UNIVERSIDAD DE GRANADA

Departamento de Ciencias de la Computación e Inteligencia Artificial

Práctica 3: TDAs Lineales. Pilas y Colas

(Práctica puntuable)

Dpto. Ciencias de la Computación e Inteligencia Artificial
E.T.S. de Ingenierías Informática y de Telecomunicación
Universidad de Granada

Estructuras de Datos

Grado en Ingeniería Informática
Doble Grado en Ingeniería Informática y Matemáticas
Doble Grado en Ingeniería Informática y ADE

1.- Introducción

1.1 Tipos de datos abstractos lineales

Los tipos de datos abstractos lineales se componen de una secuencia de elementos, a_0, a_1, ..., a_{n-1} dispuestos en una dimensión. Las estructuras de datos lineales más relevantes son:

- Vectores 1-d
- Listas
- Pilas (LIFO)
- Colas (FIFO)

Los nombres LIFO y FIFO hacen referencia a cómo se accede a los datos. Así, tenemos LIFO (last input first output), que hace referencia a las colecciones en las que el último elemento en entrar es el primero en salir. Por otro lado, la política FIFO (first input first output) hace referencia a las colecciones en las que el primero en entrar es el primero en salir. Esta es la principal diferencia que existe entre las pilas y colas.

1.2 Colas

Cuando utilizamos una política FIFO, indicamos que las inserciones se realizan en la posición inmediatamente posterior al final de la estructura (**back**), y que los borrados y consultas de valores se hacen en la posición primera de la estructura (**front**). Esta es la política de acceso que implementan las colas.

De esta manera, si tenemos la secuencia de elementos 1, 2, 3, y vamos a almacenar dicha información en una cola, insertaremos los elementos siempre por la posición **back**, como se observa en la figura 1:

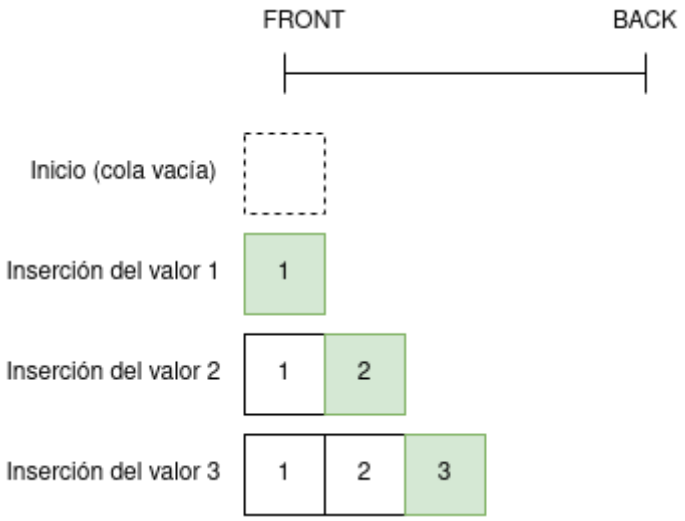

Figura 1: Procedimiento de inserción de valores en la cola

Cuando se consulta o se extrae un dato en la cola, siempre lo haremos por la posición del frente, por lo que no podremos acceder a los elementos que están detrás del frente hasta que no eliminemos dicho valor. En la figura 2 se puede observar el procedimiento de eliminación de valores de una cola FIFO:

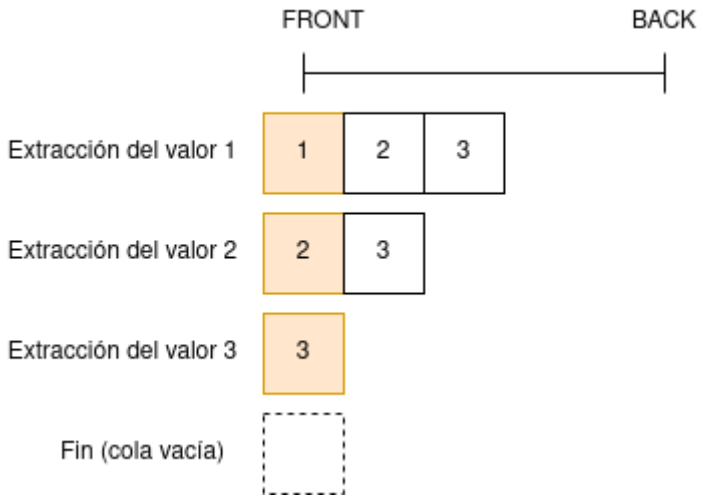

Figura 2: Procedimiento de extracción de valores en la cola

Una estructura de tipo cola deberá contar, como mínimo, con los siguientes métodos para su correcto funcionamiento:

