4. Pilas (TDA e implementación con arreglos)

Carlos Zerón Martínez

Universidad Nacional Autónoma de México

zeron@ciencias.unam.mx

Martes 3 de Noviembre de 2020

Introducción

El término pila (*Stack* en inglés) se relaciona con cosas de la vida cotidiana como por ejemplo:

- Pila de platos en la cocina
- Pila de libros en un escritorio

Cuando colocamos algo en la pila lo hacemos hasta arriba y cuando quitamos algo de la pila también.

Propiedad LIFO (*Last In First Out*): El último objeto colocado en la pila será el primer objeto que salga de ahí.

El lugar donde se hacen inserciones y eliminaciones de objetos lo denominaremos el **tope** de la pila.

Características de la pila

La pila es una estructura de datos posicional, homogénea y dinámica, como la lista, que además cuenta con estas características:

- Las operaciones de inserción, eliminación y consulta se pueden hacer en el tope de la pila. Opcionalmente, se puede admitir la consulta en cualquier posición, pero es menos frecuente.
- Es posible saber cuantos elementos tiene la pila
- Es posible determinar si la pila es vacía

Algunas aplicaciones de las pilas

Navegadores de internet

Los navegadores almacenan las direcciones de los sitios visitados recientemente en una pila. Cada vez que un usuario visita un sitio nuevo, la dirección se inserta en la pila de direcciones. El navegador entonces permite al usuario obtener las direcciones visitadas usando el botón del navegador que sirve para retroceso.

Editores de texto

Usualmente proporcionan un mecanismo para deshacer alguna operación de edición, es decir, cancela las operaciones recientes y revierte el documento a estados anteriores. Este mecanismo puede ser acompañado manteniendo los cambios de texto en una pila.

