Estructura de Datos y Algoritmos

ITBA 2024-Q2

Problemas

Ejemplo 1: Recursos compartidos

Hay un único recurso (impresora) y múltiples clientes que llegan asincrónicamente y precisan usarlo . A medida que el único recurso se libera, puede tomar otro pedido.

¿Qué estructura auxiliar nos permite implementar esta característica?

Cola de impresión

Pista de aterrizaje/pista de despegue

Problemas

Ejemplo 2: Múltiples Recursos compartidos

Similar al anterior, pero con múltiples recursos compartidos **administrados centralizadamente**. Ej: una solo lugar para acceder a las múltiples cajas para cobrar.

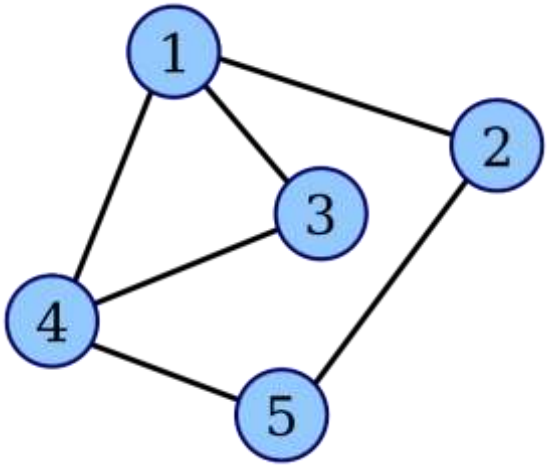
¿Qué estructura auxiliar nos permite implementar esta característica?

Problemas

Ejemplo: Algoritmos en Grafos

Algoritmos como BFS precisan estructuras auxiliares para posponer los momentos de procesamiento mientras se visitan los elementos del grafos.

¿Qué estructura auxiliar nos permite implementar esta característica?



Problemas

Ejemplo: Pipes

Un Pipe es un mecanismo para comunicar 2 procesos donde un proceso escribe en el pipe y otro proceso lee la información en el orden en que fue escrita, en forma secuencial. Ambos procesos abren su canal de comunicación en ambos extremos simultáneamente. Son procesos independientes (asincrónicos) con su propia velocidad (de producción o lectura) por eso se precisa de una estructura auxiliar.

¿Qué estructura auxiliar nos permite implementar esta característica?

Ejemplo

En Linux, si tengo 2 archivos a.txt y b.txt

hola que tal
adios

No se

Ejemplo

En Linux, si tengo 2 archivos a.txt y b.txt

hola que tal
adios

No se

$ cat *.txt

hola que tal
adios
No se

Ejemplo

En Linux, si tengo 2 archivos a.txt y b.txt

hola que tal
adios

No se

$ cat *.txt | wc -l

3

Ejemplo

En Linux, si tengo 2 archivos a.txt y b.txt

hola que tal
adios

No se

$ cat *.txt | grep "hola"

hola que tal

Ejemplo

En Linux, si tengo 2 archivos a.txt y b.txt

hola que tal
adios

No se

$ cat *.txt | grep "hola" | wc -c

13

Queue

Definición

Colección de datos ordenada por orden de llegada. La única forma de acceso es por medio de dos elementos distinguidos: FIRST indica cuál es el más antiguo de los elementos de la colección y tiene prioridad para salir, y LAST marca el elemento más reciente que ha llegado a la colección

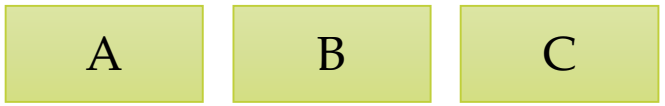
Las operaciones que debe ofrecer son:

- **queue(element)**: agrega un elemento a la colección convirtiéndolo en el más reciente o sea, se convierte en el nuevo LAST.
- **deque()**: quita el elemento más antiguo de la colección (FIRST) y cambia el FIRST. Es una operación destructiva y solo puede usarse si la colección no está vacía.
- **peek()**: devuelve el element más antiguo de la colección (FIRST) sin removerlo (sin cambiar el FIRST). No es destructiva. Solo puede usarse si la colección no está vacía.
- **isEmpty()**: devuelve true/false según la colección tenga o no elementos.
- **size():** (opcional) devuelve la cantidad de elementos de la colección y es ideal para estimar cuánto hay que esperar por ser atendido.