- Colocar un elemento al final de la cola
- Consultar el elemento al principio de la cola (sin extraerlo)
- Extraer el elemento al principio de la cola
- Consultar el tamaño de la cola (número de elementos que la componen)
- Consultar si la cola está vacía

Se puede consultar la documentación del TDA queue, disponible en la librería standard de C++ (https://www.cplusplus.com/reference/queue/queue/), para ver la funcionalidad básica de una cola (aunque dicho TDA implementa adicionalmente algunas funciones que, a pesar de no ser imprescindibles, facilitan el uso de la estructura en muchos contextos).

1.2 Pilas

Cuando utilizamos una política LIFO, indicamos que las inserciones y las consultas de valores se realizan en la primera posición de la estructura. Esta es la política de acceso que implementan las pilas.

Mientras que la representación de las colas es bastante intuitiva en horizontal, y tiene sentido tener las posiciones **front** y **back** diferenciadas, en la pila sólo podremos interactuar con una posición, ya que insertamos y extraemos por el mismo punto. Por lo tanto, podemos ver la pila como una estructura en vertical, donde vamos poniendo y quitando elementos de la parte superior. Por este motivo, la posición de inserción y extracción de las pilas se suele denominar **top**. En la figura 3 se puede observar el procedimiento de inserción y extracción de la secuencia 1, 2, 3 en una estructura de tipo pila:

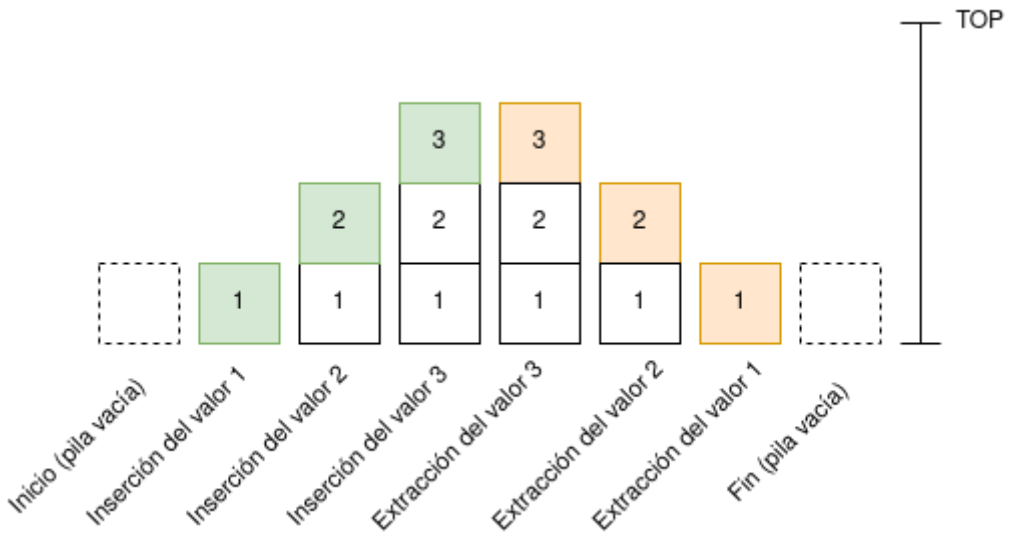

Figura 3: Procedimiento de inserción y extracción de elementos en una pila

Una estructura de tipo pila deberá contar, como mínimo, con los siguientes métodos para su correcto funcionamiento:

- Colocar un elemento en la parte superior de la pila
- Consultar el elemento en la parte superior de la pila (sin extraerlo)
- Extraer el elemento en la parte superior de la pila
- Consultar el tamaño de la pila (número de elementos que la componen)
- Consultar si la pila está vacía

Se puede consultar la documentación del TDA stack, disponible en la librería standard de C++ (https://www.cplusplus.com/reference/stack/stack/), para ver la funcionalidad básica de una pila (aunque dicho TDA implementa adicionalmente algunas funciones que, a pesar de no ser imprescindibles, facilitan el uso de la estructura en muchos contextos).