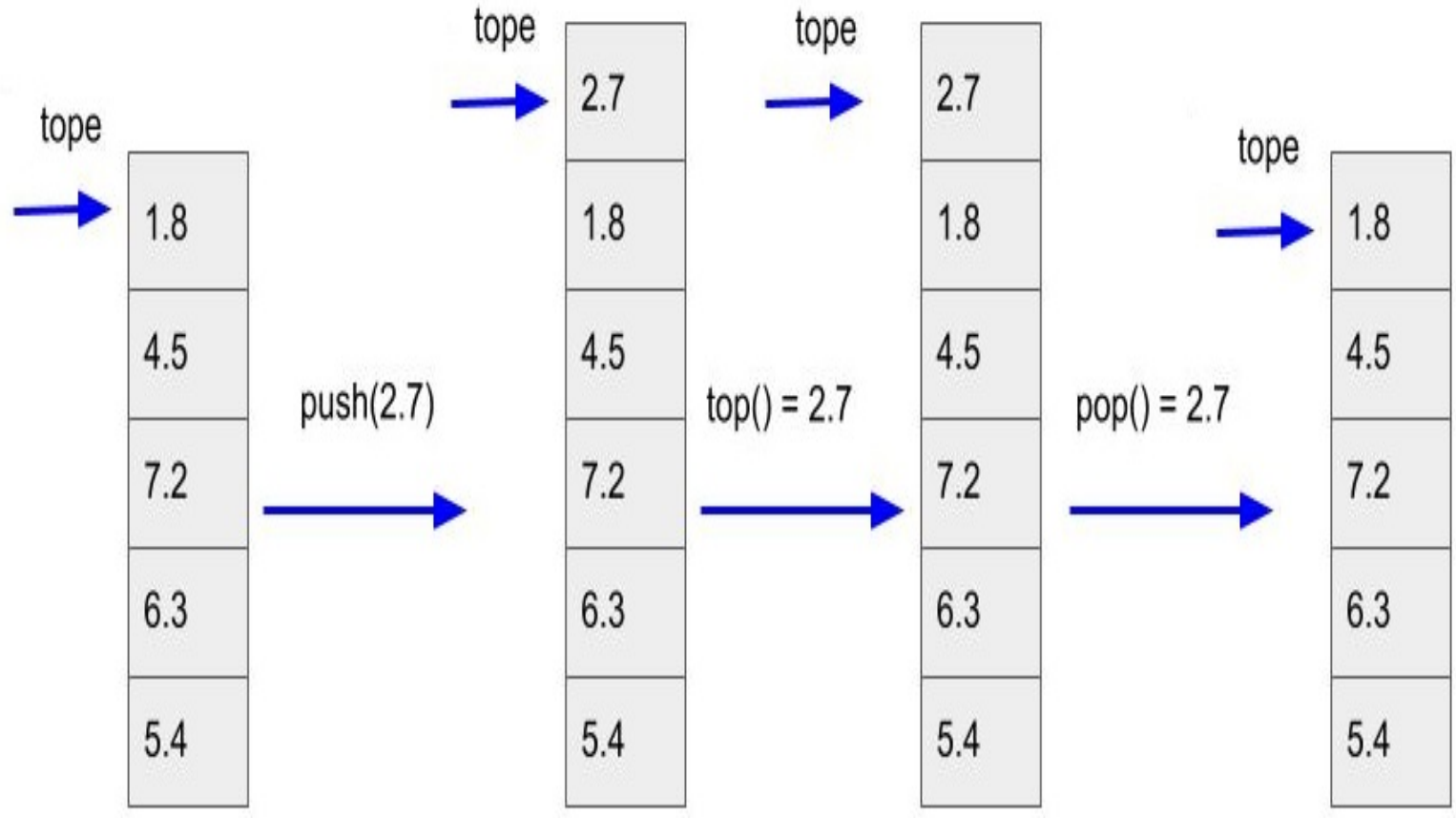
Especificación del TDA Pila

Datos: Objetos (una cantidad entera no negativa de ellos)
Operaciones:

- isEmpty() : $\{V, F\}$. Devuelve un valor lógico que indica si la pila es vacía (no tiene elementos) o no (si tiene elementos).
- size() : $\mathbb{N} \cup \{0\}$. Devuelve el número de elementos de la pila (tamaño).
- push(*obj*) : $\varnothing$. Inserta el objeto *obj* en el tope de la pila.
- pop() : *Objeto*. Devuelve y elimina el objeto en el tope de la pila. Regresa un objeto nulo si la pila es vacía.
- peek() : *Objeto*. Devuelve el objeto que se encuentra en el tope de la pila. Regresa un objeto nulo si la pila es vacía. (nombre alternativo para la operación top())

