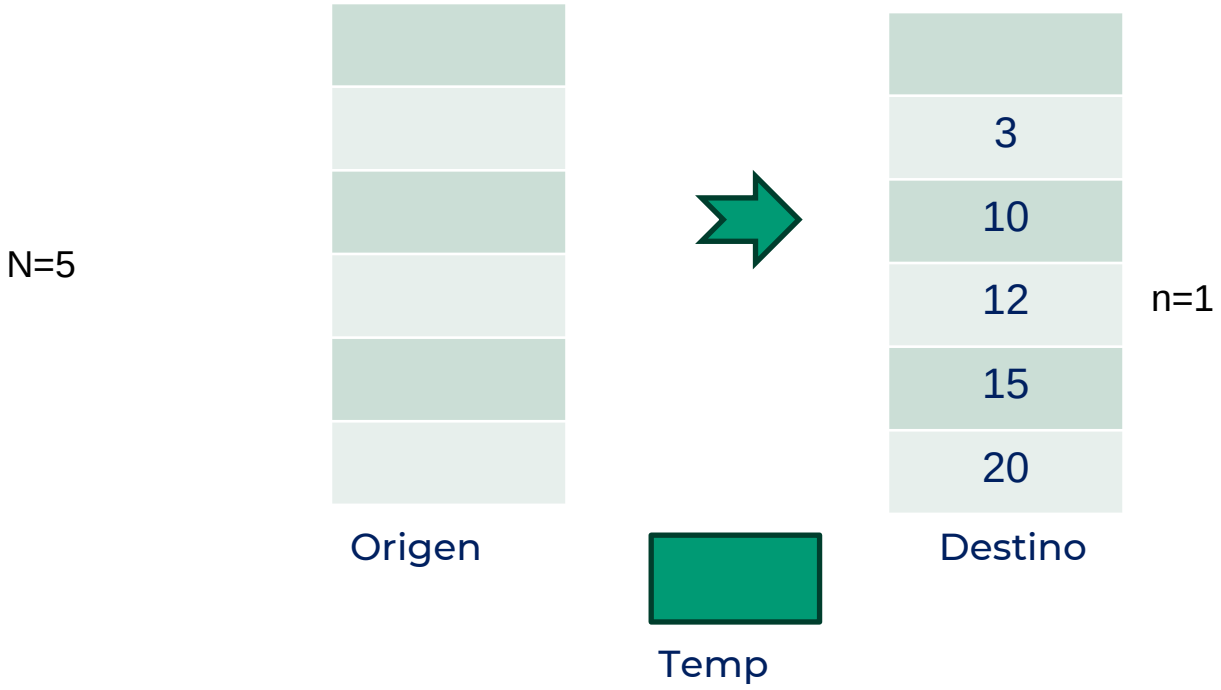
Pasar los elementos de una pila origen a una pila de destino manteniendo el mismo orden de ubicación.




```
PasarPila(Pila &origen, Pila &destino)
-----
  While not estaVacia(origen)
     temp = despilar(origen)
     n=0
     While(not esVacia(origen))
          apilar(destino, temp)
          temp = despilar(origen)
          n++
     for(i=1 … n)
        apilar(origen, desapilar(destino))
     apilar(destino, temp)
```

Pasar elementos ordenando en el destino

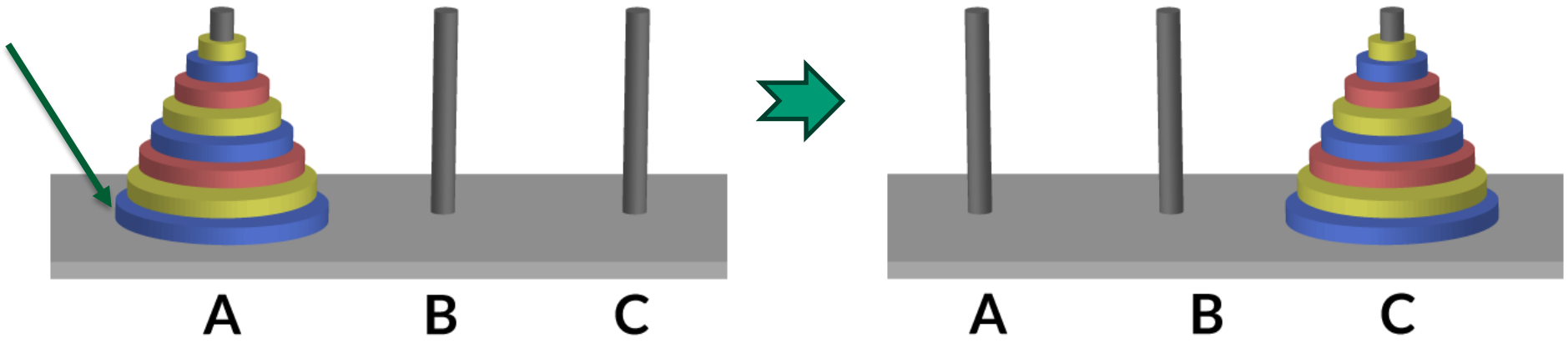
Pasar los elementos de una pila origen a una pila destino ordenando de forma ascedente.