2. T.D.A's MaxStack y MaxQueue

En esta práctica trabajaremos con dos TDA abstractos, que implementarán una cola y una pila. Estas dos estructuras serán capaces de almacenar la siguiente información:

- En cada posición, un número entero
- Adicionalmente, nos permitirán almacenar el valor máximo en la estructura en cada momento.

Para poder dotar a nuestras estructuras de esta funcionalidad, y como en cada caso sólo se nos permite consultar un elemento (el **top** de la pila o el **front** de la cola), lo que almacenaremos en cada posición es un par de dos enteros (construiremos un struct para ello), uno con el valor actual, y otro con el valor máximo. De esta forma, podemos saber tanto el valor que se encuentra en la posición actual, como el valor máximo que hay almacenado en la estructura.

Para el caso de la pila con máximo, si el valor que vamos a insertar es menor que el máximo que había hasta ese momento, mantenemos como máximo el valor previo, y si no, sustituimos dicho máximo por el valor actual. En la imagen 4 se puede observar el procedimiento de inserción de la secuencia 1, 2, 1, 3, 2, 4:

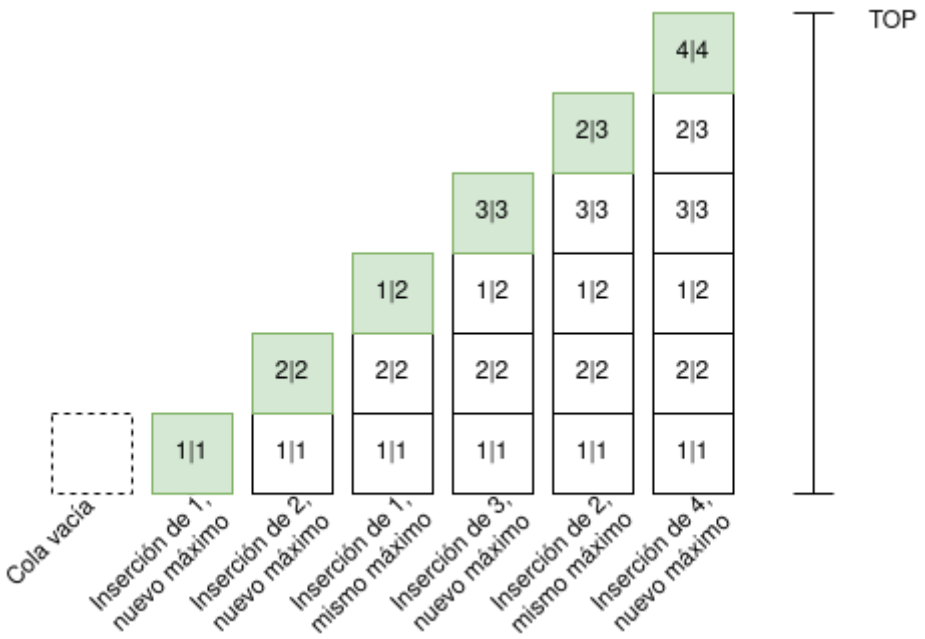

Figura 4: Procedimiento de inserción de elementos en la pila cuando se cuenta con un máximo

Como podemos observar, mantener el valor máximo almacenado en la pila resulta bastante sencillo con este procedimiento, ya que el valor máximo en la pila no varía en el momento de la inserción y en el momento de la extracción (cuando se inserta un valor en una pila, los elementos restantes son los mismos que cuando dicho valor se retira, ya que los elementos que le hayan caído encima posteriormente no estarán en el momento de la inserción, y los que tiene por debajo no habrán podido ser modificados hasta que el elemento en cuestión no salga).

El procedimiento de inserción de elementos en la cola con máximo (MaxQueue) es ligeramente más complejo que el que nos encontramos en la estructura anterior. Hay que tener en cuenta que nos interesa saber cuál es el valor máximo almacenado en la estructura **en el momento de sacar un elemento**. Mientras que en la pila este máximo era estático para cada posición, en la cola no ocurre lo mismo. Esto se debe a que en el momento de la inserción, hay elementos en la cola que no van a estar presentes en el momento de la extracción del elemento actual, y en cambio podrían haberse insertado elementos nuevos. Esto nos tiene que hacer pensar que la forma de calcular el máximo en las dos estructuras va a ser ligeramente distinto, aunque podemos mantener la misma idea de guardar en cada posición de la cola el valor actual y el máximo en la estructura.