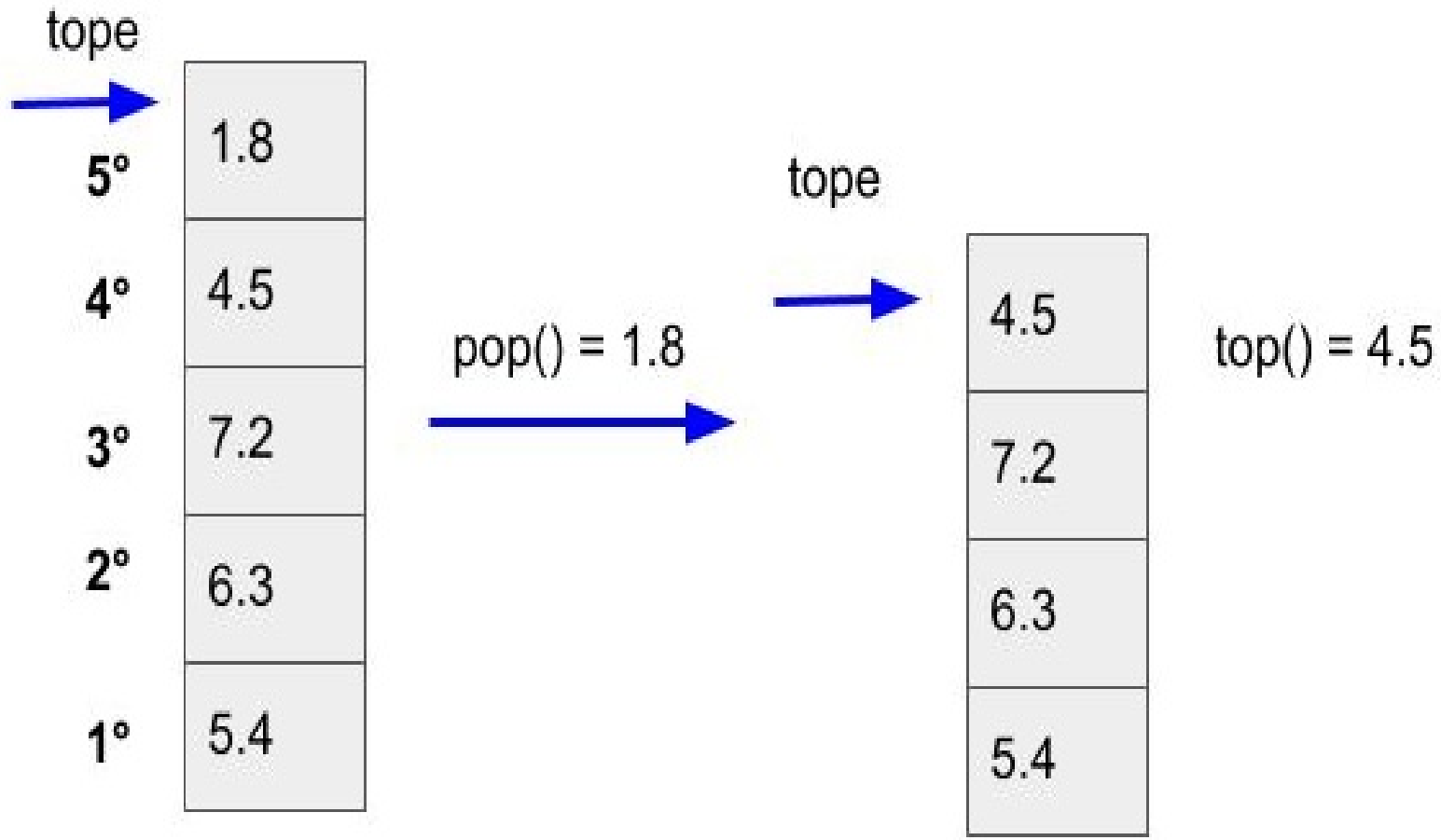
Axiomas

La inserción de un objeto en el tope de la pila incrementa el tamaño de la pila en una unidad y si se efectúa un acceso posterior al tope para eliminación o devolución, entonces se accede al objeto recién insertado.



Axiomas

La eliminación de un objeto del tope de la pila decrementa el tamaño de la pila y el elemento del tope es aquel elemento que fue insertado inmediatamente antes del objeto eliminado de la pila.



Interfaz para el TDA Pila

```
public interface Pila {

    /**
     * Indica si la pila es vacía o no.
     */
    boolean isEmpty();

    /**
     * Devuelve el número de elementos de la pila.
     */
    int size();

    /**
     * Inserta el objeto dado en el tope de la pila.
     */
    void push(Object obj);
```

Interfaz para el TDA Pila

```
    /**
     * Devuelve y elimina el objeto que se encuentra
     * en el tope de la pila. Null si la pila es vacía.
     *
     * @return El objeto ubicado en el tope.
     */
    Object pop();

    /**
     * Devuelve el objeto que se encuentra en el
     * tope de la pila. Null si la pila es vacía.
     *
     * @return El objeto ubicado en el tope.
     */
    Object peek();

}
```

Implementación de una pila basada en un arreglo

Mantenemos una variable que indique el índice del arreglo en el cual se encuentra el tope de la pila. Cuando la pila sea vacía, ese indice se hace igual a -1, para no apuntar a ninguna posición válida de la pila.

Como el número de elementos que puede almacenar un arreglo es limitado, se lanzaría una excepción *RuntimeException* en el método push si se intenta una inserción con la pila llena.

Usamos una constante que indique el máximo de elementos.

Implementación de una pila basada en un arreglo

Para determinar si la pila es vacía, es suficiente con verificar si el tope apunta a un índice inválido, como es el -1 que se asigna inicialmente; si apunta a un índice que sea cero o positivo, entonces la pila no es vacía.

Cuando la pila no es vacía, el tope va a apuntar a la posición más grande que está ocupada por un objeto no nulo y puesto que su primer índice válido es 0, hay que sumarle ese valor para poder determinar el tamaño de la pila.