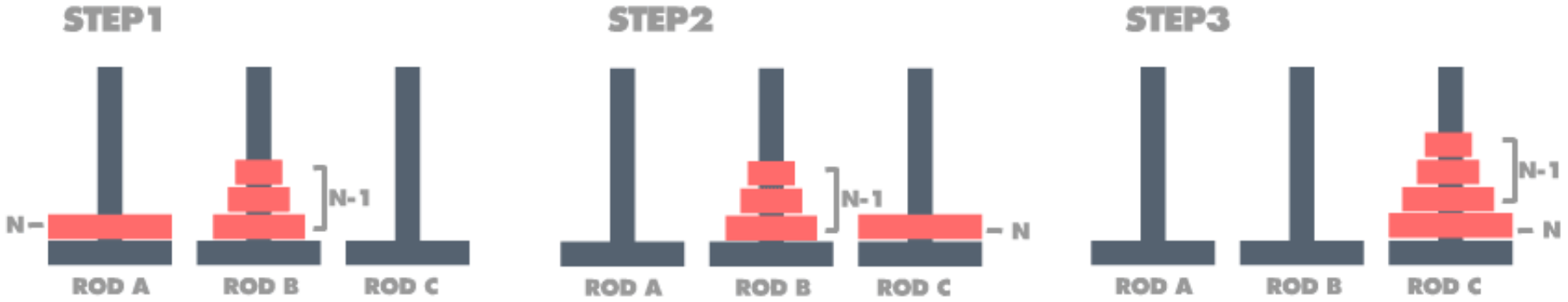

```
Ordenar(Pila &origen, Pila &destino)
----
While not estaVacia(origen)
    temp = despilar(origen)
    n=0
    While(not esVacia(destino) and cima(destino) < temp)
        apilar(origen, desapilar(destino)
        n++
    apilar(destino, temp)
    For(i=1…n)
      apilar(destino, desapilar(origen)
```

Torres de Hanoi con Pilas

Resolverr las torres de Hanói usando tres pilas: Pila de origen (A), Pila auxiliar (B) y Pila de destino (C). El objetivo es pasar desde la pila A a la pila C utilizando la pila B como paso intermedio.

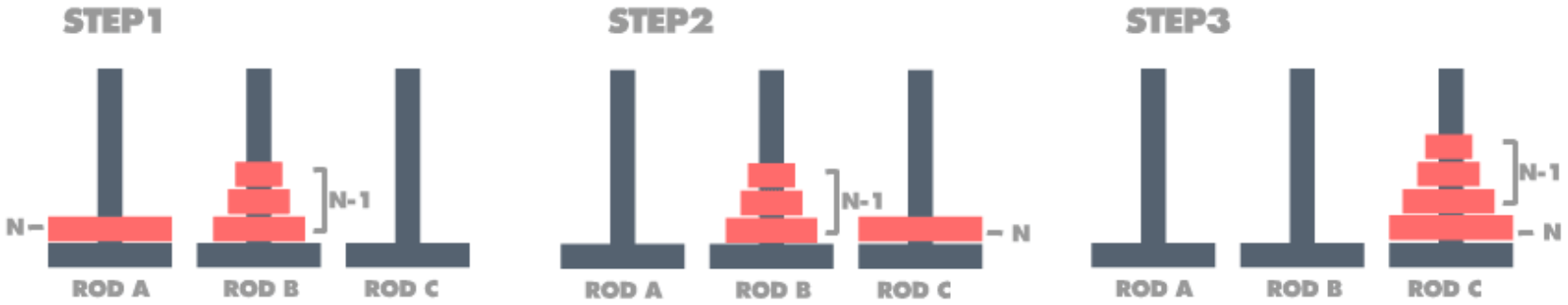


Torres de Hanoi con Pilas




```
Hanoi(Pila &Origen, Pila &Destino, Pila &Auxiliar, int N)
    if N==0:
        return
    //Pasar N-1 cajas desde el origen(A) al auxiliar(B), pero bajos las propiedades de Hanoi
    Hanoi(Origen, Auxiliar, Destino, N-1)
    // Pasar la caja N desde el origen(A) al destino(C)
    if not esVacia(origen):
        apilar(Destino,  desapilar(Origen) )
    //Pasar N-1 cajas desde el auxiliar(A) al destino(B), pero bajos las propiedades de Hanoi
    Hanoi(Auxiliar, Destino, Origen, N-1)
```