En cuanto a la implementación de la información que se almacena en cada elemento de la estructura, podemos optar por un struct como el siguiente:

```
struct element {
      int value; // Current value to store
      int maximum; // Current max value in the structure
}
```

Conviene sobrecargar el operador << para el tipo de dato `element`, para simplificar la impresión por pantalla de elementos de este tipo.

Práctica a realizar

1.- Ejercicio obligatorio: Construcción del TDA MaxStack

En el primer ejercicio de la práctica, se propone la implementación del **tipo de dato MaxStack**, que implementa la cola con máximo que hemos descrito en los apartados anteriores. Para ello, será necesario:

1. Dar una especificación del T.D.A MaxStack.
2. Definir el conjunto de operaciones necesarias para el funcionamiento de la pila junto con sus especificaciones.
3. Implementar el TDA MaxStack **utilizando como contenedor subyacente una cola.** Aunque el TDA que estamos implementando tenga que comportarse como una pila, utilizaremos como estructura de datos interna un contenedor de tipo cola. Se podrá utilizar la cola que viene implementada en la STL (`queue`). Teniendo en cuenta los principios de **ocultamiento de información**, se recomienda separar la declaración y la implementación de dicha clase (se sugieren los nombres de archivo `maxstack.h` y `maxstack.cpp`, situados en la carpeta que les corresponda)
4. Comprobar el funcionamiento del T.D.A MaxStack utilizando el programa que se proporciona en la práctica. Dicho programa itera sobre los argumentos de entrada que acompañan a la ejecución, haciendo lo siguiente:
    a. Si el elemento es numérico, lo inserta en la pila
    b. Si el elemento es un punto, saca de la pila un elemento

El código de dicho programa está implementado en el archivo `pila_max.cpp`, que se os proporciona comentado para evitar problemas de compilación. Una vez implementado el TDA, se debe descomentar este código para comprobar el funcionamiento. El código del programa es el que sigue:

```
int main (int argc, char ** argv) {
      MaxStack stack;
      for (int i=1; i<argc; i++) {
            char * v = argv[i];
            if (v[0] == '.'){
                  cout << stack.top() << endl;
                  stack.pop();
            }
            else{
                  int vint = atoi(v);
                  stack.push(vint);
            }
      }
}
```

Por ejemplo, dado el programa anterior, y dada la entrada siguiente:

```
      ./build/pila_max 1 2 . 3 4 . . .
```

La salida del programa debe ser la siguiente:

```
2,2
4,4
3,3
1,1
```

2.- Ejercicio opcional: Construcción del TDA MaxQueue

Como parte opcional de la práctica, propone la creación, especificación y documentación del **tipo de dato MaxQueue**, de forma análoga a como se construye el tipo de dato MaxStack del ejercicio anterior:

1. Dar una especificación del T.D.A MaxQueue
2. Definir el conjunto de operaciones necesarias para el funcionamiento de la cola junto con sus especificaciones
3. Implementar el TDA MaxQueue **utilizando como contenedor subyacente una pila.** Aunque el TDA que estamos implementando tenga que comportarse como una cola, utilizaremos como estructura de datos interna un contenedor de tipo pila. Se podrá utilizar la pila que viene implementada en la STL (`stack`). Teniendo en cuenta los principios de **ocultamiento de información**, se recomienda separar la declaración y la implementación de dicha clase (se sugieren los nombres de archivo `maxqueue.h` y `maxqueue.cpp`, situados en la carpeta que les corresponda)
4. Comprobar el funcionamiento del T.D.A MaxQueue utilizando el programa que se proporciona en la práctica. Dicho programa itera sobre los argumentos de entrada que acompañan a la ejecución, haciendo lo siguiente:
    a. Si el elemento es numérico, lo inserta en la cola
    b. Si el elemento es un punto, saca de la cola un elemento

El código de dicho programa está implementado en el archivo `cola_max.cpp`, que se os proporciona comentado para evitar problemas de compilación. Una vez implementado el TDA, se debe descomentar este código para comprobar el funcionamiento. El código del programa es el que sigue:

```
int main (int argc, char ** argv) {
      MaxQueue queue;
      for (int i=1; i<argc; i++) {
            char * v = argv[i];
            if (v[0] == '.'){
                  cout << stack.front() << endl;
                  stack.pop();
            }
            else{
                  int vint = atoi(v);
                  stack.push(vint);
            }
      }
}
```