Implementación de una pila basada en un arreglo

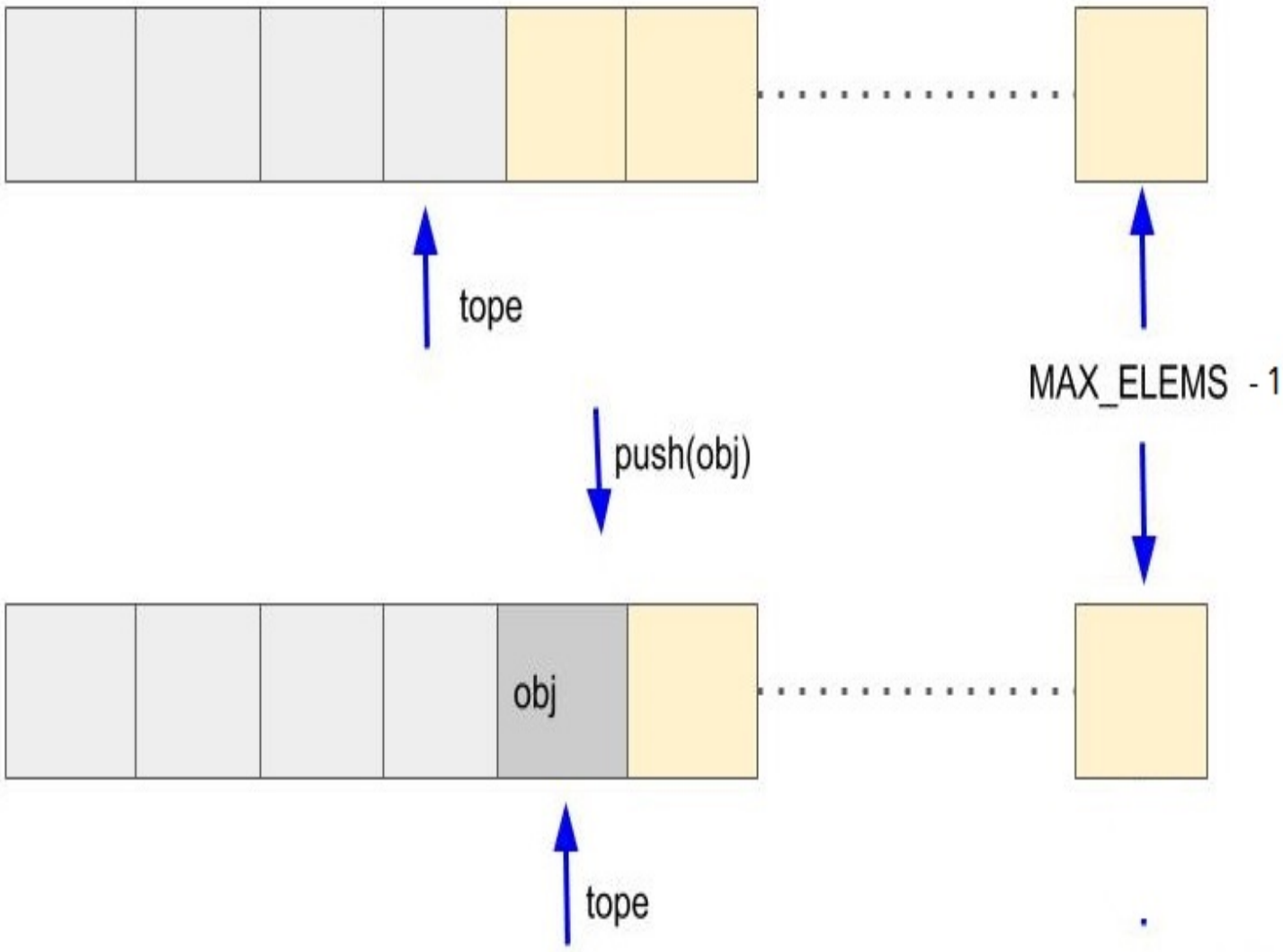
```
public class PilaArreglo implements Pila {

    private static final int MAX_ELEMS = 1000;

    private Object elementos[];

    private int tope;

    /**
     * Construye una pila basada en un arreglo
     * con un número máximo de elementos por
     * omisión.
     */
    public PilaArreglo() {
        elementos = new Object[MAX_ELEMS];
        tope = -1;
    }

    @Override
    public boolean isEmpty() {
        return tope == -1;
    }

    @Override
    public int size() {
        return tope + 1;
    }
```

Implementación del método push

```
@Override
public void push(Object obj) {
    if (size() == elementos.length)  {
        throw new RuntimeException("Pila llena. No hay espacio");
    }
    tope++;
    elementos[tope] = obj;
}
```

Implementación del método push

Aquí se ilustra el caso en que es posible realizar la inserción por disponibilidad de espacio en el arreglo



Implementación del método peek

El operador condicional tiene como sintaxis:

$$condicion\ ?\ expresion1\ :\ expresion2;$$

Si se cumple *condicion* se ejecuta la *expresion*1, si no se cumple, se ejecuta la *expresion*2. Es útil para la asignación de valores de dos posibilidades o para el retorno de valores, como en este caso.

