Pilas y Colas

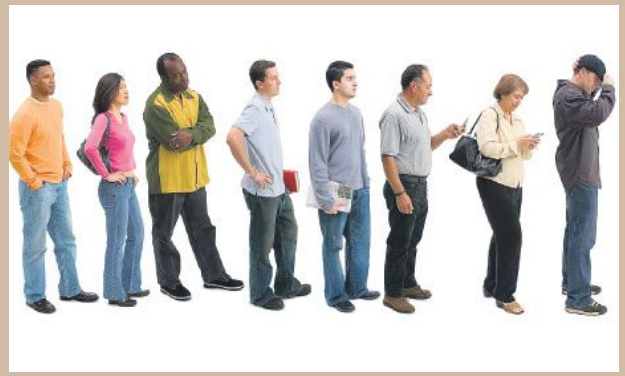
TDA - Colas (Queues)

Otra forma de llamarlas es Listas FIFO (First In First Out).

Los elementos se sacan de la Cola en el mismo orden en que entraron.

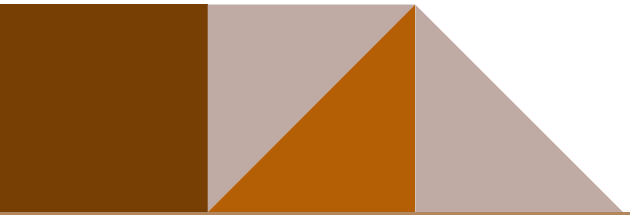
Operaciones en Colas

- Encolar (enqueue). Agregar un elemento.
- Desencolar (dequeue). Sacar el elemento que lleva más tiempo en la Cola.



Pregunta

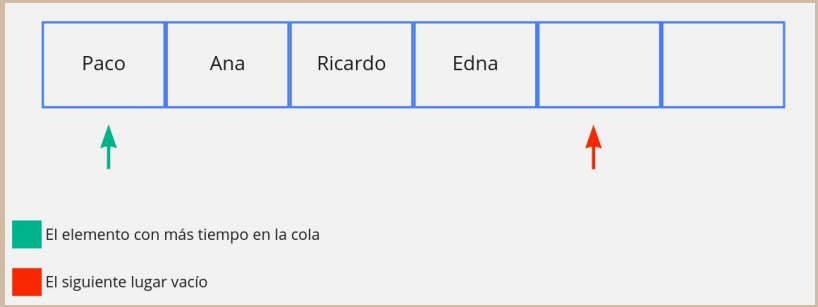
¿En qué casos nos interesaría usar una Cola?



Implementar Colas con Arreglos

Se necesita:

- Un arreglo
- Dos índices:
  - Uno para el elemento con más tiempo en la cola
  - Uno para el siguiente lugar vacío



Implementar Colas con Listas

Necesitamos una lista que nos permita ingresar elementos por un lado y que los saque por el otro, es decir, que soporte las operaciones:

- Agregar al inicio
- Eliminar el último elemento

O bien:

- Agregar al final
- Eliminar el primer elemento

TDA - Pilas (Stacks)

Otra forma de llamarlas es Listas LIFO (Last In First Out).

Los elementos se sacan de la Pila en orden inverso al que entraron.

Operaciones en Pilas

- Meter (push). Agregar un elemento.
- Sacar (pop). Sacar el elemento que lleva menos en la Pila.

* Tope (top). Inspeccionar el elemento en el tope de la pila sin sacarlo.

Ejemplo de Aplicación

Dada una expresión aritmética en post-orden con solo operadores binarios, podemos proceder a la evaluación utilizando una pila y el siguiente algoritmo:

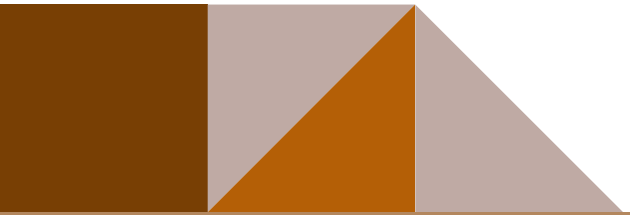
1. Recibimos una expresión en post-orden.
2. Mientras procesamos la expresión:
   a. Procesamos al siguiente elemento e en la expresión.
   b. Si e es operando, lo metemos a la pila.
   c. Si e es operador, sacamos 2 elementos de la pila, evaluamos y el resultado lo metemos a la pila.
3. Se devuelve el valor en el tope de la pila.

Prueba evaluando las expresiones:

- 3 • 10 + 4 → 3 10 • 4 +
- (2 + 5) • (9 - 5) → 2 5 + 9 5 - •
- (27 - (10 + 2 • 8)) • 2 → 27 10 2 8 • + - 2 •

Ejercicio

Crea un algoritmo que utilice pilas para verificar que una expresión formada solo por paréntesis esté correctamente balanceada.



Implementar Pilas con Arreglos

Se necesita:

- Un arreglo
- Un índice:
  - Uno para el último elemento agregado

Implementar Pilas con Listas

Necesitamos una lista que nos permita ingresar elementos por un lado y que los saque por ese mismo lado, es decir, que soporte las operaciones:

- Agregar al inicio
- Eliminar el primer elemento

O bien:

- Agregar al final
- Eliminar el último elemento

Pila de Ejecución

En Java cada hilo de ejecución tiene asociado una **pila de ejecución**.

Durante la ejecución del hilo, cada que se manda a llamar un método se monta en la pila un **registro de activación**. Una vez que finaliza el método se desmonta el último registro y continúa con la ejecución del método asociado al registro de activación en el tope.