Por ejemplo, dado el programa anterior, y dada la entrada siguiente:

```
      ./build/cola_max 1 2 . 3 4 . . .
```

La salida del programa debe ser la siguiente:

```
1,2
2,4
3,4
4,4
```

Como podemos observar en dicha salida, cuando sacamos el valor 1, el máximo en la estructura es el valor 2, que es el valor máximo en la estructura en el momento en el que dicho valor se extrae (por el punto de la tercera posición). Cuando se introducen el resto de elementos, el máximo se actualiza, y por dicho motivo a la hora de extraer el 2 el máximo ha cambiado hasta tomar el valor 4. Habrá que tener cuidado en la implementación de dicho máximo, ya que el sistema es un poco más complejo que en el caso de la pila. Existe una implementación en la que la extracción del valor actual y el máximo tienen una complejidad algorítmica O(1), es decir, tiempo constante, y es la implementación que buscamos.

4.- Documentación y entrega

Toda la documentación de la práctica se incluirá en el propio Doxygen generado, para ello se utilizarán tanto las directivas Doxygen de los archivos .h y .cpp como los archivos .dox incluídos en la carpeta doc.

1. La documentación debe incluir:
   - TODOS los métodos y clases debidamente documentados con la especificación completa. Se valorará positivamente descripciones exhaustivas.
   - Las dos tareas de la práctica tienen un ejecutable asociado que debe ser descrito en la página principal o en una página *ad hoc*.
   - Todos las páginas de la documentación generada deben ser completas y estar debidamente descritas.

2. A considerar:
   - Deben respetarse todos los conocimientos adquiridos en los temas de Abstracción y Eficiencia. En especial el **principio de ocultamiento de información** y las distintas estrategias de **abstracción**.
   - Una solución algorítmicamente correcta pero que contradiga el punto anterior será considerada como errónea.
   - Una solución algorítmicamente correcta pero que no utilice el contenedor subyacente requerido en cada caso (es decir, debemos implementar MaxQueue utilizando pilas, y MaxStack utilizando colas), también será considerada como errónea.
   - En alguno de los métodos a implementar, será necesario tener una estructura de datos auxiliar en la que almacenar información intermedia. Dichos contenedores intermedios deberán ser del mismo tipo que el contenedor principal en cada caso (es decir, si necesitáis utilizar contenedores intermedios mientras implementáis MaxStack, todos ellos deberán ser colas, mientras que si utilizáis contenedores intermedios mientras implementáis MaxQueue, todos ellos deberán ser pilas). A modo de sugerencia, la mejor implementación que hemos encontrado en ambos casos necesita **dos** contenedores, el contenedor principal, y uno auxiliar, que se utiliza sólamente en **uno** de los métodos que tenéis que implementar (para cada clase).

3. Código:
   - Se dispone de la siguiente estructura de ficheros:
     - /
       - estudiante/ (Aquí irá todo lo que desarrolle el alumno)

        - doc/ (Imágenes y documentos extra para Doxygen)
        - include/ (Archivos cabecera)
        - src/ (Archivos fuente)
    - CMakeList.txt (Instrucciones CMake)
    - Doxyfile.in (Archivo de configuración de Doxygen)
    - juez.sh (Script del juez online [**HAY QUE CONFIGURARLO**])
- Es importantísimo leer detenidamente y entender qué hace CMakeList.txt.
- El archivo Doxyfile.in no tendríais por qué tocarlo para un uso básico, pero está a vuestra disposición por si queréis añadir alguna variable (no cambiéis las existentes)
- juez.sh crea un archivo submission.zip que incluye ya TODO lo necesario para subir a Prado a la hora de hacer la entrega. No tenéis que añadir nada más, especialmente binarios o documentación ya compilada.

4. Elaboración y puntuación
    - La práctica se realizará POR PAREJAS. Los nombres completos se incluirán en la descripción de la entrega en Prado. Cualquiera de los integrantes de la pareja puede subir el archivo a Prado, pero SOLO UNO.
    - Desglose:
        - **(0.30)** Ejercicio 1
        - **(0.30)** Documentación
        - **(0.15)** Ejercicio 2 (OPCIONAL)

    - La fecha límite de entrega aparecerá en la subida a Prado.