Torres de Hanoi con Pilas



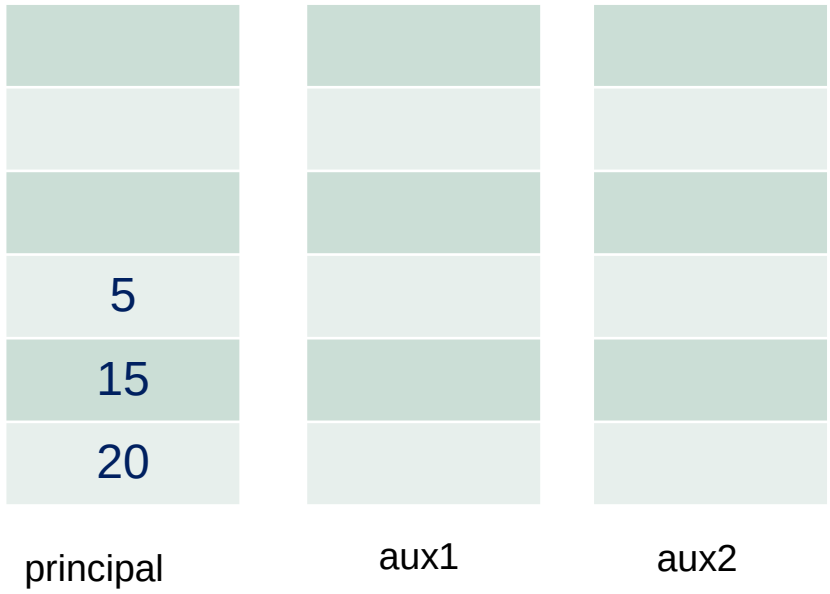
Main()

Pila Origen, Destino, Auxiliar
apilar(Origen, 1)
apilar(Origen, 2)
apilar(Origen, 3)
apilar(Origen, 4)
Hanoi(Origen,Destino,Auxiliar, 4)
mostrarPila(Origen)
mostrarPila(Destino)

Transporte de productos frágiles

Reto:

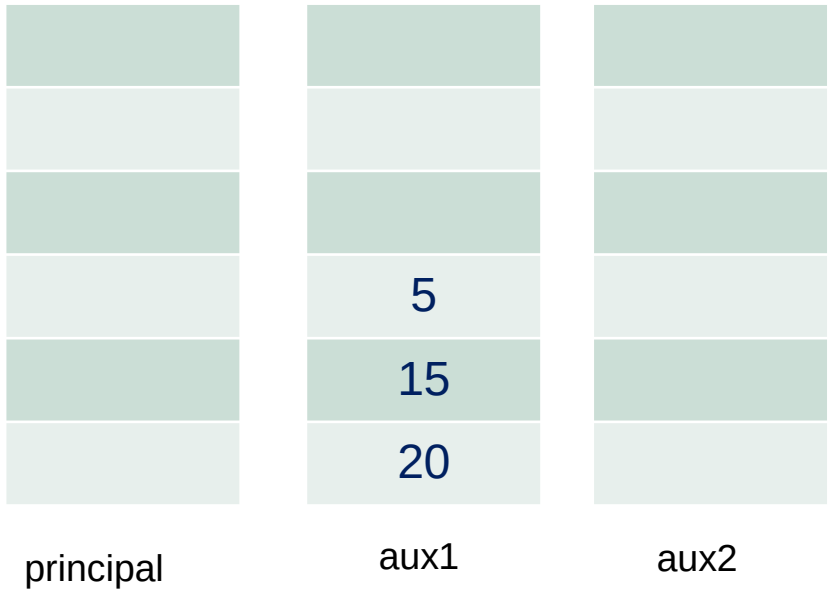
Pasar el elemento **12** a la pila principal manteniendo en todo tiempo la **restricción del peso** (no puede estar un elemento grande encima de un pequeño)



Transporte de productos frágiles

Paso 1:
---
pasar todos los elementos de la pila principal a la pila aux1 utilizando aux2 como pila auxiliar.