En cada registro de activación se guardan las variables locales del método, los argumentos con los que se invocó y la dirección de retorno.

Ejemplo

```
85    public static int duplica(int x){
86      int dos = 2;
87      int duplicado = dos * x;
88      return duplicado;
89    }
90
91    public static int cuadruplica(int x){
92      int duplicado = duplica(x);
93      int cuadruplicado = duplica(duplicado);
94      return cuadruplicado;
95    }
96
      Run | Debug
97    public static void main(String[] args){
98        int original = 5;
99        int calculado = cuadruplica(original);
100   }
```

duplica, línea 87

x = 10
dos = 2
duplicado = 20

cuadruplica, línea 92

x = 5
duplicado = 10
cuadruplicado = 20

main, línea 99

args = [ ]
original = 5
calculado = 20

Recursión

Para resolver ciertos problemas es común encontrar funciones recursivas, esto es, funciones que suelen llamar a sí mismas para devolver el resultado esperado.

```
factorial (n):
    if (n < 0) return -1
    if (n == 0) return 1
    return n * factorial (n - 1)
```

Un algoritmo recursivo posee un algoritmo iterativo equivalente y viceversa.

Diseñar métodos recursivos suele dar soluciones más elegantes que sus equivalentes iterativas.

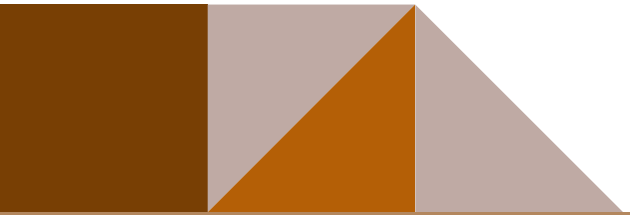
La mayor desventaja de usar recursión es que tiene un impacto en el espacio de ejecución S(n), pues con cada llamada recursiva se monta un nuevo registro de activación en la pila.

En Java suele ocurrir un StackOverflow cuando un método recursivo está mal implementado o cuando se usa con valores muy grandes

Ejercicio

Calcula la complejidad en tiempo y espacio de la siguiente implementación de factorial:

```
factorial (n):
    if (n < 0) return -1
    if (n == 0) return 1
    return n * factorial (n - 1)
```



Ejercicio

Calcula la complejidad en tiempo y espacio de la siguiente implementación para obtener el i-ésimo término de fibonacci:

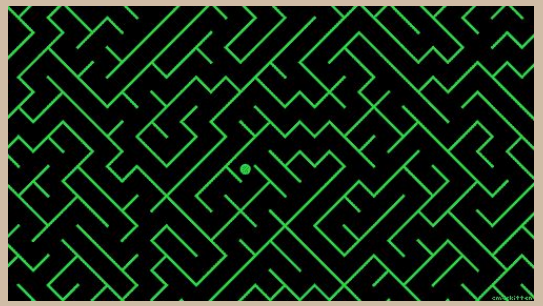
```
fibonacci (n):
    if (n < 0) return -1
    if (n == 0 || n == 1) return n
    return fibonacci (n - 1) + fibonacci (n-2)
```

Backtracking

Es la exploración de una secuencia de soluciones parciales hasta encontrar la solución total o descubrir que no hay.

Se trata de ir armando una solución a un problema y, en caso de ser necesario, regresar a un estado previo para tratar de armar una nueva solución.

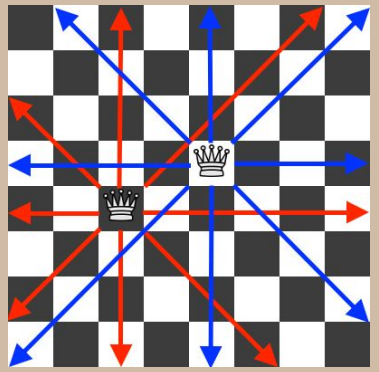
Ejemplo de Aplicación



Dado un laberinto de n x m casillas, donde una casilla es la casilla de inicio y otra es la meta, podemos usar el siguiente algoritmo para dar con una solución. Para ello necesitaremos una pila.

1. Decidir el orden en que se explorarán las casillas (Ej. primero ↑, luego →, luego ↓ y finalmente ←).
2. Nos ubicamos en la casilla de inicio y la marcamos como visitada.
3. *Mientras no hayamos llegado a la meta*:
    a. Desde la casilla en que nos encontramos, intentar movernos en la siguiente dirección posible a una casilla no visitada.
        i. *Si llegamos a una casilla no visitada*, entonces metemos a la pila la casilla actual junto con la dirección en que nos movimos, marcamos la nueva casilla como visitada y nos movemos a ella.
        ii. *Si no se puede llegar a una casilla no visitada*, marcamos a la casilla actual como no visitada, sacamos el siguiente elemento de la pila y nos ubicamos en esa casilla.
            1. *Si ya no hay elementos en la pila*, entonces no hay solución y desistimos.
4. Vaciar el contenido de la pila para obtener el camino solución.

Lo más usual es hacer backtracking con ayuda de una pila, o bien, hacerlo de forma recursiva, lo que implica el crecimiento de la pila de ejecución.



El problema de las 8 reinas consiste en buscar un acomodo para 8 reinas en un tablero de ajedrez sin que se amenacen entre sí.

También se puede hacer backtracking de forma iterativa y sin una pila.

Otras estructuras relacionadas

Deque (Double-ended queue). *Se pronuncia deck*. Es una cola en la que podemos sacar y agregar elementos por uno u otro lado.

Steque. Es una pila en la que además de hacer pop y push, nos permite encolar elementos.

Pregunta

¿Qué estructuras usarías para implementar el juego Snake?