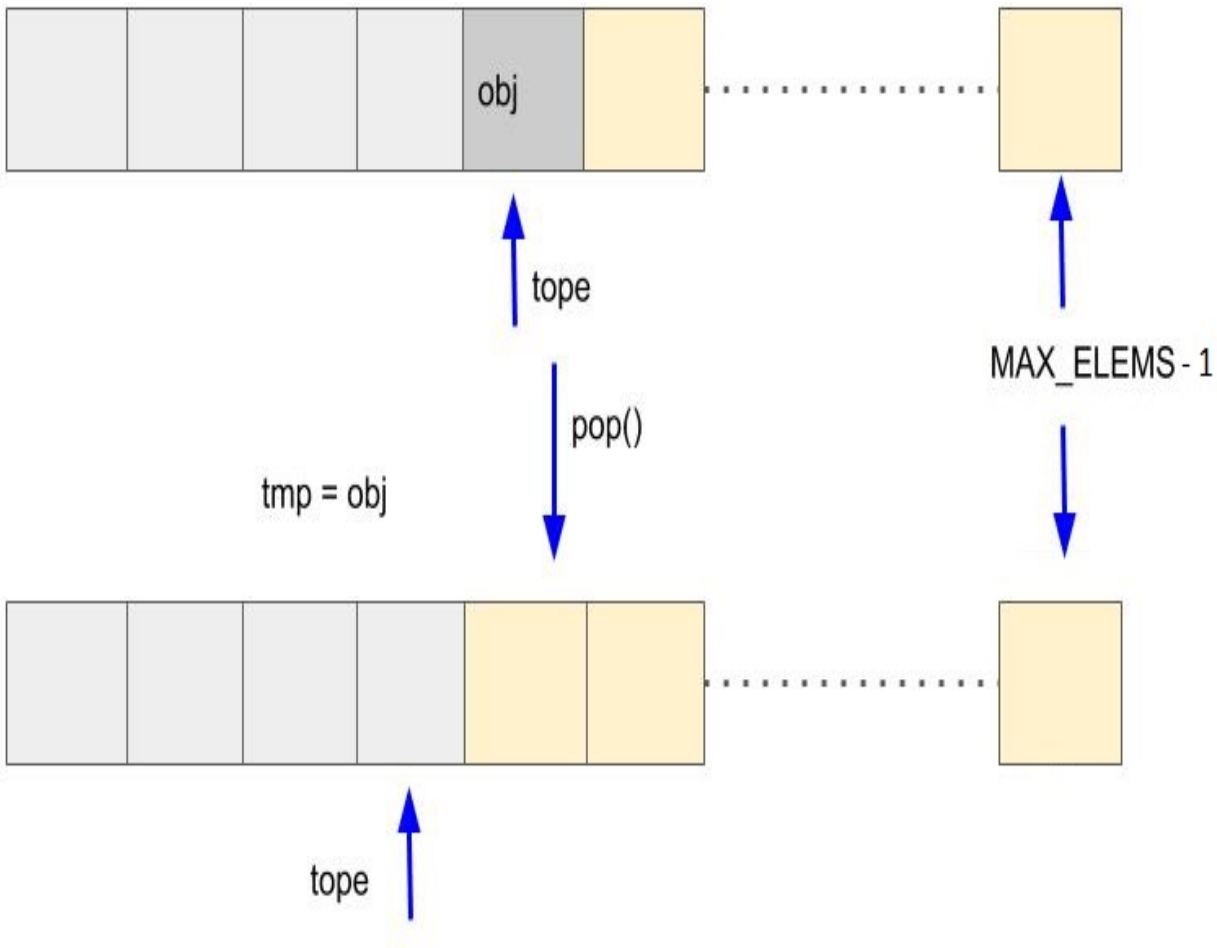
```
@Override
public Object peek() {
    return isEmpty() ? null : elementos[tope];
}
```

Implementación del método pop

```
    @Override
    public Object pop() {
        if (isEmpty()) {
            return null;
        }
        Object tmp = elementos[tope];
        elementos[tope] = null;
        tope--;
        return tmp;
    }
}
```

Implementación del método pop

Aquí se ilustra el caso en que es posible eliminar el elemento del tope. Se rescata en un objeto temporal, se nulifica la localidad donde se encontraba, se modifica el índice del tope y se regresa el objeto temporal.



Análisis de complejidad

Por el acceso directo que soportan los arreglos y dado que el número de operaciones elementales no depende de la entrada, tenemos una implementación eficiente en la que todas las operaciones del TDA Pila tienen complejidad $O(1)$.