UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria Informàtica

etsinf

Tema 5. Tipos de datos lineales

Programación (PRG)

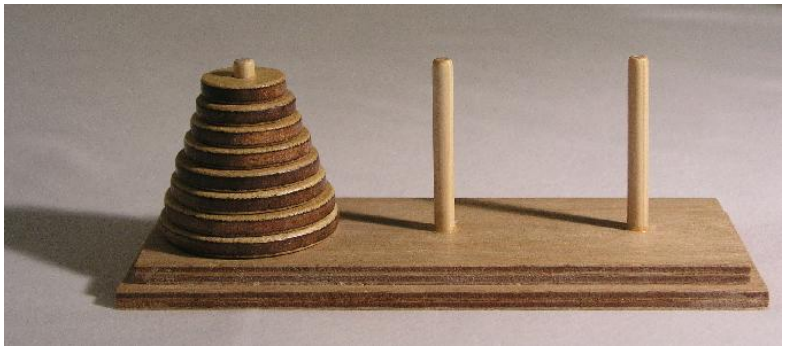
Jorge González Mollá

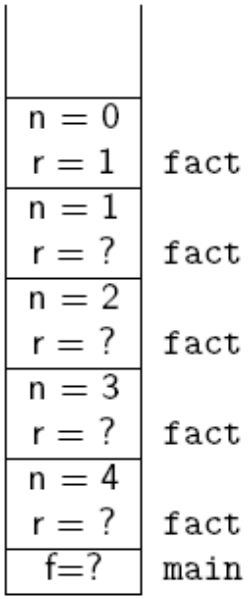
Departamento de Sistemas Informáticos y Computación

Índice

Definición

- Una *pila* (*stack*) es un tipo de secuencia de datos del mismo tipo cuya política de acceso se basa en un criterio *LIFO* (*L*ast *I*n *F*irst *O*ut).
  - Los elementos apilados en una estructura de tipo pila siempre se extraerán de ella en orden inverso al que fueron insertados, por lo que el último elemento en entrar será el primero en salir, y viceversa, el primer elemento en entrar será el último en salir.
- Ejemplos de pilas:





Interfaz de operaciones (API)

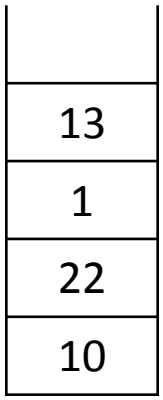
Operación	Descripción
public **Stack**()	Crea una nueva **Stack** vacía
public void **push**(Tipo x)	Apila el dato en la **Stack**
public Tipo **pop**()	Desapila el dato de la cima de la **Stack** y lo devuelve *
public Tipo **peek**()	Devuelve (sin desapilarlo) el dato de la cima de la **Stack** *
public boolean **isEmpty**()	Devuelve `true` si la **Stack** está vacía y `false` en caso contrario
public int **size**()	Devuelve el nº de datos de la **Stack** (≥0)

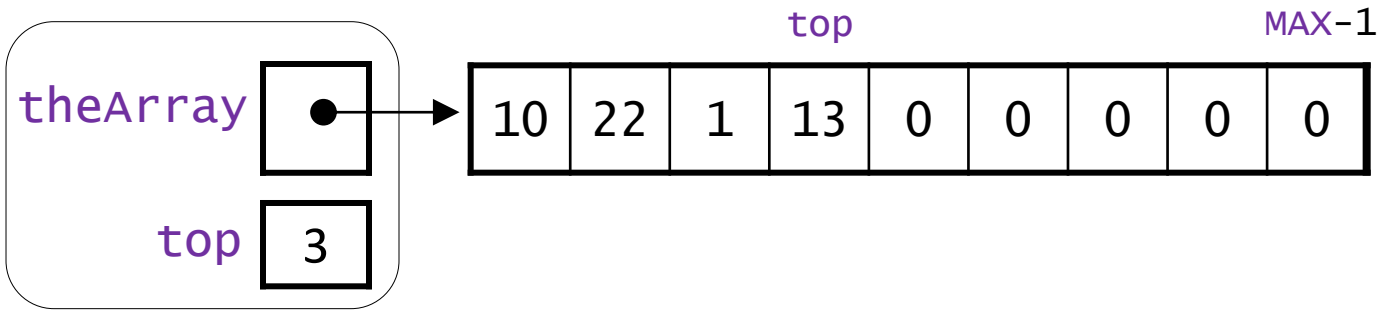
* Lanza `NoSuchElementException` si la **Stack** está vacía.

Se puede conseguir implementar estas operaciones con coste constante.