Hanoi(principal, aux1, aux2, 3)

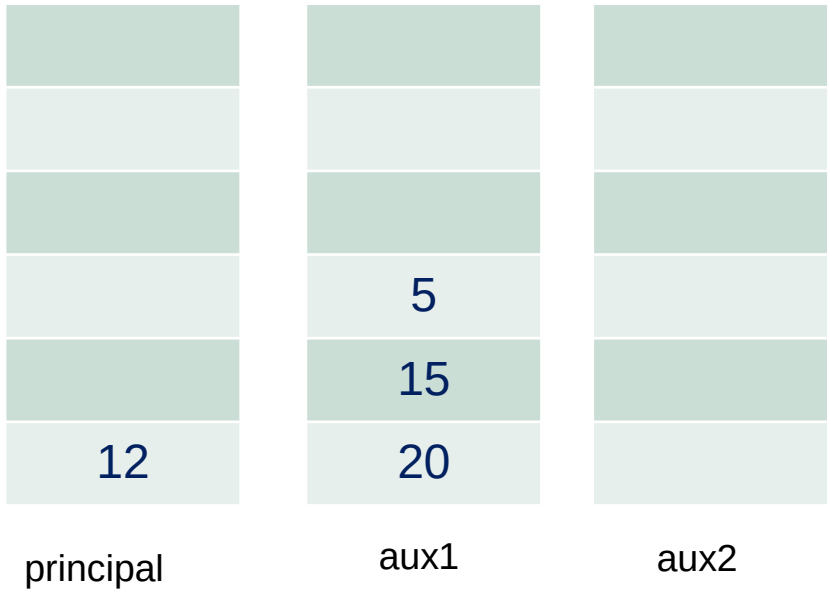


Transporte de productos frágiles

Paso 2:

Mover el nuevo paquete a la pila principal

apilar(principal, 12)

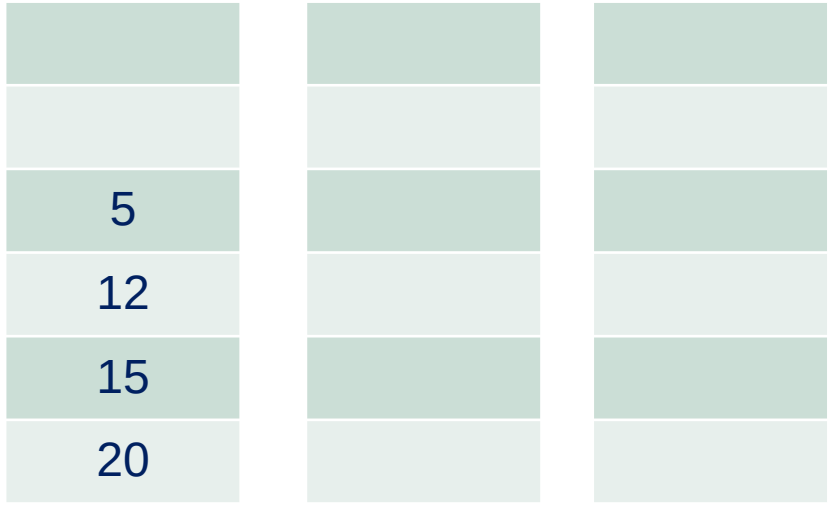


Transporte de productos frágiles

Paso 3:

Pasar de la pila aux1 al principal utilizando aux2 como temporal, **pero al momento de mover se debe validar la cima de la pila destino.**

Hanoi(aux1, principal, aux2, 3)



Transporte de productos frágiles

Paso 3:

```
Hanoi(Pila &origen, Pila &destino, Pila &auxiliar, int N)
    if N==0:
        return
    Hanoi(origen, auxiliar, destino, N-1)
    if not esVacia(origen):
        Mover(origen, destino)
    Hanoi(auxiliar, destino, origen, N-1)
```

```
Mover(Pila &origen, Pila &destino)
   elem1 = desapilar(origen)
   if not esVacia(destino):
       elem2 = desapilar(destino)
       if elem1 > elem2:
           apilar(destino, elem1)
           apilar(destino, elem2)
       else:
           apilar(destino, elem2)
           apilar(destino, elem1)
    else:
      apilar(destino, elem1)
```