Queue

myQueue.queue(A)

myQueue.queue(B)



myQueue.queue(C)

myQueue.dequeue() → A

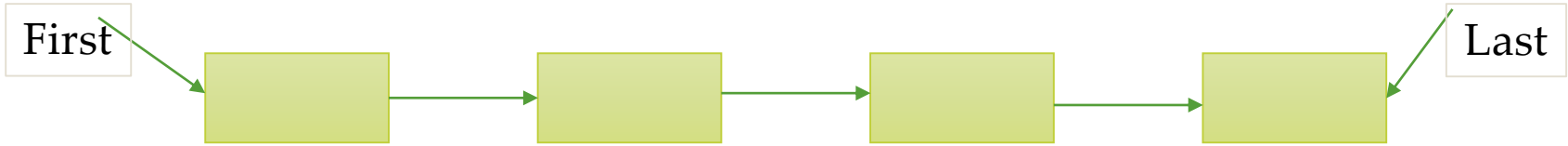
myQueue.peek() → B

Queue: su implementación

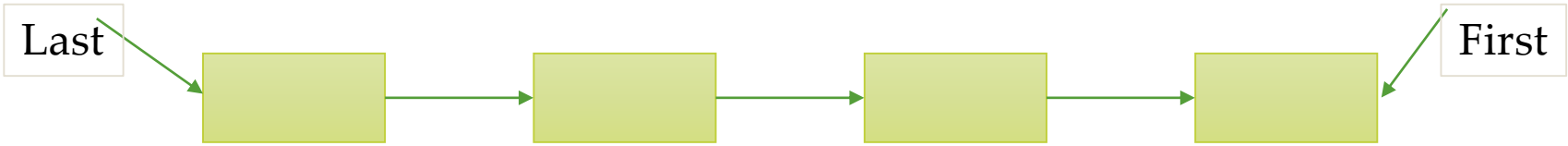
¿Puede implementarse con una lista simplemente encadenada? ¿ Con un arreglo?

Lista

- Opción 1: Si la estructura subyacente fuera una lista lineal simplemente encadenada, ¿Dónde conviene hacer que se encuentre el First y el Last para que las operaciones encolar y desencolar sean O(1)?
- 1-A)



- 1-B)



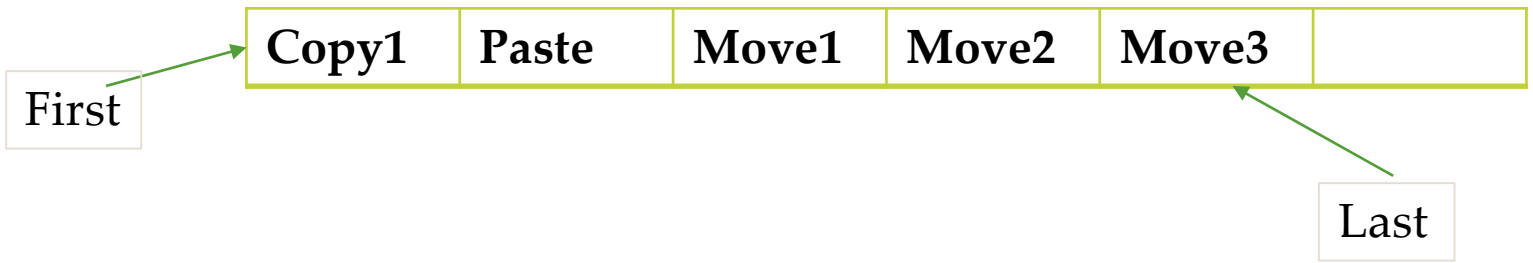
Lista

¿Tenemos el problema de navegación?

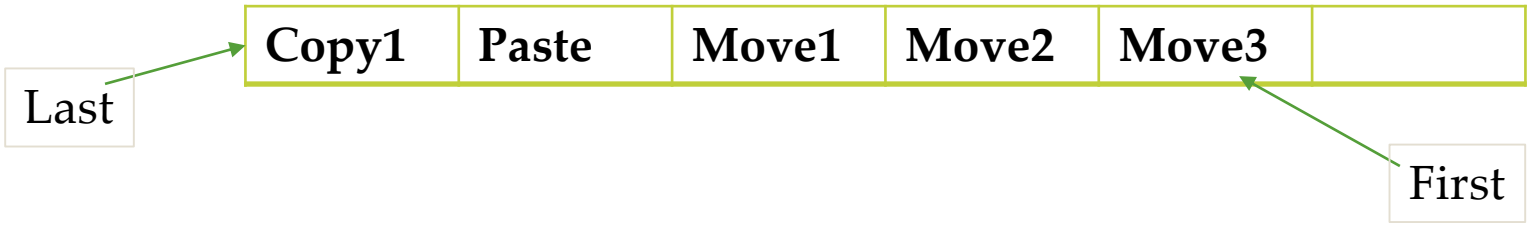
No. Los elementos en la lista solo se acceden por el FIRST o LAST. Es solo cuestión de "apuntarlos" convenientemente. Ojo!