Implementación mediante arrays

- Una pila se puede implementar utilizando un array (`theArray`) que almacene los datos, un índice (`top`) que marque el punto de acceso a la pila y una constante que defina la dimensión máxima del array (`MAX`). Presentamos una pila cuyos elementos son de tipo `int`:






```
public class StackIntArray {
   private int[] theArray;
   private int top;
   private static final int MAX = ...;

   // Implementación de las operaciones:
   ...
}
```

Implementación mediante arrays

- Constructor `StackIntArray`: Crea el array e inicializa el punto de acceso a -1, indicando que la pila está vacía.

```
public StackIntArray() {
    theArray = new int[MAX];
    top = -1;
}
```

- Operación `push`:

```
public void push(int x) {
    if (top + 1 == theArray.length) {
        duplicateArray(); }
    else {
        top++;
        theArray[top] = x;
    }
}
```

Implementación mediante arrays

- Operaciones `pop` y `peek`:

```
public int pop() {
   if (top < 0) { throw new NoSuchElementException(); }
   int x = theArray[top]; top--; return x;
}
public int peek() {
   if (top < 0) { throw new NoSuchElementException(); }
   return theArray[top];
}
```

- Operación `isEmpty`:

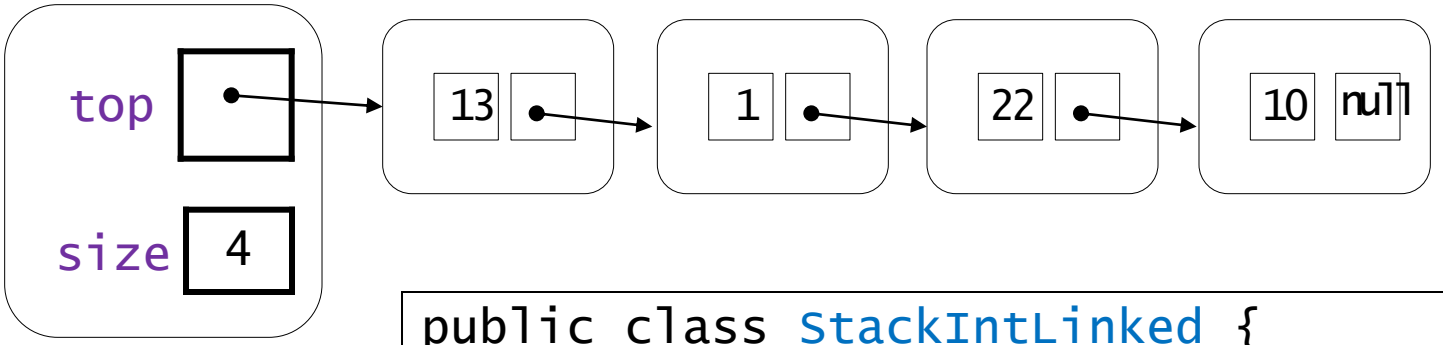
```
public boolean isEmpty() { return top == -1; }
```

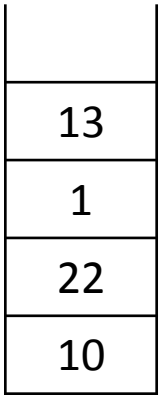
- Operación `size`:

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

Implementación enlazada

- En la implementación enlazada de una pila, se considera que la cima de la pila debe ser el primer nodo de la secuencia y se representa mediante un atributo llamado `top`. Además, se añade un atributo `size` que almacena el número de elementos de la pila.






```
public class StackIntLinked {
    private NodeInt top;
    private int size;

    // Implementación de las operaciones:
    ...
}
```

Implementación enlazada

- Constructor `StackIntLinked`: Asigna `null` a la referencia `top` e inicializa `size` a 0.

```
public StackIntLinked() {
    top  = null;
    size = 0;
}
```

- Operación `push`:

```
public void push(int x) {
    top = new NodeInt(x, top);
    size++;
}
```

Implementación enlazada

- Operaciones pop y peek:

```
public int pop() {
   if (size == 0) { throw new NoSuchElementException(); }
   int x = top.data;
   top   = top.next;
   size--;
   return x;
}


public int peek() {
   if (size == 0) { throw new NoSuchElementException(); }
   return top.data;
}
```

Implementación enlazada

- Operación `isEmpty`:

```
public boolean isEmpty() {
    return top == null;
    // o return size == 0;
}
```

- Operación `size`:

```
public int size() {
    return size;
}
```

Comparación de implementaciones

- La complejidad temporal de todas las operaciones en las dos representaciones estudiadas es independiente de la talla del problema: $\Theta(1)$.
- En cuanto a la complejidad espacial, la implementación con arrays presenta el inconveniente de la estimación del tamaño máximo del array y la reserva de espacio que en muchos casos no se utilizará.
- Este consumo adicional de espacio no tendrá demasiada importancia si el tipo de las componentes de `theArray` es relativamente pequeño, como es el caso de una pila de `int`, o de pilas de objetos (las componentes del array son entonces referencias).
- Por otro lado, la representación enlazada requiere un espacio de memoria adicional para los enlaces.