Arreglo

- Opción 2: Si la estructura subyacente fuera una Arreglo, ¿Dónde conviene hacer que se encuentre el First y el Last para que las operaciones encolar y desencolar sean O(1)? ¿Observan algún problema?



- O bien



Arreglo

¿Tenemos el problema de “movimiento de datos”?

En un arreglo es un problema tener espacio libre y no usarlo.

Si se le acaba espacio, hay que realocar (se vuelve O(N))

Eso ocurre en unbounded Queue.

No ocurre en bounded Queue.

Observaciones

Si se tiene una unbounded Queue (no hay límite en la cantidad de elementos que puede manejar), LinkedList es superior a ArrayList.

Pero… si se trata de un bounded Queue (hay límite y podría arrancar con ese tamaño pre alocado porque nunca crecerá) se puede realizar una implementación de las operaciones de encolar y desencolar en O(1) también.

Hay que hacer un tratamiento "circular" de un arreglo para aprovechar al máximo ese espacio pre alocado.
Agregar el método private isFull() para chequear si se puede o no seguir encolando.

Queue

¿Cuál de las 2 implementaciones garantizar O(1) en las operaciones y por lo tanto, resulta más conveniente?

Para unbounded queue: lista lineal simplemente encadenada.

Para bounded queue es indistinto.

Ejercicio

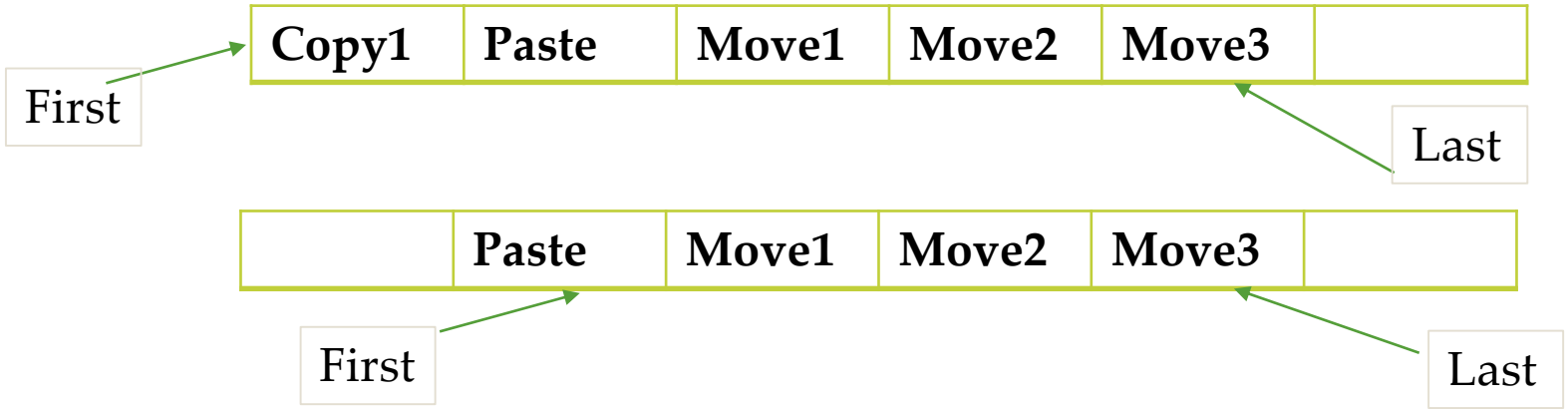
Implementar un bounded queue con un arreglo estático.

Ejercicio

Implementar un bounded queue con un arreglo estático.

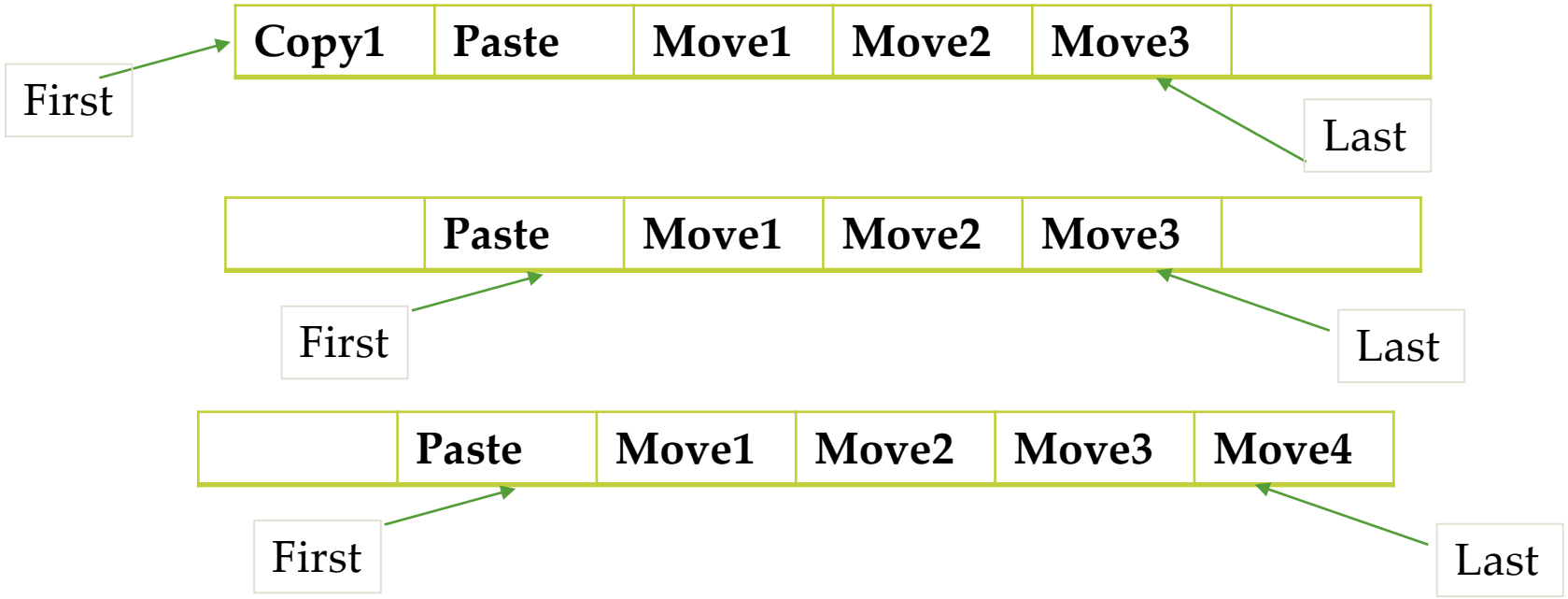
Ejercicio

Implementar un bounded queue con un arreglo estático.



Ejercicio

Implementar un bounded queue con un arreglo estático.



Caso de uso

```
BoundedQueue<Integer> myQueue = new BoundedQueue<>(10);

myQueue.enqueue(10);
myQueue.enqueue(20);
myQueue.enqueue(30);
myQueue.enqueue(40);

System.out.println(myQueue.dequeue() );
System.out.println(myQueue.dequeue() );

myQueue.enqueue(50);
myQueue.enqueue(60);
myQueue.enqueue(70);

System.out.println("\nquedaron 5 elementos");
myQueue.dump();
```

```
10
20
```

```
30
40
50
60
70
```

```
public class BoundedQueue<T> {

    private T[]  elements;
    private int first;
    private int last;
    private int qty= 0;

    public BoundedQueue(int limit) {  // TODO
    }

    public boolean isEmpty() {
       return qty==0;
    }

    public boolean isFull() {
       return qty==elements.length;
    }

    public void enqueue(T element) {       // TODO
    }

    public T dequeue()  {          // TODO
    }

    private void dump() {  // TODO
    }
}
```

Queue

Java no viene equipada con una clase Queue sino que es una interface que limita las operaciones de la clase LinkedList

http://hg.openjdk.java.net/jdk8/jdk8/jdk/file/687fd7c7986d/src/share/classes/java/util/Queue.java

Las operaciones ofrecidas son similares a las discutidas y son las esperadas para una Queue.

Queue

Típico uso en Java

```
public static void main(String[] args) {

// Queu es interface
// LinkedList es clase
    Queue<String> myQueue = new LinkedList<>();


    // add() instead of queue()
    myQueue.add("copy 1");
    myQueue.add("paste 1");
    myQueue.add("move 1");
    myQueue.add("move 2");
    myQueue.add("move 3");

    // remove() o bien poll()    insetad of dequeue();
    myQueue.remove();

    myQueue.forEach(System.out::println);

}
```